# Accounting

**Uday S. Murthy**
*Texas A&M University*

## GLOSSARY

**accounting software package** Software comprising standardized modules for processing accounting transactions for each of the major accounting cycles, such as sales, purchases, inventory, receivables, payables, payroll, and general ledger.

**business process modeling** Creation of tempates or diagrams with standardized symbols for representing various aspects of business activities.

**control procedures** Mechanisms or activities for preventing or detecting the occurrence of errors and irregularities in data.

**database** An organized repository of data with functionality for adding, deleting, updating, and retrieving the data.

**enterprise-wide system** An information system spanning all the major functional areas within an organization, enabling seamless integration of business processes across the functional areas.

**entity-relationship diagram** A widely accepted modeling convention for representing business entities and the relationships between them.

## I. ACCOUNTING INFORMATION SYSTEMS DEFINED

Accounting information systems primarily focus on fulfilling the accounting information needs of an organization's internal and external users. The *accounting information system* in an organization is designed to take business transactions and events as data inputs and generate a variety of financial reports as information outputs. Until the advent of computers and the information technology revolution of the last few decades, the accounting process was performed manually, using the centuries-old double-entry bookkeeping system. With the information technology revolution of the past few decades, however, manual bookkeeping has become defunct. Even very small organizations can afford to automate their accounting system using low-cost, off-the-shelf software. However, many automated accounting systems still use the double-entry model as the basis for accounting. If accounting is seen as a system of providing information useful for decisionmaking, traditional accounting information systems that focus only on financial information and provide only periodic highly aggregated data can fall short of completely meeting the needs of users internal and external to an organization. Internal users include employees at all levels from top management to the lowest level worker who has a legitimate need for information. External users comprise investors, stockholders, creditors, customers, government and regulatory agencies, and financial institutions. Both internal and external users are increasingly demanding instantaneous user-friendly access to relevant and reliable information. Modern accounting systems must be designed to fulfill a wide range of users' needs with reliable information that is available on-line on demand.

## II. TRADITIONAL AUTOMATED ACCOUNTING INFORMATION SYSTEMS

The first generation of computerized accounting information systems continued using the traditional accounting methodology, except that computer technology replaced manual paper-and-pencil journals and ledgers. Computer files replaced journals and ledgers, while printed reports and screen displays replaced manually created reports. Thus, rather than "reengineering" the accounting system, computers were used simply to automate manual accounting systems. Of course, computerization resulted in much greater speed and accuracy in the accounting process. In the early days of computer-based accounting (the 1960s and early 1970s), the typical approach was to develop custom transaction processing applications using common business-oriented language or COBOL. Programmers developed COBOL programs for processing the accounting transactions in each of the organization's accounting cycles—revenue (sales, billing, collections, returns), procurement (ordering, receiving, recording purchase liability, payment), conversion (production), payroll, and general ledger. Users could not easily modify existing reports and had no ability to generate ad hoc reports on an "as needed" basis.

In the late 1970s and even more so in the 1980s and 1990s, organizations increasingly adopted off-the-shelf accounting software packages rather than custom building their own COBOL-driven systems. An accounting software package has a predefined user interface, chart of accounts system, various transaction processing options, and report generators. While most packages can be customized and modified to varying degrees, an organization implementing an accounting package must adapt somewhat to the idiosyncrasies of the package. For example, the package may generate sales analysis reports broken down by salesperson or by region, but not broken down by salesperson within a region. Thus, an off-the-shelf package may satisfy a firm's bookkeeping requirements, but typically will not satisfy the need for nonstandard or ad hoc reports. The chief advantage of accounting packages, relative to custom-developed accounting systems, is their low cost and superior reliability.

Both custom-developed COBOL programs and off-the-shelf accounting packages are geared toward automating the process of recording accounting transactions. For example, an accounting package will typically accept the following revenue cycle accounting transactions: cash sales, credit sales, collections on account, and sales returns and allowances. In a custom-developed environment, the same transactions are typically handled by a series of COBOL programs. Thus, computerized bookkeeping retains an "accounting transactions orientation" with a focus on financial data alone. Business events such as a salesperson contact with a potential customer or vendor quotations for products will find no place in a standard accounting software package. Figure 1 depicts the functioning of the sales and accounts receivable portion of an accounting software package. A custom-developed COBOL program for handling sales and accounts receivable would function in a similar manner.

As shown in Fig. 1, the "Sales and Accounts Receivable Module" is designed to accept the typical set of accounting transactions—credit sales, cash receipts from customers, and sales returns and allowances. Through proprietary file management software (the most common of which is the Btrieve structure for microcomputer-based packages), input transactions are recorded in computer files. However, it is important to recognize that the data in these files are accessible only through the file management software. Thus, reports that can be generated are only those that are already programmed into the software package, or designed by the programmer (in the case of custom-developed software). The basic structure of the system, particularly the separate books (files) for each cycle, is not fundamentally altered. Users still did not have the ability to generate custom reports that accessed data in separate subsystems. For exam-

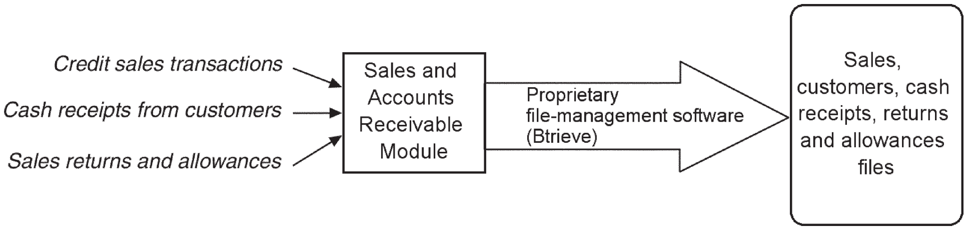

Credit sales transactions
Cash receipts from customers
Sales returns and allowances
Sales and Accounts Receivable Module
Proprietary file-management software (Btrieve)
Sales, customers, cash receipts, returns and allowances files

**Figure 1** Structure of traditional accounting software packages.

**Table I** Advantages and Drawbacks of Traditional Automated Accounting Systems

| Advantages | Drawbacks |
|---|---|
| Automation of tedious manual accounting tasks (e.g., posting to ledgers) | Accounting transactions orientation only; non-financial events not recorded |
| Speed and accuracy | Periodic rather than "real-time" reports |
| Low cost (accounting packages only) | Limited flexibility in generating "ad-hoc" reports |
| Automatic generation of standard accounting reports for common needs (e.g., sales analysis reports, customer statements, financial statements). | Data accessible only through proprietary file management systems |
| Redundant data storage permits efficient generation of certain standard accounting reports | Cross-functional queries difficult to answer |

ple, since customer and vendor data are typically maintained in separate subsystems, it is difficult to determine which customers are also vendors and vice versa. The advantages and drawbacks of computerized bookkeeping are summarized in Table I.

## III. MODERN DATABASE-ORIENTED INTEGRATED ACCOUNTING SYSTEMS

The need for real-time information about enterprise-wide business events and the advent of database technology has led to a more modern view of accounting information systems. The new view of accounting reverts to the fundamental definition of accounting as the main information providing function within organizations. Ideally, as business events transpire, the accounting system should collect and store data about all aspects of those events. Users at all levels in the organization could then obtain instant real-time reports regarding the results of those business events. Data should be stored at the most elemental level, with all aggregation and summarization being left to individual users. The data stored should not be limited to financial data. Most importantly, *all* event data should be stored in a single integrated repository (i.e., a database). This approach visualizes an enterprise-wide information system for an organization spanning departmental and functional lines. Indeed, the so-called enterprise resource planning (ERP) systems that have been adopted by a large number of Fortune 500 companies implement exactly such a vision.

There are two distinct approaches to constructing a database-driven enterprise information system. The first approach is to use relational database tools to custom-develop a suite of applications to meet the organization's information needs, with all data being stored in a relational database like Oracle, Sybase, IBM's DB2, or Microsoft's SQL Server. The second approach is to implement ERP software from vendors such as SAP, Oracle, J.D. Edwards, and PeopleSoft. ERP systems like SAP's R/3 system provide the benefits of an off-the-shelf package—they are reliable and cost less than a completely custom-developed system. However, installing an ERP system in a large multinational organization can literally take years and several million dollars, since there are thousands of configurable switches that must be painstakingly set to meet the needs of the business. Moreover, businesses often reengineer their business processes as they replace their age-old ("legacy") systems with an ERP system, increasing the cost and complexity of the ERP implementation even further. With either the custom-developed path or the ERP path, the key advantages of this modern approach to accounting information systems are (1) the capturing of data about all significant business events and (2) the storage of that data in an integrated enterprise-wide database accessible by *all* users and that provides *all* information necessary to run the business.

### A. Key Features of Modern Accounting Systems

**Events orientation**—In contrast to the "transactions" orientation of traditional manual and automated systems, modern accounting systems focus on capturing data about business events. All relevant business events should be captured, preferably as the events transpire. A call from a potential customer, a complaint from an existing customer, and a suggestion from an employee are some examples of relevant business events that have no place in a traditional transactions-oriented accounting system. However, most managers would argue that information about customer complaints is valuable and should be stored in the organization's information system. The "transactions" mentality in traditional systems focuses on dollar amounts and the accounts to be debited and credited. The events orientation simply seeks to record data about all events in a system that is not constrained by the debit-credit rules of double-entry bookkeeping.

**Enterprise-wide repository**—A key feature in modern accounting systems is the storage of all entity and event data in a single integrated repository. Such a repository, in practical terms, is the enterprise's database. The database is "integrated" in a cross-functional sense—data about events occurring in one functional area (e.g., sales order processing) are captured in the database and can automatically trigger events in another functional area (e.g., shipping). Furthermore, all organizational users that need certain information (e.g., a customer's contact information) have instant access to that information which is stored only once and at one location. The design of the enterprise's database is a critical issue that would determine the degree of success of the database approach. Is the database comprehensive enough? Have appropriate links been established between related sets of data within the database? Are data about all aspects of events stored in the database? Affirmative answers to these questions would indicate a successful design of the event repository. The enterprise repository approach, which is at the heart of modern accounting systems, is illustrated in Fig. 2.

## B. Core Concepts and Building Blocks of Database-Oriented Accounting Systems

In creating a database-oriented, enterprise-wide accounting information system, a number of design decisions must be made relating to field formats, keys, and table types. Some core concepts and basic building blocks of database-oriented accounting systems are now discussed.

### 1. Hierarchy of Data

At the lowest level, all computer systems store data in the form of *bits.* A bit is a binary digit and can take a value of either 0 (turned off) or 1 (turned on). In essence, a bit is turned on by a tiny electrical impulse and is turned off when the impulse is discharged. A group of bits forms a *byte.* In most computer systems, eight bits form a byte. A byte is roughly equivalent to a *character,* which is the lowest element understandable by humans. A group of related bytes forms a *field.* Thus, the group of bytes that forms the word "Jones"

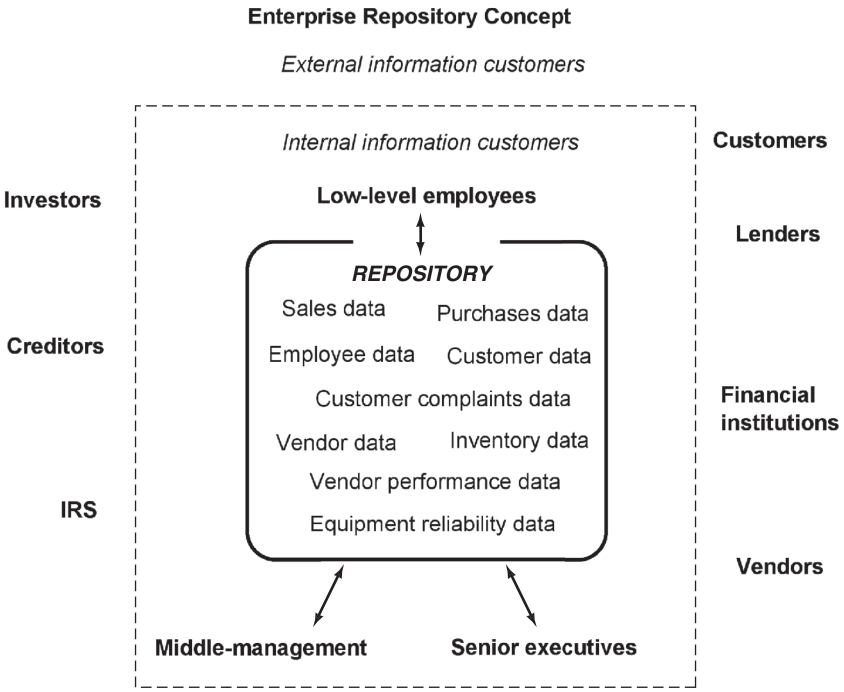


**Figure 2** Enterprise repository concept.

makes up a "name" field. A group of related fields makes up a *record,* or in relational database terminology, a *row* in a table. Tom Jones, Marketing, Sales Associate, 9/1/99, $2800 are a group of fields which make up an employee record showing the employee's name, department, grade, date hired, and monthly salary. A group of logical records constitutes a *file,* which is equivalent to a *table* in relational database terminology. All employee records (rows) taken together would make up the employee file (table). A collection of sales invoices for April 2000 would form the sales file, or table, for April. Finally, a collection of logically related files/tables is a *database.* All files/tables relating to Division A's operations would constitute the "Division A database."

## 2. Field Formats

Although field format decisions are usually not irreversible, giving some forethought to the kind of data that might be stored in each field is time well spent. Most database systems support the following formats for fields in a table: *numeric, text, currency, date/time,* and *Boolean.* Some systems also support counter fields and fields capable of storing binary objects such as a graphical image, a video clip or a sound recording. As the name suggests, *numeric* fields can only store numbers. In contrast, *text* fields can store any keyboard character. Moreover, these text fields can store any character in the complete character set used for a particular computer. All the characters (numeric and non-numeric) in the character set for a computer are called *alphanumeric* characters. *Currency* fields are used for storing dollars and cents appropriate formatted. *Date/time* fields can store dates or times in a variety of user-specified formats (e.g., MM/DD/YYYY or DD/MM/YYYY). *Boolean* fields can hold only one of two possible values—either true or false, or in some computer systems, yes or no. *Counter* fields are automatically incremented by one whenever a new record in the file is created. The user cannot update the value in the counter field. Fields capable of storing binary large objects, or BLOBS, are very useful for present day information systems which are increasingly becoming multimedia. No longer are information systems limited to storing text and numbers. Other forms of data such as a photograph of an employee or a video image of a product can also be stored in tables.

## 3. Keys

In addition to selecting appropriate formats for fields in a table, another critical design decision is the designation of the *primary key* for the table. The primary key is the field or set of fields that uniquely identify the rows (records) in a table. For example, in an employee table, the social security number (SSN) could serve as the primary key because every employee would have a unique SSN. In most tables, the primary key is obvious. Customer tables typically have a customer number field that would be the primary key. The sales invoice number would be the primary key in a table of sales invoices. Some tables may require more than one field to uniquely identify records in the file. Consider a table used to store customer complaints. Assume that the customer number of the customer who makes each complaint is recorded. In addition, the date and time of the complaint is also recorded. However, no separate "complaint number" is generated. Assuming that a customer could have more than one complaint, then the customer number alone would not uniquely identify rows (records) in the table. However, if we assume that a customer will not have more than one complaint in a day, then the combination of customer number and date would uniquely identify the rows in the complaints table. If we assume that customers could make more than one complaint in a day, then the combination of customer number, date, and time, would uniquely identify the rows in the table. When a combination of fields is needed to uniquely identify records in a table, the primary key is called a *composite key* or a *concatenated key.* While primary keys uniquely identify records in a table, the other fields or attributes in the table, called "nonkey attributes" may be used to facilitate user queries. Consider a table of sales invoices with the following fields: invoice number, date, customer number, salesperson number, sales region, and sales amount. The primary key of this table is the invoice number. Each of the remaining fields is a nonkey attribute that can be used to sort the table to respond to a user request for information. Thus, the sales manager might want a listing of sales invoices by date, by salesperson number, or by sales region. These requests could be handled by sorting the table by each of the relevant fields.

The enterprise database of a large organization may contain thousands of tables, each of which is related to at least one other table. In a relational database system, the interrelationships between tables are implemented using common fields across tables. These common fields are referred to as *foreign keys.* A foreign key is a field in a table, which is the primary key in a related table, referred to as the "master table" in the relationship. It is important to identify foreign keys in a database because they enable linking of tables that are related to one another. From the viewpoint of database integrity, foreign key values must always refer to an existing value in the "master table" for that foreign key. In database terminology, this requirement is referred to as "referential integrity" and can usually be enforced by the database management system itself.

### 4. Relationship Types

Relationships between entities and events, both of which are represented in a relational database by means of tables, can be of three types, referred to as the *relationship cardinality*. One-to-one (1:1), one-to-many (1:M), and many-to-many (M:M) relationships are the three relationship types. Consider the relationship between the "department" and "manager" entities. A 1:1 relationship between departments and managers implies that each department can have one and only one manager and each manager can manage one and only one department. Now consider the relationship between the "salespersons" and "customers" entities. A 1:M relationship between salespersons and customers means that each salesperson can have many customers but every customer is assigned to exactly one salesperson. Note that a 1:M relationship can be interpreted as an M:1 relationship when read from the opposite direction. Thus, the relationship from customers to salespersons is an M:1 relationship (many customers have one salesperson). An M:M relationship between salespersons and customers indicates that each salesperson can have many customers *and* each customer can work with many salespersons.

### 5. Table Types

Depending on the contents of a table it can be classified under one of the following categories: master, transaction, reference, history, and backup. *Master tables* contain relatively permanent information and are analogous to ledgers in a manual accounting system. Customer, employee, vendor, and inventory tables are examples of master tables. *Transaction tables* contain relatively temporary information and are analogous to journals in a manual accounting system. Examples of transaction tables include those for storing typical business transaction data, such as sales, purchases, payroll vouchers, and cash receipts, and also non-accounting business event data such as customer suggestions and complaints, equipment failure reports, etc. *Reference tables* are used to store relatively permanent information that is needed only for reference purposes during the process of updating one or more master tables. Tax tables and price list tables are examples of reference tables. *History tables* are old transaction and master tables that are maintained only for reference purposes and for legal reasons. An example of a history table would be the July 1998 sales invoices table. These tables are usually maintained off-line as archive files. The last table type is *backup tables*. As the name suggests, backup tables are duplicate copies of transaction, master, and reference tables (since history tables are typically maintained only for legal reasons, most organizations would typically not maintain backup copies of history tables).

## C. The Data Processing Cycle

Given a basic understanding of the building blocks of database-oriented enterprise accounting systems as presented above, the data processing cycle can now be discussed. All information systems applications undergo a sequence of steps from data input to information output. This process, called the "data processing cycle," comprises the sequence of steps from data capture to the generation of meaningful information for end users. Specifically, the steps in the data processing cycle are data input, data preparation, data processing, table maintenance, and information output.

*Data input* involves collection of data and converting data into computer-readable form. Data input results in the creation of a transaction data. While in the past transaction data was first captured on paper and was then keyed into the computer system, newer technologies such as bar code scanners facilitate automatic entering of data. *Data preparation* may be needed to facilitate data processing and maintenance of tables. Data preparation entails two main steps: (1) *validating* input data to filter out erroneous transactions, and (2) *sorting* input data to facilitate the process of updating master tables, *if* the update is to be performed in "batch" mode on a periodic basis rather than "on-line" instantaneously. *Data processing* represents the next step in the data processing cycle. This step includes all the calculations, comparisons, and manipulations that are undertaken in the particular application. For example, in a sales application system, data processing might involve calculating the sales tax payable on each invoice. *Table maintenance* is the next step in the data processing cycle. This is the step where the master table(s) is(are) actually updated using transaction data. For example, customer balances in a customer table would be updated (in-

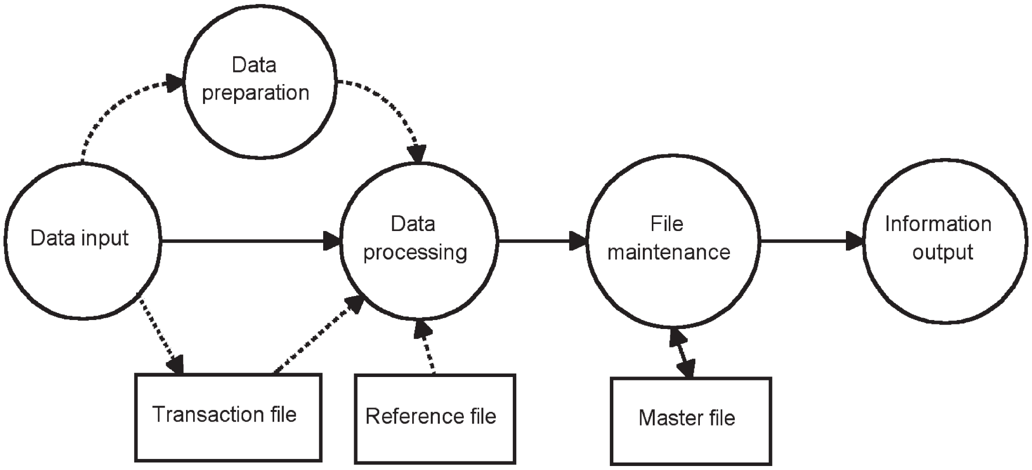


**Figure 3** Data processing cycle.

creased) to reflect new credit sales transactions for customers. Although data processing and table maintenance are shown as distinct steps in the data processing cycle, they are performed simultaneously in on-line database-oriented systems. *Information output* is the last step in the data processing cycle. This step is where reports are generated either on paper or on the user's computer screen. For example, in sales application systems, the information output step could result in the following reports: sales analysis report, salesperson performance report, product turnover analysis, and customer statements of account. The data processing cycle is shown in Fig. 3, along with the table type(s) accessed at each stage. Note in Fig. 3 that the transaction table is accessed during the data input step in order to store input transactions and the master table is accessed during the table maintenance step in order to read and update the appropriate row in the master table.

## IV. ACCOUNTING SYSTEMS FOR REVENUE (SALES)

Using the basic building blocks discussed above, we focus now on accounting systems for meeting specific business needs. Every business will have slightly different business processes and it is the function of a firm's accounting system to record, track, and report on the business processes unique to that firm. Presented next are an illustrative set of business processes for revenue, assuming a retailing organization. Generating revenue is one of the main processes at all business organizations. The primary events related to generating revenue for most retailing firms are to sell merchandise and to obtain payment from customers. Secondary events for a retailing firm would typically include contacting potential customers, accepting sales orders, shipping merchandise to customers, and dealing with sales returns and allowances. Note that these events comprise both economic events (selling merchandise and collecting payment) and noneconomic events (contacting potential customers). The sales, collections, and returns processing functions may be carried out by one system or a set of related subsystems. In either case, the data underlying the system should ideally reside in one repository, consistent with the notion of the enterprise wide repository discussed earlier.

### A. Business Processes Related to Revenue Generation

At a very general level, questions that the information system for a retailing firm should provide answers for include: How much did we sell? How much do customers owe us? What is the value of sales returns and allowances? What is the total amount of collections we have received from customers? In addition to these general questions, other information needs include reports detailing the performance of salespersons, aging of accounts receivable, evaluating the performance of shippers, determining which products are selling well and which are not selling as well, and determining which products are being returned the most. The flexibility and power of the database approach allows both financial and nonfinancial information needs to be served by one integrated repository of organizational data.

Revenue business processes actually begin with the advertising of merchandise for sale, salesperson contacts with potential customers, and unsolicited customer inquiries. The recording of revenue business processes within the system, however, generally begins with a customer order (i.e., a sales order) for merchandise. However, prior to receipt of a customer order there could be salesperson contacts with potential customers. It is therefore necessary to keep track of contacts between salespersons and potential customers. The extent to which these contacts result in sales orders would be one dimension of salesperson performance which management would be interested in. Contacts with existing and potential customers would also likely be a source of feedback about the company's products and services. As a sales order is being input by a salesperson, it is necessary to first verify that the customer has sufficient credit available.[1] If the customer's credit status does not permit the order to be placed then the customer is simply referred to the credit department for further action. Each customer could obviously place many sales orders. However, each sales order must relate to only one customer. Every sale is executed by one salesperson, who can obviously input many sales orders. Each sales order is for at least one but possibly many items of merchandise inventory. Obviously, a merchandise inventory item could be sold on many orders. It is of course possible that a new inventory item has not as yet been sold on any order, but the company would still like to keep track of that inventory item. Before the order is recorded it is necessary to check whether the on-hand quantity of inventory is sufficient for each requested item. Upon recording of the sale it would be necessary to decrease the quantity on hand of the merchandise inventory items being sold.

The accounting system would generate several copies of the sales order document. The first copy is given (or sent) to the customer as a confirmation of the order. One copy is forwarded to the shipping department. A warehouse clerk in the shipping department packs and ships out merchandise inventory. Each warehouse clerk can process either none or many shipments, but each shipment must be processed by exactly one warehouse clerk. The warehouse clerk prepares a shipping document (e.g., a bill of lading) for each shipment. However, there will be some time lag between the sales order and the shipping of that order. Thus, it is possible that a recorded sales order has not as yet been shipped. Every shipping document, once prepared, must relate to exactly one sales order which has already been recorded. It is necessary to keep track of the inventory items and the quantity of each item actually shipped out on a shipment. Each shipment is assigned to one of several approved shippers with which the company does business. Each shipper could obviously be involved with many shipments. When the shipment is recorded by the warehouse clerk, a sales invoice is automatically generated and sent to the customer along with the merchandise. Note that when the sales invoice is generated, the sale is recorded in the system, but this recording of the sale is simply an *information process* and is not a "significant business event" in its own right.

For simplicity, it is assumed that all sales orders are on credit (i.e., no immediate cash sales). Along with each shipment, the customer is sent a sales invoice and it is then necessary to update the customer's account to increase the outstanding balance (i.e., increase "accounts receivable"). Customers make payments subsequently and each payment relates to a specific invoice resulting from a specific shipment. Collections are taken by cashiers. Each cashier can handle either none (i.e., a newly hired cashier) or many collections. Since collections from customers are received several days or weeks after the sale, it is possible that certain shipped orders do not have corresponding collections. However, a collection must relate to one sales invoice, i.e., one shipment received by the customer. Of course, there can be many collections for a given customer. Upon receipt of a collection from a customer it is necessary to update the customer's account to decrease the outstanding balance. Cash collected from customers must be deposited in the bank. Finally, revenue business processes must also consider the possibility of returns and allowances. It is conceivable that customers could return merchandise. All returns and allowances are processed by returns clerks. Each return clerk can handle either none (newly hired clerk) or many returns. Customers could also be granted an allowance on a sale. Every return or allowance would relate to one shipment that has taken place, and it is possible that a shipment could be associated with none or many returns and allowances. The difference between a return and an allowance is that a return is associated with merchandise inventory whereas an allowance is not. For example, a customer may receive an allowance for slightly damaged merchandise, which the customer decides to keep. In the case of returns, at least one but possible many items of inventory may be returned by the customer. Returns and allowances

[1] The assumption that all sales are on credit is for ease of exposition.

also necessitate updates to the customer's account to decrease the outstanding balance. For returns it would be necessary to keep track of the inventory items returned. Let us assume that the company would like to keep track of regular ("good") inventory and returned (possibly defective) merchandise separately.

## B. Entity-Relationship Data Model for Revenue Processes

Based on the above narrative description of revenue business processes, the entity-relationship (ER) model shown in Fig. 4 is developed. The purpose of ER modeling is to develop a formal representation of the proposed accounting information system, in terms that both nontechnical users and information systems designers can understand. The ensuing ER model can then be implemented in a database system, as discussed later in the chapter. The ER model follows McCarthy's "REA" framework proposed in 1982, which shows three categories of entities (rectangles)—resources on the left, events in the middle, and agents on the right. The lines connecting the entities indicate the type of relationship between the entities, including the relationship cardinality and whether the entity's participation in the relationship is optional or mandatory. Recall that relationships can be 1:1, 1:M, or M:M. In the model below, the "many" side of a relationship is shown using crow's feet and the "one" side of a relationship is indicated with the absence of crow's feet. Also, note that the "|" at the end of a relationship line indicates a mandatory participation in a relationship and an "○" indicates an optional participation in a relationship.

The entities in the middle column in Fig. 4 represent revenue-related events in chronological order from top to bottom. The entities in the column on the left are the organization's resources and the entities in the column on the right are the various agents,

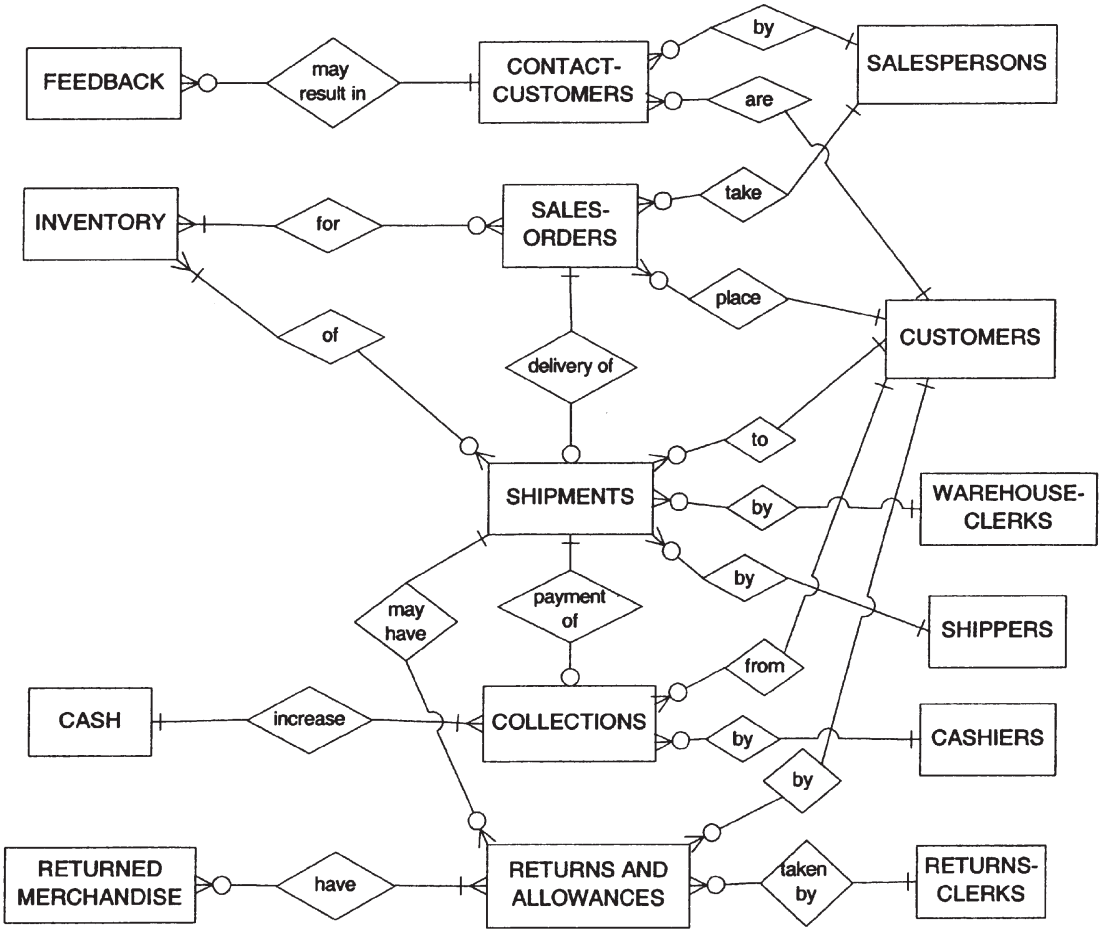


**Figure 4** ER diagram for revenue business processes.

internal and external to the organization, who are involved with events. Taken together, the crow's feet, | for mandatory participation and ○ for optional participation make it possible to interpret the relationship between entities. For example, the relationship between the CONTACT-CUSTOMER event and the SALESPERSON entity would be interpreted as follows: each customer contact must be performed by exactly one salesperson; each salesperson may be involved with either none or many customer contact events. As another example, the relationship between CUSTOMERS and SALES-ORDERS would be interpreted as follows: each sales order must be placed by exactly one customer; each customer may place either none or many sales orders.

## C. Data Repository for Storing Revenue-Related Information

The ER diagram in Fig. 4 depicts the various entities and relationships that must be represented in the enterprise database for storing information related to revenue business processes for the illustrative retailing firm scenario. A standard set of conversion rules are applied to deduce the relational tables that should result from the ER diagram. The conversion rules are as follows: (1) a separate table is created for each entity; (2) attributes are created for each entity and the primary key in each entity is identified; (3) for the purpose of conversion, all "optional" relationships are treated as "mandatory many" relationships; (4) the primary key of the entity on the "one" side of a relationship is posted to the table of the entity on the "many" side of a relationship; and (5) a separate table is created for the relationship itself for entities participating in an M:M relationship with the primary key of each table being posted to the new relationship table to form a composite key. Attributes unique to many-to-many relationships are posted as nonkey attributes in the composite key table. Applying the conversion rules and streamlining the resulting tables to eliminate redundancies and inconsistencies, we arrive at the set of tables shown in Fig. 5. Primary keys are

EMPLOYEES (EMPLOYEE-NO, NAME, DEPARTMENT, GRADE, PHONE#, DATE-HIRED, SALARY, COMMISSION-RATE)

CUSTOMERS (CUSTOMER-NO, NAME, ADDRESS, PHONE#, BALANCE, CR-LIMIT)

CONTACTS (CONTACT-NO, DATE-CONTACTED, EMPLOYEE-NO*, CUSTOMER-NO*)

FEEDBACK (FEEDBACK-NO, CONTACT-NO*, TYPE, REMARKS)

SALES-ORDERS (ORDER-NO, DATE, CUSTOMER-NO*, EMPLOYEE-NO*, AMOUNT)

ITEMS-ORDERED (ORDER-NO*, ITEM-NO*, QTY-SOLD, PRICE)

INVENTORY (ITEM-NO, DESCRIPTION, SELLING-PRICE, QUANTITY-ON-HAND, REORDER-POINT)

SHIPMENTS (SHIPMENT-NO, DATE, ORDER-NO*, SHIPPER-NO*, EMPLOYEE-NO*, SHIPPING-WEIGHT, CHARGE, DATE-DELIVERED)

ITEMS-SHIPPED (SHIPMENT-NO*, ITEM-NO*, QTY-SHIPPED)

SHIPPERS (SHIPPER-NO, NAME, ADDRESS, PHONE#, BALANCE)

COLLECTIONS (RECEIPT-NO, DATE, SHIPMENT-NO*, EMPLOYEE-NO*, ACCOUNT-NO*, AMOUNT)

CASH (ACCOUNT-NO, BANK, BANK-ACCT-NO, BALANCE)

RETURNS-AND-ALLOWANCES (CREDIT-MEMO-NO, DATE, SHIPMENT-NO*, EMPLOYEE-NO*, TYPE, AMOUNT)

ITEMS-RETURNED (CREDIT-MEMO-NO*, ITEM-NO*, QUANTITY-RETURNED, REASON)

RETURNED-MERCHANDISE (ITEM-NO, RETURNED-QUANTITY)

**Figure 5** Tables for revenue processing subsystem.

underlined and foreign keys are indicated with an asterisk at the end of the field.

## D. Explanation of Tables

The tables listed above should correspond to the entities on the revenue ER model. Data in the form of rows (records) in tables will be added, deleted, and updated via application programs. These programs, which could be written in languages such as C++ and Visual Basic, capture business event data, process the data, and update all affected tables. The various screens in a high-end ERP system such as SAP R/3 invoke programs that take data entered on the screen and update the relevant event, resource, and agent tables. The fields in the EMPLOYEES table should be self-explanatory. There are no separate tables for salespersons, warehouse clerks, shipping clerks, and cashiers, although those agents were shown separately on the ER diagram. Data pertaining to all these internal agents can be stored in one employees table, with the "department" field indicating the particular department in which the employee works. The CUSTOMERS table contains typical customer-related information of which the company would like to keep track. The "balance" field in the CUSTOMERS table shows the current balance owed by the customer. Updates to this field occur as follows: (1) increases resulting from credit sales; (2) decreases resulting from collections from customers; and (3) decreases resulting from adjustments such as returns, allowances, and perhaps bad debt write-offs (which we are not considering in our simplified example). The sum total of all "balances" in the CUSTOMERS table equals the company's "Accounts Receivable" at any point in time. The CONTACTS table shows details about contacts between salespersons and customers. The FEEDBACK table is also used to record the actual feedback received from customers (i.e., complaints and suggestions). The SALES-ORDERS table can be thought of as being equivalent to a sales journal in a manual accounting environment. A listing of sales orders for a particular period can be generated by performing a query on the SALES-ORDERS table, specifying an appropriate criterion in the DATE field. The customer to whom the sale is made, and the salesperson who made the sale, can be found out using the foreign keys as the basis for linking the SALES-ORDERS table with the CUSTOMERS and EMPLOYEES tables. The INVENTORY table shows all the items of inventory available for sale, the current price, and the quantity on hand.

Three additional tables are needed for the M:M relationships. These are the ITEMS-ORDERED table (for the M:M relationship between SALES-ORDERS and INVENTORY), the ITEMS-SHIPPED table (for the M:M relationship between SHIPMENTS and INVENTORY), and the ITEMS-RETURNED table (for the M:M relationship between RETURNS-AND-ALLOWANCES and RETURNED-MERCHANDISE). Attributes unique to each of these composite key tables are listed as nonkey attributes in each table. For example, the ITEMSORDERED table has two nonkey attributes: QTY-SOLD, and PRICE. These fields show the quantity of each item sold on each invoice and the price of each item sold on each invoice.

The COLLECTIONS table shows payments on account received in settlement of invoices. Note that the COLLECTIONS table has SHIPMENT-NO as a foreign key. By linking the COLLECTIONS, SHIPMENTS, SALES-ORDERS, and CUSTOMERS tables it is possible to identify the customer who made each payment. As entries are made in the COLLECTIONS table, the appropriate BALANCE field for the customer who made the payment will be updated (decreased). The SHIPMENTS table also indicates the shipper assigned to make the shipment (via SHIPPER-NO). The weight being shipped and shipping charges that apply are nonkey attributes in this table. The ITEMS-SHIPPED table indicates which items were actually shipped on each shipment and the quantity of each item that was shipped. The SHIPPERS table is used to keep track of shipping service providers. Amounts payable to shippers are reflected in the BALANCE field in the SHIPPERS table. Returns and allowances are recorded in the RETURNS-AND-ALLOWANCES table, which has SHIPMENT-NO as a foreign key to indicate the shipment against which the return is being accepted. If the "returns and allowances" entry is simply an allowance, then no entries will be made in the ITEMS-RETURNED table. The amount of the allowance as shown in the AMOUNT field will be used to decrease the customer's balance in the CUSTOMERS table. The CASH table can be thought of as the "cash at bank" resource of the firm. Moneys collected from customers result in an increase in the "cash at bank" resource. Finally, the ITEMS-RETURNED and RETURNED-MERCHANDISE tables are used to keep track of items returned by customers. The quantity returned and the reason for the return (i.e., defective merchandise, customer not satisfied, etc.) are indicated in the ITEMS-RETURNED table. Note that the item description is not repeated in the RETURNED-MERCHANDISE table because the description can be obtained by joining the RETURNED-MERCHANDISE table with the INVENTORY table.

## V. ACCOUNTING SYSTEMS FOR PROCUREMENT (PURCHASING)

The main business processes or events related to procurement for a retailing organization are to order merchandise from vendors and to pay vendors to settle accounts payable. Secondary events include processing requests for merchandise from departments, obtaining information about pricing and availability of merchandise from vendors, receiving and inspecting incoming merchandise, and returning defective or unneeded merchandise back to vendors. As with the discussion pertaining to revenue processes, the procurement business processes outlined here are also illustrative; actual processes would vary across companies. As with the systems for processing revenue activities, the various procurement activities could be performed by one system or a set of tightly integrated systems. In either case, the data underlying the procurement system resides in one repository, with appropriate links to the relevant revenue-related tables that were presented earlier. The essence of an enterprise wide repository is that *all* information needs are met via one integrated database (rather than a set of loosely coupled systems).

### A. Business Processes Related to Procurement/Purchasing

The procurement information system for a retailing organization should be able to answer the following questions: At what prices are various items being offered for sale by various vendors? What have user departments requested for purchase? How much did we purchase? How much do we owe vendors? What is the value of purchase returns? What are the total payments made to vendors? Which purchase orders are outstanding (merchandise not yet received)? Are there any items received for which vendor's invoices have not been received? In addition, management would also like reports about which vendors supply the lowest cost items of each type, vendor performance in terms of on-time delivery of merchandise, quality of merchandise supplied as gauged by purchase returns, and trends in requests for items received from departments within the retail store. Many of these information items do not have a strict "financial" orientation. The design of the database for meeting the needs of procurement business processes should consider *all* information needs and not just financial or accounting needs. At the same time, it is important to ensure that the database design can provide all the necessary accounting reports. The flexibility of the database approach allows both financial and nonfinancial information needs to be easily served by one integrated repository of organizational data.

Procurement business processes for a retailing firm generally begin with a need for merchandise on the part of a department within the store. Let us assume that each department within the store has an employee who is responsible for keeping track of the quantity on hand of each item. This tracking would be done by accessing the INVENTORY table, which was presented in the discussion of revenue-related business processes. In the earlier presentation, the INVENTORY table was being accessed and updated to reflect the *sale* of merchandise. Here, the very same table is being accessed and updated to reflect *purchases* of merchandise. This is the essence of cross-functional integration which is at the heart of complex ERP systems like SAP, PeopleSoft, and J.D. Edwards. The INVENTORY table has "quantity on hand" and "reorder point" fields. In essence, when the quantity on hand falls below the reorder point it is necessary to initiate a purchase requisition for that item. This tracking of the quantity on hand of each item could easily be done by means of a programmed procedure that scans each row in the INVENTORY table. Whether initiated by a programmed procedure or by an employee in each department, a purchase requisition, or a formal request for items to be purchased, must be prepared and authorized. It is this requisition (request for items) that documents a legitimate need for purchases. A purchase requisition would list at least one item needed but possibly many items needed by the department.

Independent of the procedures involved in creating purchase requisitions, personnel in the purchasing department obtain information about the availability of items from different vendors. Quotations from vendors indicate the price of each item being offered for sale. It is important to keep track of this information even though it has no direct "accounting" implication. This information would be valuable when a department issues a purchase requisition. Specifically, the purchasing agent can scan the "items available for sale" table to determine whether any vendor has a special discount on the needed item(s). Keeping such information current is a proactive measure in contrast to a reactive process of soliciting quotations from vendors *after* a purchase requisition has been received. The next step in the purchasing process is to actually issue a purchase order for a particular requisition on an approved vendor. One purchase order is placed for every purchase requisition. Due to the time lag between placing the requisition and the order it is possible that a requisition does not have an associated purchase order. A purchase order can be placed with only one vendor, but a vendor can obviously have many purchase orders. Each purchase order would contain at least one item but possibly many items to be purchased. When the vendor delivers the merchandise a receiving report must be prepared. While there may be many receiving reports for a purchase order, each receiving report can relate to only one purchase order on a specific vendor. However, due to the time lag between placement of the purchase order and receipt of the merchandise it is possible that there is no receiving report for a purchase order. In effect, the purchase orders for which there are no receiving reports constitute “open” purchase orders (i.e., outstanding orders for which the merchandise has not as yet been received). Each receiving report would have at least one but possibly many items received.

Upon receipt of the vendor’s invoice for items delivered, and after the merchandise has been received, a purchase liability can be recorded. As was the case with the sales invoicing process, note that the actual recording of the purchase liability is simply an information process and is not a “significant business event” in its own right. The three prerequisites for the recognition of a purchase liability are (1) a vendor’s invoice, (2) a receiving report, and (3) a purchase order. The invoice represents a claim for payment for merchandise delivered. The receiving report acknowledges that the merchandise was in fact received. Finally, the purchase order indicates that the items received were in fact ordered. In a manual system, the three documents are physically matched before a purchase liability is recorded. In a relational database system, the existence of a valid receiving report and purchase order are determined through foreign keys. Foreign keys in a relational database system in effect constitute an “electronic audit trail.”

Vendors must periodically be paid, and these payments result in a decrease in the firm’s cash resource. Every payment to a vendor is associated with one merchandise receipt, which in turn relates to one purchase order. Of course, since there will be a time lag between receipt of merchandise and the actual payment; it is possible that a merchandise receipt does not yet have an associated payment. However, when merchandise is received and the vendor has submitted an invoice, as already discussed, a purchase liability exists. In effect, the purchase liabilities that are “open” (i.e., unpaid) constitute the company’s accounts payable at any point in time.

The only other procedures related to procurement of merchandise have to do with returns. Defective or damaged merchandise is returned to vendors and they are issued a debit memorandum to that effect. The debit memoranda represent decreases in the accounts payable liability. Each debit memorandum for a purchase return relates to only one merchandise receipt and thus one purchase order. Assuming that there are many items received, it is possible that different items can be returned on several different returns. Obviously, it is possible that a merchandise receipt has no associated purchase returns. Each purchase return will have one or more items returned, and each item can be returned on several different purchase returns. Since a debit memorandum could be issued only to request an allowance without actually returning merchandise, it is possible that a purchase return does not actually have any items returned associated with it. The increases and decreases to the vendor’s balance and to the inventory quantity on hand indicated are simply the “debits” and “credits” to the vendor’s and inventory accounts.

## B. ER Data Model for Procurement/Purchasing Processes

Based on the above narrative description of procurement business processes, the following ER model is developed. Again, note that the purpose of ER modeling is to develop an easily understandable formal representation of the proposed accounting information system as the first step toward building a functioning database-driven system. The ER model shown in Fig. 6 follows the same conventions described earlier, with resource entities on the left, event entities in the middle, and agent entities on the right. Relationships between entities also use the same conventions for indicating mandatory and optional relationships, with crow’s feet being used to indicate the “many” side of a relationship.

The ER model in Fig. 6 is interpreted much the same as the model shown earlier for revenue business processes. The entities in the middle column represent procurement-related events in chronological order from top to bottom, those on the left are the organization’s resources that are affected by procurement activities, and the entities in the column on the right are the agents involved with procurement-

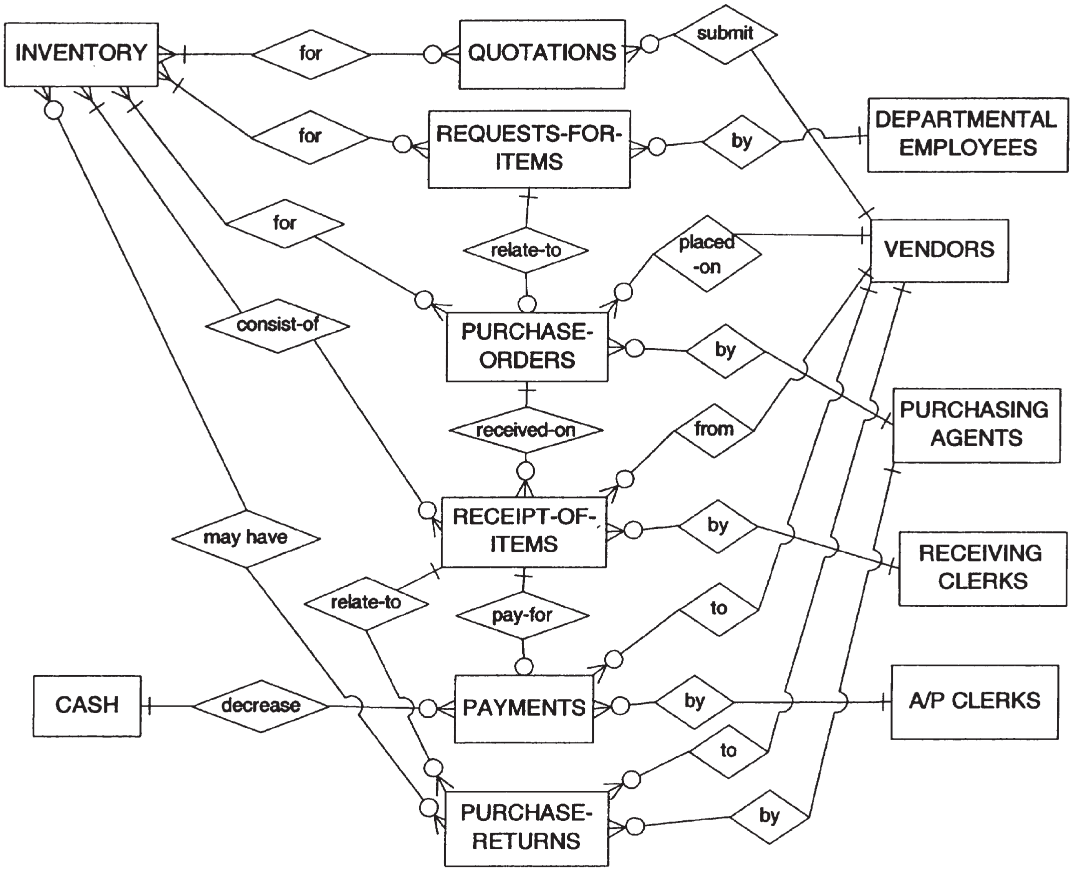


**Figure 6** ER diagram for procurement business processes.

related events. The crow's feet, | for mandatory participation and ○ for optional participation make it possible to interpret the relationship between entities. For example, the relationship between the VENDORS agent and the PURCHASE-ORDERS event would be interpreted as follows: vendors place none or many purchase orders; each purchase order must be placed by exactly one vendor.

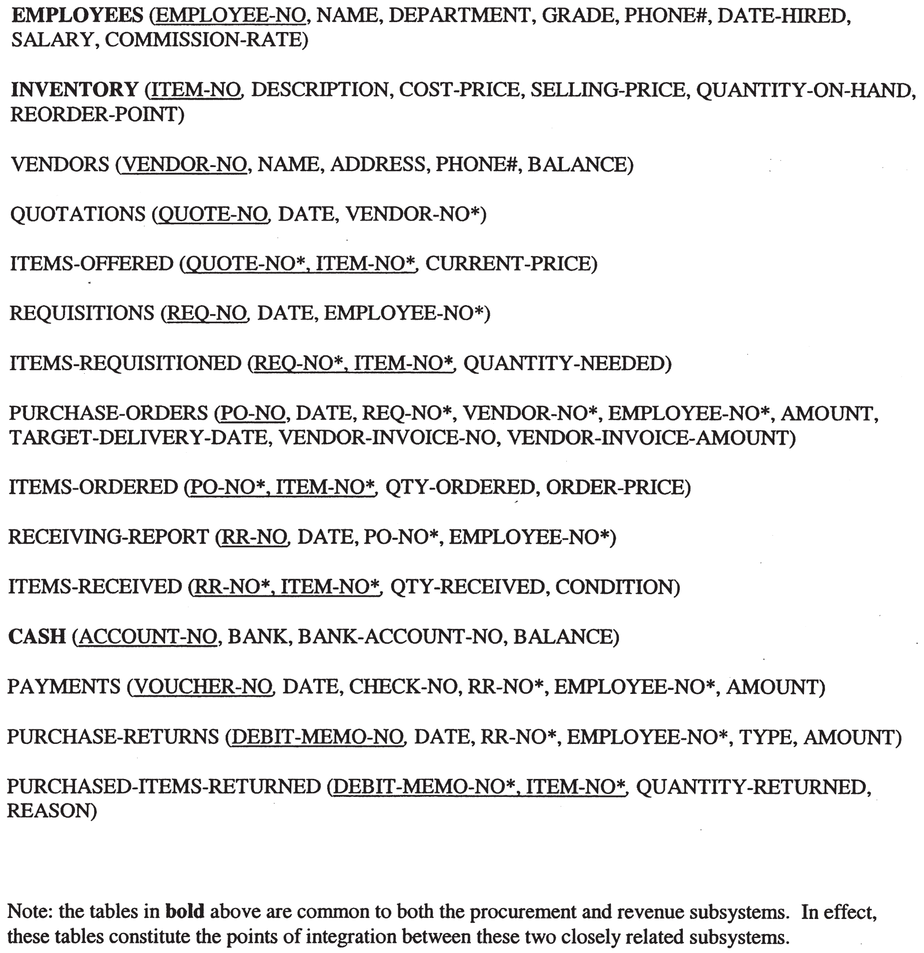
## C. Data Repository for Storing Revenue-Related Information

Using the same rules outlined earlier, the ER model for procurement activities is converted to a set of tables, as shown in Fig. 7. As before, primary keys are underlined and foreign keys are indicated with an asterisk at the end of the field. The tables shown in Fig. 7, and the tables shown earlier as a result of converting the revenue ER model, would be implemented in a relational database management system such as Oracle 8i or Microsoft SQL Server 2000.

## D. Explanation of Tables

As with the tables resulting from conversion of the revenue ER model, the tables shown in Fig. 7 should correspond to the entities on the procurement ER model. In an ERP system, the tables in Fig. 7 would be updated as a result of application programs that process data entered on user screens. Note that the EMPLOYEES, INVENTORY, and CASH tables, highlighted in bold type, are the same tables shown earlier for revenue business processes. This is the essence of cross-functional integration—related business processes share information by accessing the same tables from different applications or modules.

Most of the fields in the tables in Fig. 7 should be self-explanatory. Note that although the various internal agents involved with procurement processes are shown separately on the ER diagram, there is only one employees table. The EMPLOYEE-NO field in various tables indicates the employee who added the entry into the table. How is it possible to ensure that say only employees who are purchasing agents in the



**EMPLOYEES** (EMPLOYEE-NO, NAME, DEPARTMENT, GRADE, PHONE#, DATE-HIRED, SALARY, COMMISSION-RATE)

**INVENTORY** (ITEM-NO, DESCRIPTION, COST-PRICE, SELLING-PRICE, QUANTITY-ON-HAND, REORDER-POINT)

VENDORS (VENDOR-NO, NAME, ADDRESS, PHONE#, BALANCE)

QUOTATIONS (QUOTE-NO, DATE, VENDOR-NO*)

ITEMS-OFFERED (QUOTE-NO*, ITEM-NO*, CURRENT-PRICE)

REQUISITIONS (REQ-NO, DATE, EMPLOYEE-NO*)

ITEMS-REQUISITIONED (REQ-NO*, ITEM-NO*, QUANTITY-NEEDED)

PURCHASE-ORDERS (PO-NO, DATE, REQ-NO*, VENDOR-NO*, EMPLOYEE-NO*, AMOUNT, TARGET-DELIVERY-DATE, VENDOR-INVOICE-NO, VENDOR-INVOICE-AMOUNT)

ITEMS-ORDERED (PO-NO*, ITEM-NO*, QTY-ORDERED, ORDER-PRICE)

RECEIVING-REPORT (RR-NO, DATE, PO-NO*, EMPLOYEE-NO*)

ITEMS-RECEIVED (RR-NO*, ITEM-NO*, QTY-RECEIVED, CONDITION)

**CASH** (ACCOUNT-NO, BANK, BANK-ACCOUNT-NO, BALANCE)

PAYMENTS (VOUCHER-NO, DATE, CHECK-NO, RR-NO*, EMPLOYEE-NO*, AMOUNT)

PURCHASE-RETURNS (DEBIT-MEMO-NO, DATE, RR-NO*, EMPLOYEE-NO*, TYPE, AMOUNT)

PURCHASED-ITEMS-RETURNED (DEBIT-MEMO-NO*, ITEM-NO*, QUANTITY-RETURNED, REASON)

Note: the tables in **bold** above are common to both the procurement and revenue subsystems. In effect, these tables constitute the points of integration between these two closely related subsystems.

**Figure 7** Tables for procurement processing subsystem.

purchasing department add orders to the purchase orders table? A control could be programmed into the system to verify that the "grade" of the employee entering the purchase order is in fact "purchasing agent" and that the employee is in the "purchasing" department.

The "balance" field in the VENDORS table shows the current amount owed to each vendor. The sum total of all "balance" fields in the VENDORS table constitutes current accounts payable. Updates to this field occur as follows: (1) increases resulting from purchase liabilities executed by means of an "update query" (using Access terminology); (2) decreases resulting from payments recorded in the PAYMENTS table; and (3) decreases resulting from purchase returns recorded in the PURCHASE-RETURNS table. The balance field is an example of a logical field (calculated field) whose value can be derived based on other values in the database. Thus, rather than actually storing "balance" as a physical field, its value can be recomputed whenever it is needed by running a query. For expediency, however, most organizations will find it convenient to store this value, to avoid overloading the information system especially when purchase transactions number in the millions.

The ITEMS-SUPPLIED table shows which vendor supplies which items and the current price of the item. It is this table that would be accessed by purchasing department personnel upon receipt of a purchase requisition from a department. The RECEIVING-REPORTS and related ITEMS-RECEIVED tables would be accessed upon receipt of merchandise. The PAYMENTS table is accessed periodically whenever payments are made to vendors. Note the connection between PAYMENTS and CASH—payments to vendors result in a decrease in available cash. The PURCHASE-RETURNS and related PURCHASED-ITEMS-RETURNED tables are accessed when items are returned to vendors. Note that, in the PURCHASE-ORDERS table, the vendor-invoice-no and vendor-invoice-amount fields would be left blank when the purchase order itself is created. These fields are relevant only when the vendor's invoice is actually received. As discussed above, recording the fact that the vendor's invoice is received and that a purchase liability exists is simply an information process. The vendor's invoice should indicate the purchase order to which the invoice relates—this purchase order number is then used to retrieve the appropriate row in the PURCHASE-ORDERS table to update the vendor-invoice-no and vendor-invoice-amount fields. Periodically, a query can be run to determine the rows in the purchase orders table that have values in the vendor-invoice-no and vendor-invoice-amount fields. The resulting rows can then be matched with the RECEIVING-REPORTS table using the PO-NO foreign key in the RECEIVING REPORTS table to ensure that the items on those purchase orders have in fact been received.

The ITEMS-REQUISITIONED, ITEMS-ORDERED, ITEMS-RECEIVED, and PURCHASED-ITEMS-RETURNED tables all represent M:M relationships. Attributes unique to each of these composite key tables, such as the quantity needed field in the ITEMS-REQUISITIONED table, are listed as nonkey attributes in each table. You might observe that the PAYMENTS table does not include VENDOR-NO as a foreign key. How can the vendor for that payment be identified? The answer is to follow the trail of foreign keys linking the PAYMENTS, RECEIVING-REPORTS, PURCHASE-ORDERS, and VENDORS tables. A whole host of queries can similarly be answered by following the trail of foreign keys linking related tables.

## VI. ENTERPRISE-WIDE ACCOUNTING SYSTEMS

In order to achieve an integrated enterprise-wide accounting information system, all enterprise-wide business processes should be modeled together in an *integrated* manner. Models for sales and purchases business processes were shown above. Now let us examine how these closely related business processes are modeled *together,* especially in a retailing organization. As shown in the ER diagram below, a comprehensive model showing all purchases and sales-related business processes can be somewhat cumbersome. The diagram shown in Fig. 8 covers sales and purchase related processes, as well as business processes related to employees, other expenses, fixed assets, and loans. An enterprise-wide model for a manufacturing organization would be much more complex since it would have to encompass all processes related to procurement of raw materials, manufacturing, labor, work in progress, finished goods inventory, and all sales related processes.

In the cross-functional ER model shown in Fig. 8, note in particular the *integration points* between business processes. For example, *inventory* and *cash* are the two key resources that are affected by both purchasing and selling activities. Inventory increases as a result of purchases and decreases as a result of sales; cash increases as a result of sales and decreases as a result of purchases. Another key aspect of the cross-functional ER model to note in Fig. 8 is that every resource both *increases* and *decreases* in value. Resources are at the heart of every business and the value of an organization's resources would fluctuate over time. In the context of the enterprise ER model above, the resource inventory increases as a result of purchases and decreases as a result of sales. By contrast, the resource cash *decreases* as a result of purchases and *increases* as a result of sales. Thus, each *increase* in a resource because of an event will eventually result in a corresponding *decrease* in another resource through another event. For example, the purchasing set of events results in an *increase* in the inventory resource and a corresponding *decrease* in the cash resource. Similarly, the sales set of events results in a *decrease* in the inventory resource and a corresponding *increase* in cash. Note also that each resource *increase* or *decrease* occurs in response to an event and there is one internal and (usually) one external agent involved in that event. In addition to revenue (sales) and procurement (purchasing) processes, the ER model also shows (1) human resources related processes—hiring, compensating (i.e., paying), and terminating employees; (2) processes for recording and paying other expenses such as utilities, maintenance expenses, phones, etc.; (3) fixed asset processes—acquiring and retiring/selling fixed assets; and (4) loan related processes—obtaining and paying off loans. The addition of these processes results in a comprehensive *enterprise-wide* model for meeting both accounting and other information needs.

### A. Controls in Enterprise Accounting Systems

A critical issue regarding enterprise accounting systems is the accuracy and integrity of the data stored within the system. Users of the enterprise system must have assurance that the information they receive from the system is error free. Recall that in an enterprise system data are stored in a single reposi-

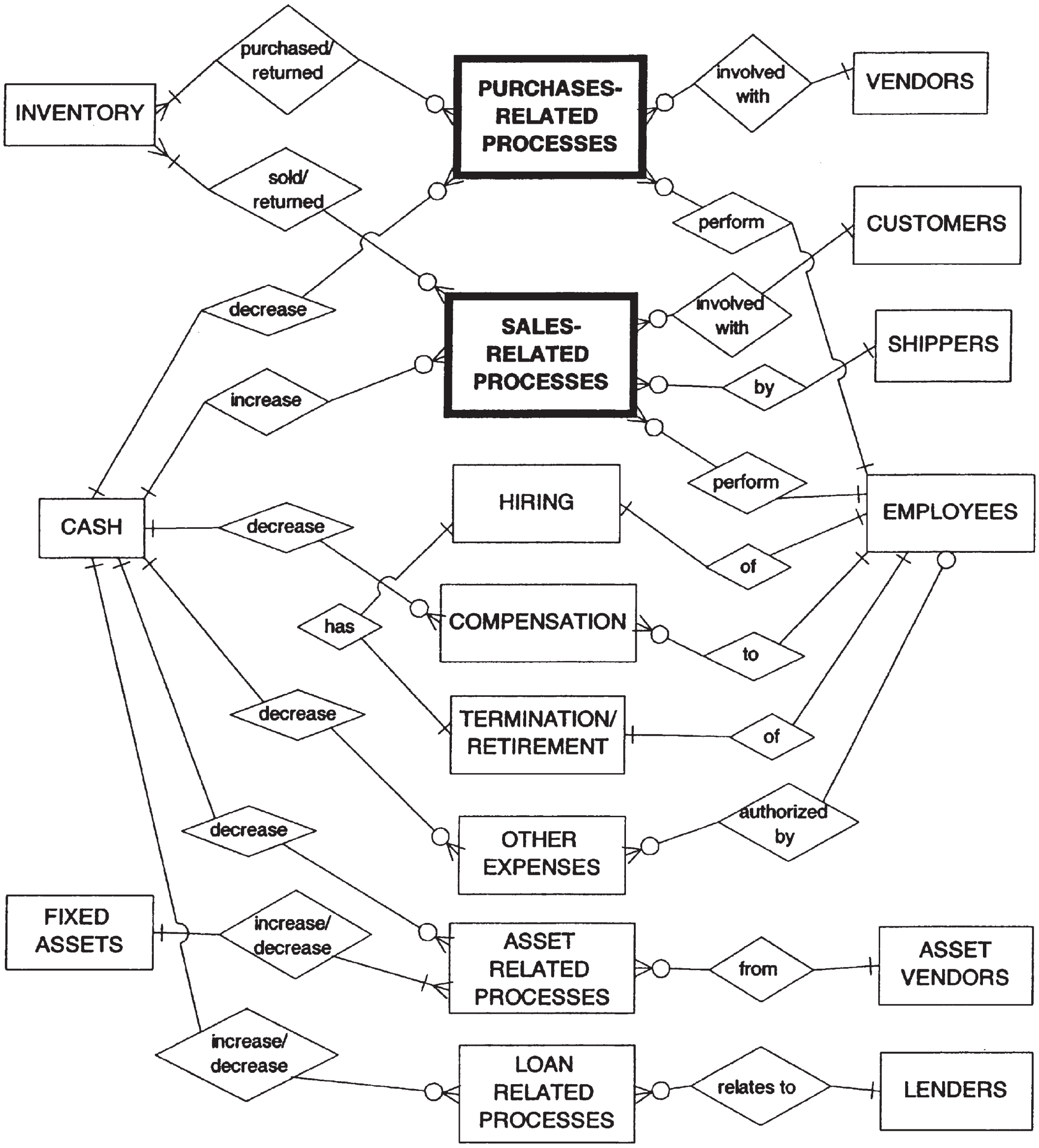


**Figure 8** ER diagram for other business processes (beyond purchases and sales).

tory or database. Control and security concerns are heightened in database environments because of the single shield protecting the entire database, i.e., the database management system. However, database technology also provides opportunities to build controls into the system itself such that errors and irregularities are prevented from ever occurring. *Control procedures* are mechanisms designed to prevent, detect, or correct errors and irregularities. The hardware inside computer systems will usually process transactions and perform calculations in a flawless manner. However, the software that directs the functioning of computer hardware is designed and created by humans. It is the software component of computer-based information systems, and the human component that interacts with computer-based systems, that can cause errors and irregularities in data and thus bring the need for good controls. Control procedures may be applied at various organizational and data processing levels. Control procedures that affect all information systems and subsystems within the organization are categorized as *general control procedures,* while controls designed to prevent or detect errors within each information system or sub-system are categorized as *application control procedures.*

General control procedures are the methods and measures adopted within a business to promote operational efficiency and encourage adherence to prescribed managerial policies. Segregation of duties, maintenance of proper documents and records, and making sure that all accounting transactions are appropriately authorized are some of the common general control procedures that should exist in *all* accounting systems, regardless of the presence or extent of computerization. For computer-based accounting systems, general controls are those controls that facilitate the effective operation and management of the organization's computer systems and all its applications. The six categories of general control procedures are (1) proper organization of the information systems department to ensure adequate supervision over employees and segregation of duties, (2) system development and program change controls including procedures to authorize, test, and document new systems and changes to existing systems, (3) hardware controls including scheduled maintenance of computing and related equipment, (4) access controls to ensure an appropriate match between the duties of employees and the type of access they have to specific files and programs, (5) computer operations controls that cover the functioning of the organization's computing center, and (6) backup and recovery procedures, including a disaster recovery plan, to protect against accidental or intentional loss of data.

Application control procedures are focused on ensuring the accuracy and reliability of data within each subsystem of the enterprise-wide information system. Computer-based application control procedures include *input controls, processing controls,* and *output controls.* As their names suggest, these three sets of control procedures are applicable during the input, processing, and output stages of the data processing cycle. Input control procedures are essentially procedures to validate the data. In an enterprise system employing a relational database, a number of data validation rules can be defined at the table level within the database. In addition, the field type designated for each field in a table can itself serve as a control mechanism. For example, fields defined as Date/Time will accept only date and time data appropriately formatted. In addition, validation rules, which are enforced automatically, can be designed to check whether data entered into a field in a table (1) fall within a certain range, (2) are of the correct length, and (3) match one of the acceptable values for the field. In on-line systems, if a field can have only one of several acceptable values, then the user can be presented with a "pick list" of acceptable values from which a selection can be made. Another powerful feature in on-line systems is the ability to program the system to automatically enter data in certain fields. This control procedure, referred to as *system generated data,* can for example, enter the current date and next order number on an order entry form.

## B. Beyond Transaction Processing Systems

For large organizations, several gigabytes of data may be recorded in the enterprise database within a week or even a day. Moving beyond simply recording transactions, organizations are seeking to obtain business intelligence from their large data repositories. In essence, the goal is to find the "gems" of information in the gigabytes of transaction data. Data warehousing, data marts, and data mining are three concepts aimed at allowing an organization to lever its data to obtain a competitive advantage. A *data warehouse* is a repository of historical business transaction data, organized in a manner to facilitate efficient querying for reaching marketing, tactical, and strategic decisions. The key point is that the data warehouse is separate from the organization's "live" database that captures and stores current transaction data. A *data mart* is a closely related concept. It can be defined as a repository of data gathered from operational data and other sources that is designed to serve certain users' needs. The data in a data mart may actually come from a data warehouse, or it may be more specialized in nature. The emphasis of a data mart is on meeting the specific demands of a particular group of users in terms of analysis, content, presentation, and ease of use. The terms data mart and data warehouse often imply the presence of the other in some form. *Data mining* is the analysis of data for relationships that have not previously been discovered. For example, data mining of sales records for a large grocery store may reveal that consumers that buy diapers also tend to buy ice cream and apple juice. Data mining, also referred to as knowledge discovery, looks for associations (correlated events—beer purchasers also buy peanuts), sequences (one event leading to series of related events—someone buying tile for a new home later buys other home products), classification of data (generating profiles of consumers showing their interests), clustering of data (finding and visualizing groups of related facts), and forecasting (making predictions about future events based on an analysis of patterns of existing events).

## SEE ALSO THE FOLLOWING ARTICLES

Computer-Aided Manufacturing • Control and Auditing • Human Resource Information Systems • Operations Management • Procurement • Productivity • Public Accounting Firms • Supply Chain Management • Transaction Processing Systems

## BIBLIOGRAPHY

Hollander, A., Denna, E., Cherrington, J. O. (1999). *Accounting, information technology, and business solutions,* 2nd edition. Boston, MA: McGraw-Hill.

Murthy, U. S. (2000). *Database systems design and development,* 2nd edition. Bloomington, IN: CyberText Publishing. <http://www.cybertext.com>.

Murthy, U. S., and Groomer, S. M. (2000). *Accounting information systems: A database approach,* 5th edition. Bloomington, IN: CyberText Publishing. <http://www.cybertext.com>.

Perry, J. T., and Schneider, G. P. (2000). *Building accounting systems using Access 2000.* Cincinnati, OH: Thomson Learning/Southwestern.

Potter, D. A. (1993). *Automated accounting systems and procedures handbook.* New York, NY: John Wiley.

Romney, M., and Steinbart, P. (1999). *Accounting information systems,* 8th edition. Reading, MA: Addison-Wesley.

# Advertising and Marketing in Electronic Commerce

**Brenda J. Moscove and Robert G. Fletcher**
*California State University, Bakersfield*

## GLOSSARY

**advertising** Paid communications to the masses by an identifiable sponsor about a product, service, or idea. It should be a persuasive message.

**banner** A miniature billboard that appears when an Internet user calls up a specific web page.

**button** Smaller than a banner providing a link to the advertiser's home page. It can be an icon and generally costs less than a banner because of its smaller size.

**demographics** The statistical characteristics of the group (population, target market, age, income, etc.).

**electronic commerce (e-commerce)** A buying and selling process that is supported by the use of the Internet.

**interstitials** Animated screens that pop up briefly as information is downloaded from a web site. Quite often these screens, sometimes known as "splash pages," are advertisements.

**lifestyle** Describes how customers live. It is helpful in planning advertising and marketing strategies attractive to users of the product/service category.

**marketing concept** Managing a company based upon determining the wants and needs of target markets through delivering satisfaction to these customers at a profit. In other words, the profit is derived from customer satisfaction. In the case of a not-for-profit organization, the same definition applies except that the words "at a profit" are omitted.

**product life cycle** Defines the stages of the product category (introductory, growth, maturity, and decline) and determines the way in which the product/service can be advertised and marketed.

**psychographics** Used to group target market segments according to psychological characteristics (attitudes, values, personality, lifestyle, etc.).

**spam** Unsolicited e-mail contact with present and potential customers. It is like junk mail.

**target market** A group of customers that have common wants and needs that the company or organization chooses to serve.

## I. MARKETING AND ADVERTISING DECISION AREAS

There are several decisions that must be made before designing an information system for e-commerce. Several of the important decisions are discussed in the following sections. This discussion is not meant to be comprehensive but is included to illustrate the types of questions and issues that an organization's management should consider before setting up an information system. The information system is built to satisfy the organization's goals and objectives for the system; thus, every information system for e-commerce advertising and marketing may be unique to the sponsoring organization. The organization's goals and objectives may change and become more complex over time. Conversely, an organization may find that a simple system would be satisfactory. Therefore, it is necessary to revise, simplify, or expand the system to keep it practical and up to date in terms of carrying out management's objectives.

The primary emphasis of this article is the information system needed for e-commerce with focus on advertising and marketing specific goods and services. The issues for building a support system to facilitate advertising and marketing strategies are twofold: a consideration of target market information needed for planning and implementing the strategies; and the ability to take orders, process orders, and deliver the goods and services in a satisfactory manner.

A fundamental consideration is whether or not the organization's marketing and advertising strategies should include e-commerce. In other words, would the organization become more efficient and/or profitable by engaging in e-commerce?

## A. Target Market Selection

The organization needs to determine the audience for the e-commerce message. Is the audience the general public looking for information such as free information dispensed by a local, state, or national government? Another example is free information that is distributed by trade organizations, unions, and cause groups.

Is the desired target a present or potential consumer? Consumers are people who will utilize or consume the product or service for their own personal use. Ski equipment, clothing, and books are examples of consumer products. Airline tickets, concert tickets, and on-line investments are consumer services.

Is the target audience likely to be a business customer? A business customer uses the product and service to carry on or maintain the business activities. Examples are raw materials, component parts, equipment, supplies, and services like bookkeeping or advertising.

Another category for defining the target audience is the institutional or nonprofit segment. The customer may be a governmental entity, a nonprofit hospital, a museum, or a charity, for example. Both business and institutional customers can be combined under the umbrella of "organizational" customers.

A critical issue for defining target markets is the marketing strategy chosen by the organization wishing to engage in e-commerce advertising and marketing. If they choose to market to only one segment, the strategy is called concentration. For instance, if the seller identified three potential segments in the hotel industry such as budget, family, and status, the organization may decide to target only the family segment. Thus, the seller concentrates its strategies on this single target market. All advertising and marketing efforts, including e-commerce, are designed to attract and satisfy this single customer group—the family.

Another strategy the organization could choose is to select the family and budget segments and devise target advertising messages and marketing efforts for each group. This strategy involves multiple segments; i.e., targeting more than one segment, but not all the segments identified, and developing advertising and marketing cues for each target served.

A third strategy is the combination method. The organization may consider the wants and needs of the budget and family market(s) to be fairly similar. Therefore, it combines, or aggregates, these two segments into one target market. It would reject involvement with the status segment. Advertising and marketing e-commerce efforts are designed to appeal to both the family and budget combined target market. These efforts must be cleverly designed to attract and satisfy the combined sector with one set of strategies.

Of course, the final option in overall target audience identification is mass marketing. The organization decides to treat all possible buyers the same and develops only one set of strategies for the entire range of present and potential customers. Mass marketing is the least expensive strategy but can be costly in terms of lost sales because the mass marketing strategy lacks focus.

A complete discussion of target markets and advertising and marketing strategies is beyond the scope of this article. However, additional, detailed information is presented in marketing and advertising textbooks such as: *Principles of Marketing,* 9th Edition, by Philip Kotler and Gary Armstrong, and *Contemporary Advertising,* 7th Edition, by William F. Arens. Further insights into business to business on-line commerce is available in "Let's Get Vertical" in *Business 2.0* magazine.

Marketing and advertising information systems differ according to the type of audience sought. The starting place for determining the type of information system needed for e-commerce is defining the target market(s) the organization wants to serve and designing the system to incorporate the information needed by the target audience(s).

## B. Differences in Reaching the Target Markets with E-Commerce

In conventional advertising and marketing, the marketer and/or advertiser can select the method of reaching the target markets. Thus, the initiative rests with the marketer to choose the media and message that will best attract the attention of the desired target market(s). Chances are that, with the right combination of media and messages, a portion of the target audience(s) will at least be exposed to the product, service, and/or advertising message.

With e-commerce, the initiative is in the hands of the target market (prospective buyer). The prospects elect to visit the home pages for the various companies. This means they can elect *not* to view the home pages as well. The shift in balance for e-commerce, placing the selection power in the hands of the prospective buyer, creates new challenges for the advertisers and marketers. How do they get the prospects to visit their web pages? For specific hints about turning first-time visitors into repeat customers for a web site see "Success Is a Repeat Visitor," in *Fast Company*, June 1998. What information is sought once the web site is visited?

First, the organizations wishing to attract buyers must advertise their web sites in a way that gets the prospective customers to visit the web site. Simply having a site and posting information on it is no guarantee that the prospective customers will find any relevant information on the site. Many of the dotcom ads do not explain what the company does or why the audience should visit the site. They are not memorable. They only add to the clutter of messages that constantly bombard the public and organizational target markets. What would encourage a site visit? What would make the target audience remember the dotcom address, company, and products/services? The dotcom ads should be precise and detail the benefits to the audience for visiting the web site.

Then, the target market actually must visit the web site. Often, the organization has not clearly defined what the target market wants in terms of information; and the web pages fail to attract interest and attention. Furthermore, often the web pages are not user friendly; and the prospect exits the advertiser's site without getting the desired information.

The advertisers and marketers must understand why the prospective customers are visiting the web site and make certain that the proper information and appeals are included in the messages posted on the site in a user-friendly, attractive format. For instance, do prospective customers need to read the company history or do they simply want product/service information? Do they care how many dealerships/retail outlets a seller has? Do they need to know about the size, location, and capacity of the company's production plants? Further, do they need to know how the products are made? Such information is usually considered image rather than product/service information. Image information is aimed at constituencies like shareholders, potential investors, and lending institutions rather than customers and potential buyers.

The real question is how to emphasize information and appeals that will stimulate action for the party visiting the web site instead of concentrating on the items in the above paragraph that have little impact on purchasing decisions. What kind of response action does the organization wish to elicit? Are the materials designed to stimulate the desired response? Part of the system established must contain information directly from prospective customers about what is desired (See Section D, Scope of Electronic Commerce and Marketing Activity).

## C. Audience Demographics and Behavioral Characteristics

Another series of questions involves whether the target market(s) is a specific demographic group of consumers, i.e., sex, age, homeowner, etc., or, a specific group of organizational customers like banks, hospitals, retailers, etc. Furthermore, final consumers can be defined in terms of life cycles, lifestyles, psychographics, and other behavioral characteristics in addition to demographic categories. Examples include: athletic, active/passive, outdoor person, couch potato, etc., lifestyles; type A or type B personalities; and nonuser, light user, medium user, and heavy user consumption patterns.

Businesses can be defined in terms of organizational characteristics as well as types of business. A useful way to identify possible target markets in the United States is to consult the Standard Industrial Classification (SIC) codes issued by the federal government. These numerical codes categorize organizations into specific industries and give information about each organization like the industry type and number of organizations in the industry. It also identifies individual firms, sales volume, number of employees, location, and other information similar to demographic information for final consumers. A North American Industry Classification System (NAICS) is being developed for North American Free Trade countries. Other countries, like Singapore, have similar classification systems. For example, this type of data may be useful in determining the size of the potential business or organizational customer. Size must be defined. Is size the number of employees, the total annual revenue, the yearly gross or net profit, the annual earnings per share, or any other factor that is meaningful in identifying which potential customer group(s) should be targeted? For instance, if a company is selling cardboard boxes, the potential number of cardboard boxes used per year by specific businesses may be helpful in determining its target group. The classification codes' detailed lists of organizations for certain industries, if helpful, could be built into the information system for advertising and marketing using e-commerce techniques.

In addition, target organizations may be defined in terms of organizational behavioral characteristics like always looking for new technology, the lowest price, or status products and services. Usage patterns also are important. These behavioral characteristics of organizational buyers and potential purchasers can be determined best by primary marketing research information that must be gathered directly from the desired target markets. The information cannot be found in secondary reports like government census and other statistical data banks. Behavioral characteristics are vital to the information systems for planning good marketing and advertising strategies for reaching reseller and not-for-profit markets.

Another issue for consideration is the geographical boundary for the target market(s) regardless of whether it is a final consumer, business, or institutional customer. Is the market area a city, a region, a country, a group of countries, or worldwide market? Should a Geographic Information System or Global Positioning System be integrated with the advertising and marketing information system for planning effective advertising and marketing strategies? In determining the audience, these geographical factors must be considered in order to define the target market(s) and location(s) before the information system can be defined and constructed. See Section II.B for additional information.

Any information useful in identifying potential target segments and their wants and needs should be built into the support system for e-commerce. The information must be timely and accurate. The success of using e-commerce for advertising and marketing is largely dependent on how well the e-commerce advertising and marketing strategies satisfy the target audience(s).

## D. Scope and Purpose of E-Commerce Marketing and Advertising Activities

It is important to clearly state the purpose for incorporating e-commerce into marketing and advertising strategy. There are many examples of alternative purposes. Is the purpose to dispense information to the public without seeking a sale like facts about health, community information, or a listing of charities? Is the information geared for building the image (goodwill) of the sponsor rather than immediate sales? Or, is the purpose to attract investors for the sponsoring organization? Is the purpose to generate an inquiry for the sponsoring firm so that a sales person can close the sale? Is the message designed to generate a direct sale without human contact? Also, e-commerce can be used to influence future sales like providing information that may lead to a sale direct from the Internet or through an inquiry during the next year. For example, an automobile manufacturer may sponsor messages about various vehicles knowing that it may take the buyer nine months to decide to actually purchase a new vehicle. The manufacturer wants to ensure that its vehicle models will be considered when the actual purchase occurs.

The e-commerce efforts can include support activities designed to influence purchases by enhancing the competitive position. Offering an e-mail service line to solve the most common problems with the product or service for the buyer may give the company a competitive edge in the market place. This strategy can also be a cost-saving measure on the part of the sponsor. Information about recurring problems and complaints must be built into the information system so that the service line can be responsive. In addition, the service line must be advertised to attract users.

## E. Purpose of the Web Site

There are many purposes for e-commerce efforts. Too often, organizations design web sites without considering what the purpose is for the web site. The purpose helps define the type of information needed to support the web site. The target audience(s) and purpose(s) should determine the type of information, amount of information, and mechanics for accessing the information from the support system.

For instance, the prospective buyer is looking for benefits. When he/she finds the product or service that provides the desired benefits, a transaction will take place. If the seller's message includes a complete company history, product features, and attributes, and other irrelevant information instead of benefits, the prospective buyer will quickly lose interest and visit another site. Also, the web site must be designed to provide easy access to the desired information through a logical series of linkages. The user must be able to understand the linkages and quickly access only the information desired.

A major issue, often overlooked, is whether the web site is designed to replace the organization's current marketing and advertising strategies or supplement present efforts. A company can use e-commerce as its only means of generating sales or as a means to enhance its present traditional advertising and marketing endeavors. Different marketing and advertising strategies apply for brick-and-mortar companies versus those organizations that do business only over the Internet.

For example, a retail chain can use e-commerce to add to its existing customer base supplementing its in-store sales. E-commerce could be used to generate an inquiry for the retailer. Then, the retailer or dealer could close the sale, or, sales may be completed on-line. The on-line sales supplement the organization's traditional in-store sales. Part of the information system consists of getting existing customers to visit the web site to view additional merchandise: clearance items, items that are no longer stocked by the retailers (bulky items like furniture, special orders, etc.). Establishing an interface between in-store customers and Web contacts is essential.

An organization can replace its brick-and-mortar outlets by choosing to do business only by e-commerce. In this case, the existing dealerships, retail establishments, etc., are closed. For example, retail chain Z closes all stores converting all marketing operations to on-line activities. A catalog business has the same options as a brick-and-mortar operation: to continue to do business by catalog and supplement its orders through e-commerce or to convert entirely to e-commerce and dispense with the catalog.

Clearly, the place strategy shifts from emphasis on the retail location to emphasis on the on-line location. However, the distribution strategy becomes more important in terms of the ability to fulfill customer orders on time and in a satisfactory manner. Promotion strategies emphasize attracting the target audience to the web site and obtaining on-line orders instead of the store or dealership visits where merchandise can be seen, touched, tasted, smelled, etc., before purchasing and, often, serviced after the sale. The information base of present and desired customers, their wants and needs, a means of tracking orders, deliveries, and customer satisfaction must be established.

Finally, a new business can decide to maintain both e-commerce and brick-and-mortar sites (supplemental) or to rely exclusively on e-commerce techniques for advertising and marketing its products and services.

## F. Interface with Other Marketing and Functional Activities (Production, Product/Service Offerings, Distribution, Pricing, and Other Promotion Activities)

The information system for advertising and marketing using e-commerce techniques must consider the impact of e-commerce activities on the other functional areas of the company. A vital consideration for the information system is the type of goods and services to market on-line. For example, does the company wish to market their full line of products and services or only those that cannot be obtained elsewhere like clearance merchandise, custom-designed products/services, help-line services to back up dealer products/services, business-to-business networking services? Business networking services include those linkages with business or institutional customers requiring regular repurchasing contacts and establishing procedures that can be on-line functions. The mix of products and services most often featured on-line includes computer, travel, entertainment, books, gifts and flowers, apparel, and food and beverage goods and/or services. If the desired outcome of the web site is an on-line order, the information system must be designed to accommodate all the needed information for taking, processing, and delivering the order in a satisfactory manner.

In terms of order fulfillment, can the present structure used for brick-and-mortar operations accommodate sales stimulated by e-commerce? Many organizations use Just-In-Time (JIT) inventory procedures. Would an expansion in sales through e-commerce interfere with the operation of these procedures? For example, would a retail organization be able to obtain and deliver the large quantities of merchandise that are ordered just before the Christmas holiday? Would producers utilizing JIT inventory procedures be able to produce the items? It is possible that JIT may have to be supplemented with inventory buildup and storage procedures needed to provide satisfactory order fulfillment.

Alliances with Federal Express and United Parcel Service solved the distribution problems experienced by many companies sending individual packages to on-line purchasers. Toys R Us allows on-line customers to avoid shipping charges by using its existing stores as pick up and/or return sites. Wal-Mart hired Fingerhut, a company with a long history in logistics and distribution, to assist in handling on-line orders and deliveries. Wal-Mart associated with Books-A-Million to supply and deliver books ordered on-line. The alliance also increases Wal-Mart's ability to compete with amazon.com. These alliances require changes in the distribution and information systems appropriate to support the seller's activities and service levels. For a more detailed discussion of the implications of e-commerce on the spatial aspects of organizations, see the working paper by Fletcher and Moscove (2000).

The costs of e-commerce must be factored into the costs of doing business. In some instances, replacing brick-and-mortar outlets with e-commerce results in savings to the sellers. Cisco, an Information technology manufacturer, estimates a savings approximating $300 million per year in staff, software distribution, and paperwork costs due to e-commerce transactions. Conversely, the costs of doing business may actually increase. For instance, reorganizing distributions systems to include warehouse facilities needed to service the e-commerce customers satisfactorily may increase the total costs of transactions. Such additional costs must be taken into account when establishing prices for e-commerce offerings. Pricing philosophies of the seller's organization must be factored into the information system as well as changes in the cost/profit relationships produced by the e-commerce activities.

### G. Customer Satisfaction and Relationships

Good marketing and advertising focuses on customer and potential customer satisfaction. This statement is true regardless of whether the customer is a final consumer, a business customer, or an institutional buyer. Satisfied customers demand more products and services and create more sales transactions. More sales transactions usually result in increased profits. Good advertising encourages people to try the product/service and to keep repurchasing it. It helps the purchaser determine that he/she has made the right choice, defend the purchase, and persuade others to buy the same products/services.

The information system must contain helpful data and information from the customers' perspectives, not the sellers' perspectives, about what benefits the buyer pool expects from the products and services being marketed. The system also must contain information about how the customers evaluate their purchases after use. Thus, market research must become part of the information base including customers' reactions to e-commerce efforts. The information base includes primary research to determine the customers' perceptions—the customers' viewpoints and opinions about the marketing communications, product/services, price, and place.

E-commerce marketing and advertising efforts are geared toward building relationships with current customers. It is easier and less expensive to sell more to an existing customer than to obtain a new customer. Many business relationships, therefore, do not end with the transaction itself but are built on establishing strong relationships with the customers that extend to after sale services and feedback. It is essential that the information system includes methods to track after sale customer satisfaction and relationships. The system also should incorporate schedules for recontacting customers about the company, activities, products and service offerings, rewards to present customers, etc. In this way, advertising can serve as a reminder to the customers about the company and its offerings. An organization considering marketing and advertising through e-commerce may find the Cisco Internet Quotient Test helpful in assessing its ability of managing customer relations for e-commerce buyers.

Organizations advertising and marketing through e-commerce offer various services to customers in order to build relationships. A greeting card or gift seller could provide reminders of birthdays, anniversaries, and other special days. Reminders of maintenance needed could be provided to a purchaser of computer equipment and/or vehicles. New product information can be regularly e-mailed to present customers or posted on the web site. These are just a few of the ways in which e-commerce can be used to establish relationships. The information supporting these activities linking the product/service order to the customers must be considered concurrent to the advertising and promotion strategies and integrated into the support system.

## II. MECHANICS OF ADVERTISING AND MARKETING FOR E-COMMERCE

### A. Advertising and Buying Time and Space for E-Commerce

Ways to advertise and market using e-commerce are limited only by existing technology. Today's technology offers web sites, banners, buttons, sponsorships, and interstitials. The web site is a storefront where customers and prospects, shareholders, and other stakeholders can discover more about the company, what the company does and believes, and its product and service offerings. Thus, the company may regard the entire web site as an ad. Because this article assumes the perspective of customers and potential buyers of the firm's products and services, product/service linkages and information inducing a sale are the primary focus.

From a usage standpoint, clicking a banner sends the user to the advertiser's location on the Web. The costs of banners vary depending on the number and types of visitors the site attracts. A typical charge for placing a simple banner ad is $10–$40 per thousand viewers. The banner approach can be supplemented by keyword purchases offered by most search engines. If a user requests information under the keyword, the seller's ad will appear along with ads from any other seller paying for the same keyword.

Buttons are less expensive than banners because they are much smaller. They are about the size of an icon. Software packages such as Java, Shockwave, Acrobat, and Enliven add motion and animation to the buttons and banners. Search engines, like Excite and Webcrawler, provide audio capabilities. Interactive banners and buttons are available. Remember that the banners and buttons should provide easy to use links with the product/service information for the customer pool.

Sponsorships of web pages are available to companies. For instance, IBM sponsors the Superbowl web page. Because of the high costs of sponsorships, the sponsors' banners, buttons, and other information are integrated with the web page messages and visuals.

Interstitials (sometimes called intermercials) are emerging as an effective e-commerce technique. These visuals and messages occur while the user downloads a site. Because they are more intrusive than buttons and banners, it is likely that usage will increase.

Spam, unsolicited e-mail to present and potential customers, is another form of advertising that can have less than positive effects. A number of e-mail providers offer services to eliminate spam (junk) messages. Essentially, these services block any further contact with the potential customers thus eliminating these viewers from the contact pool. If an organization wishes to engage in spam as a way of advertising, access lists and networks can be obtained and incorporated into the information system.

The above techniques are offered as examples of ways to guide buyers and potential target market customers to the marketing information provided online. The techniques do not represent an all-inclusive use of advertising possibilities, search engine selection, and the e-commerce system which are beyond the scope of this article. Once the user accesses the site, the appropriate products/services and proper information and appeals must be conveyed to evoke a purchase response. Also, customer relationships should be solidified by the materials presented on-line.

The costs of using the various techniques to access the marketing communications and the effectiveness of the techniques should become vital parts of the e-commerce information system for marketing and advertising. Advertising on the Web emerged in the last half of the 1990's. Since on-line advertising is relatively new, there still are many questions about the costs associated with e-commerce.

A usual billing technique is cost per thousand based upon the number of page requests for a banner ad. If specific target markets are sought by purchasing space in the search engine's categories and subcategories (finance, games, etc.), various charges result depending on the category and tier. Click-throughs are less popular ways of billing. A user actually clicking on the banner to visit the advertiser's home page is a click-through. Ad networks allow pooling web pages for ad placement but costs are difficult to calculate and verify by the advertiser. Also, various discounts may be negotiated.

In addition to the costs of advertising on-line, the costs of designing and producing the home page and other pages, maintaining the pages, insuring currency and accuracy of information, and establishing the proper linkages should be included in the information system for e-commerce. For example, a professional Web page designer may charge $300–1500 depending on the extent of information and complexity of the page(s). Gathering and accessing information about on-line service suppliers, costs, and media effectiveness is another consideration. These costs include, but are not limited to, home page design and updates, directory service, networking, and newsletters mailed or emailed to online customers. Typical costs can be quite low for less complex online advertisers: $475 for one year including Internet hosting fees, web page design, registration of the web site with various web indexes, and monthly updates. Larger and more complex site charges could be $995 for 6 pages to about $2395 for 20 pages, with service provider fees ranging between $35–55 per month, domain registration of $100, plus setup fees.

## B. Service Providers, Software, Data Bases, and Tracking

A complete listing of various types of service providers and databases is extensive. Therefore, the illustrations given here are intended to stimulate further exploration of the topic on the part of the reader. Search engines such as Yahoo, AOL.com, Excite Network, Lycos, HotBot, and Netscape are e-commerce facilitators.

These companies (publishers) also provide demographic and other information allowing the marketing organization to track visitors to various web-sites (www.relevantknowledge.com). An organization must advertise its web site to web search engines that index the Web. Submitit! (http://www.submit-it.com) and All4oneSubmission Machine (http://www.aall4one.com/all4submit/) are examples of ways to advertise the organization's services to the search engine providers. The web site is registered with the most important indexes.

New software developments can also assist firms in their marketing and advertising efforts. For example, OnDisplay provides infrastructure software to build supply and revenue channels; engage and retain customers; and open on-line trading network partners, distributors, and suppliers. An e-commerce development platform that allows on-line retailers to promote products to customers is Art Technology Group Inc. (ATG). A key feature of this platform is the ability to collect information on individual customers and use this information to create marketing rules based upon their shopping patterns.

Establishing links to important industry sites (free or paid for) are essential for the success of e-commerce advertising and marketing strategies. Obtaining listings of trade association members can be a starting point for establishing these linkages. Obtain targeted mailing lists and news groups by using services such as Dejanews (http://www.deganews.com) to find appropriate sources. Consider joining a shopping mall. For a detailed listing of Banner Exchange Programs see http://bannertips.com/exchangenetworks.shmtl. Media brokers provide appropriate and cost-effective locations for placing banner ads especially helpful to boosting on-line sales for well-known branded products and services.

General information about the Internet demographics and usage is available from Nua Internet Surveys (www.nua.ie), including research reports and demographic surveys produced by major research companies. Another source information about demographics, and other aspects of the Internet (surveys, statistical sources, articles, and reports) is www.teleport.com. Additional general statistical information about the Internet usage can be accessed at the following web site: www.marketinginc.com. For specific information about visitors to individual web sites, one source of information can be found at http://geocities.yahoo.com/home.

Methods for tracking and evaluating the success of on-line marketing and advertising efforts should be built into the information system. A standardized system for tracking the ads is lacking. Some basic tracking issues that should be considered are: Do potential and present customers see the ads?; how effective are the ads? Simply counting the advertising exposures (hits) from web pages is not a satisfactory way of evaluating the effectiveness of the exposure. A more fundamental question is does the advertising evoke the desired response, like an order, an inquiry, or provide additional customer satisfaction?

Tracking considerations when establishing an information system include:

1. Who will provide tracking information
2. What is the cost of obtaining tracking information
3. What type of information will be used for tracking
   - Number of repeat visitors
   - Number and frequency of repeat visitors
   - Location of site prior to visit (search engine used, etc.)
   - Length of visit
   - Pages visited
   - Items examined by visitors
   - Domain names of visitors
   - Geographic location of visitors
   - Purchases made by visitors
   - Inquiries made by visitors

According to the Internet Advertising Bureau (IAB), ad requests can be measured. An ad request is the opportunity to offer the advertisement to a web site visitor. When the user loads a web page containing ads, the browser pulls the ad from the server. The ads appear as banners, buttons, or interstials. The number of ad requests generated can be converted to standard cost per thousand measures. However, an ad request is no assurance that the user actually reads the materials or pursues further linkages.

The click rate is a more refined measure indicating that the viewer actually clicks the linkage to another web page. The click rate is the number of clicks on an ad divided by the number of ad requests. It is used to evaluate the frequency with which users try to obtain additional information about a service or product. However, the click rate still does not measure whether or not a user ever sees the ad or retains the message in a positive way.

Cookies are pieces of information that record a user's activities. Leading browsers, like Netscape Navigator and Microsoft Internet Explorer, support cookies. The use of cookies enables tracking of user demographics and preferences. Internet tracking, profile, and/or rating services are provided by Internet Profiles Corp. (I-PRO) in partnership with A. C. Nielson, Media Metrix, BPA Interactive, and Relevant Knowledge. Or, organizations can build their own tracking devices into the information system. The tracking devices and costs must be factored into the information support systems.

## III. IDENTIFYING OPPORTUNITIES AND PROBLEMS

There are many advantages to organizations resulting from Internet advertising and marketing. Among the advantages are

- An interactive marketing and advertising medium
- Accessibility from around the world
- Immediate response
- Selective targeting
- Reaching affluent, sophisticated users
- Ability to reach organizational users
- Providing detailed information
- Replacing brick-and-mortar operations
- Constant technological advances

Likewise, there are many disadvantages cited for Internet marketing efforts:

- Unclear definition about the purpose of using on-line advertising and marketing strategies
- Security and privacy risks
- Lack of knowledge and standards for measuring advertising effectiveness
- Costs of targeting specific markets
- Other costs associated with on-line marketing
- Inappropriately placed ads
- Inability to fill orders and deliver goods as promised
- Geographic limitations according to economic development and infrastructure of various countries
- Spamming (the on-line equivalent to junk mail)
- Ever-changing technology

There are many issues connected to e-commerce that influence information systems designed to support marketing and advertising activities. Organizations need to consider the opportunities and problems associated with e-commerce marketing and the extent to which the information systems should accommodate the opportunities and risks involved. For a detailed listing of electronic resources for researching and evaluating e-commerce activities, see Fletcher, Moscove, and Corey. See Corey (1998) for a sampling of web sites, e-mail services, and list servers that influence devising marketing and advertising and e-commerce strategies for urban regions derived from a comparative analysis of information technology policy for various locations.

## IV. NEW DEVELOPMENTS

The technology of e-commerce is dynamic, expanding exponentially. In fact, technology is advancing so rapidly that there are few predictions about the final outcome. A few trends that warrant mention are

1. *Wireless technology including handheld devices.* Wireless devices connecting "to the Internet will increase 728% . . . That's an increase from 7.4 million United States users in 1999 to 61.5 million users in 2003."
2. *Mobile commerce.* Mobile commerce negating the necessity for PCs is very popular in Japan and countries where mobile phone usage and handheld devices often surpass PC usage.
3. *Broadband technologies.* Improvements in bandwidth technology increase the speed with which content can be delivered across a network.
4. *Push technology.* Push technology allows messages to be delivered to web users of e-mail and other services for example. Often, spamming has undesirable results.

The emergence of new technology and techniques for advertising and marketing for e-commerce and the absence of industry standards make it impossible to predict the final structure of information systems supporting such activities. However, the technological changes do present exciting challenges for organizations involved in or initiating e-commerce activities. The issues raised in this article are not intended to be all inclusive; they are intended to stimulate further thought and action by organizations building information systems to support e-commerce advertising and marketing strategies.

## SEE ALSO THE FOLLOWING ARTICLES

Business-to-Business Electronic Commerce • Electronic Commerce • Electronic Commerce, Infrastructure for • Enterprise Computing • Marketing • Sales • Service Industries, Electronic Commerce for

## BIBLIOGRAPHY

Arens, W. F. (1999). *Contemporary advertising,* 7th Edition, pp. 515–523. New York: Irwin/McGraw-Hill.

Beck, R. (July 1999). Competition for cybershoppers on rise, p. E1. *Bakersfield Californian*. Bakersfield, CA.

Cisco. (June 2000). Cisco internet quotient test. www.cisco.com/warp/public/750/indicator/quiz.html.

Corey, K. E. (1998). Information technology and telecommunications policies in southeast Asian development: Cases in vision and leadership. *The naga awakens: Growth and change in southeast Asia,* pp. 145–200. Singapore: Times Academic Press.

Fletcher, R. G., and Moscove, B. J. (February 2000). E-commerce and regional science. A Working Paper. Western Regional Sciences Association Conference. Kauai, Hawaii.

Fletcher, R. G., Moscove, B. J., and Corey, K. E. (2001). Electronic commerce: planning for successful urban and regional development. *International urban settings: Lessons of success,* pp. 431–467. Amsterdam: Elsevier Publishers.

Greenstein, M., and Feinman, T. M. (2000). *Electronic commerce; security, risk management and control*. pp. 384–385. New York: Irwin/McGraw-Hill.

ISP-Planet Staff. (February 2000). Wireless to outstrip wired net access. E-mail: townsnda@Yahoo.com.

Norris, M., West, S., and Gaughan, K. (May 2000). *Ebusiness essentials,* pp. 252–254. New York: John Wiley & Sons.

Quain, J. R., (1988). Success is a repeat visitor. *Fast Company,* No. 15, p. 194.

# Artificial Intelligence Programming

**Günter Neumann**
*German Research Center for Artificial Intelligence*

## GLOSSARY

**clauses** Prolog programs consist of a collection of statements, also called clauses, which are used to represent both data and programs.

**higher order function** A function definition which allows functions as arguments or returns a function as its value.

**lists** Symbol structures are often represented using the list data structure, where an element of a list may be either a symbol or another list. Lists are the central structure in Lisp and are used to represent both data and programs.

**recursion** An algorithmic technique where, in order to accomplish a task, a function calls itself with some part of the task.

**symbolic computation** Artificial intelligence programming involves (mainly) manipulating symbols and not numbers. These symbols might represent objects in the world and relationships between those objects. Complex structures of symbols are needed to capture our knowledge of the world.

**term** The fundamental data structure in Prolog is the term which can be a constant, a variable, or a structure. Structures represent atomic propositions of predicate calculus and consist of a functor name and a parameter list.

**PROGRAMMING LANGUAGES IN ARTIFICIAL INTELLIGENCE (AI)** are the major tools for exploring and building computer programs that can be used to simulate intelligent processes such as learning, reasoning, and understanding symbolic information in context. Although in the early days of computer language design the primary use of computers was for performing calculations with numbers, it was found out quite soon that strings of bits could represent not only numbers but also features of arbitrary objects. Operations on such features or *symbols* could be used to represent rules for creating, relating, or manipulating symbols. This led to the notion of *symbolic* computation as an appropriate means for defining algorithms that processed information of any type and thus could be used for simulating human intelligence. Soon it turned out that programming with symbols required a higher level of abstraction than was possible with those programming languages which were designed especially for number processing, e.g., Fortran.

## I. ARTIFICIAL INTELLIGENCE PROGRAMMING LANGUAGES

In AI, the automation or programming of all aspects of human cognition is considered from its foundations in cognitive science through approaches to symbolic and subsymbolic AI, natural language processing, computer vision, and evolutionary or adaptive systems. It is inherent to this very complex problem domain that in the initial phase of programming a specific AI problem it can only be specified poorly. Only through interactive and incremental refinement does more precise specification become possible. This is also due to the fact that typical AI problems tend to be very domain specific; therefore, heuristic strategies have to be developed empirically through generate-and-test approaches (also known as *rapid proto-typing*). In this way, AI programming notably differs from standard software engineering approaches where programming usually starts from a detailed formal specification. In AI programming, the implementation effort is actually part of the problem specification process.

Due to the "fuzzy" nature of many AI problems, AI programming benefits considerably if the programming language frees the AI programmer from the constraints of too many technical constructions (e.g., low-level construction of new data types, manual allocation of memory). Rather, a declarative programming style is more convenient using built-in, high-level data structures (e.g., lists or trees) and operations (e.g., pattern matching) so that symbolic computation is supported on a much more abstract level than would be possible with standard imperative languages such as Fortran, Pascal, or C. Of course, this sort of abstraction does not come for free, since compilation of AI programs on standard von Neumann computers cannot be done as efficiently as for imperative languages. However, once a certain AI problem is understood (at least partially), it is possible to reformulate it in the form of detailed specifications as the basis for reimplementation using an imperative language.

From the requirements of symbolic computation and AI programming, two new basic programming paradigms emerged as alternatives to the imperative style: the *functional* and the *logical* programming styles. Both are based on mathematical formalisms, namely, *recursive function theory* and *formal logic*. The first practical and still most widely used AI programming language is the functional language Lisp developed by John McCarthy in the late 1950s. Lisp is based on mathematical function theory and the lambda abstraction. A number of important and influential AI applications have been written in Lisp, so we will describe this programming language in some detail in this article. During the early 1970s, a new programming paradigm appeared, namely, logic programming on the basis of predicate calculus. The first and still most important logic programming language is Prolog, developed by Alain Colmerauer, Robert Kowalski, and Phillippe Roussel. Problems in Prolog are stated as facts, axioms, and logical rules for deducing new facts. Prolog is mathematically founded on predicate calculus and the theoretical results obtained in the area of automatic theorem proving in the late 1960s.

## II. FUNCTIONAL PROGRAMMING

A mathematical function is a mapping of one set (called the domain) to another (called the range). A function definition is the description of this mapping, either explicitly by enumeration or implicitly by an expression. The definition of a function is specified by a function name followed by a list of parameters in parenthesis followed by the expression describing the mapping, e.g., $\mathtt{CUBE}(X) \equiv X \cdot X \cdot X$, where X is a real number. Alonso Church introduced the notation of *nameless* functions using the lambda notation. A lambda expression specifies the parameters and the mapping of a function using the lambda ($\lambda$) operator, e.g., $\lambda(X)X \cdot X \cdot X$. It is the function itself, so the notation of applying the example nameless function to a certain argument is, for example, $(\lambda(X)X \cdot X \cdot X)(4)$.

Programming in a functional language consists of building function definitions and using the computer to evaluate expressions, i.e., function application with concrete arguments. The major programming task is then to construct a function for a specific problem by combining previously defined functions according to mathematical principles. The main task of the computer is to evaluate function calls and to print the resulting function values. In this way the computer is used like an ordinary pocket computer, but at a much more flexible and powerful level. A characteristic feature of functional programming is that if an expression possesses a well-defined value, then the order in which the computer performs the evaluation does not affect the result of the evaluation. Thus, the result of the evaluation of an expression is just its value. This means that in a pure functional language no side effects exist. Side effects are connected to variables that model memory locations. Thus, in a pure functional programming language no variables exist in the sense of imperative languages. The major control flow methods are recursion and conditional expressions. This is quite different from imperative languages, in which the basic means for control are sequencing and iteration. Functional programming also supports the specification of higher order functions. A higher order function is a function definition which allows functions as arguments or returns a function as its value.

All these aspects together, but especially the latter, are major sources of the benefits of functional programming style in contrast to imperative programming style, viz. that functional programming provides a high-level degree of modularity. When defining a problem by dividing it into a set of subproblems, a major issue concerns the ways in which one can glue the (sub) solutions together. Therefore, to increase ones ability to modularize a problem conceptually, one must provide new kinds of glue in the programming language—a major strength of functional programming.

# III. FUNCTIONAL PROGRAMMING IN LISP

Lisp is the first functional programming language. It was invented to support symbolic computation using linked lists as the central data structure (Lisp stands for *List processor*). John McCarthy noticed that the control flow methods of mathematical functions—recursion and conditionals—are appropriate theoretical means for performing symbolic computations. Furthermore, the notions of functional abstraction and functional application defined in lambda calculus provide for the necessary high-level abstraction required for specifying AI problems.

Lisp was invented by McCarthy in 1958, and a first version of a Lisp programming environment was available in 1960 consisting of an interpreter, a compiler, and mechanisms for dynamic memory allocation and deallocation (known as *garbage collection*). A year later the first language standard was introduced, named Lisp 1.5. Since then a number of Lisp dialects and programming environments have been developed, e.g., MacLisp, FranzLisp, InterLisp, Common Lisp, and Scheme. Although they differ in some specific details, their syntactic and semantic core is basically the same. It is this core which we wish to introduce in this article. The most widely used Lisp dialects are Common Lisp and Scheme. In this article we have chosen Common Lisp to present the various aspects of Lisp with concrete examples. The examples, however, are easily adaptable to other Lisp dialects.

## A. The Syntax and Semantics of Lisp

### 1. Symbolic Expressions

The syntactic elements of Lisp are called symbolic expressions (also known as *s-expressions*). Both data and functions (i.e., Lisp programs) are represented as s-expressions, which can be either *atoms* or *lists.*

**Atoms** are word-like objects consisting of sequences of characters. Atoms can further be divided into different types depending on the kind of characters which are allowed to form an atom. The main subtypes are

- Numbers: 1 2 3 4 –4 3.14159265358979 –7.5 6.02E+23
- Symbols: Symbol Sym23 another-one t false NIL BLUE
- Strings: ”This is a string” ”977?” ”setq” ”He said: \”I’m here.\” ”

Note that although a specific symbol such as BLUE is used because it has a certain meaning for the programmer, for Lisp it is just a sequence of letters or a symbol.

**Lists** are clause-like objects. A list consists of an open left round bracket ( followed by an arbitrary number of *list elements* separated by blanks and a closing right round bracket ). Each list element can be either an atom or a list. Here are some examples of lists:

```
(This is a list) ((this)((too))) ()
  (((((((())))))))
(a b c d) (john mary tom) (loves
  john ?X)
(* (+ 3 4) 8) (append (a b c) (1 2
  3))
(defun member (elem list)
    (if (eq elem (first list)) T
         (member elem (rest list))))
```

Note that in most examples the list elements are lists themselves. Such lists are also called *nested lists.* There is no restriction regarding the depth of the nesting. The examples also illustrate one of the strengths of Lisp: very complex representations of objects can be written with minimal effort. The only thing to watch for is the right number of left and right round brackets. It is important to note that the meaning associated with a particular list representation or atom is not “entered” into the list representation. This means that all s-expressions (as described above) are syntactically correct Lisp programs, but they are not necessarily semantically correct programs.

### 2. Semantics

The core of every Lisp programming system is the *interpreter* whose task is to compute a value for a given s-expression. This process is also called *evaluation.* The result or value of an s-expression is also an s-expression which is returned after the evaluation is completed. Note that this means that Lisp actually has operational semantics, but with a precise mathematical definition derived from recursive function theory.

#### a. Read-Eval-Print Loop

How can the Lisp interpreter be activated and used for evaluating s-expressions and therefore for running real Lisp programs? The Lisp interpreter is actually also defined as a function usually named `EVAL` and is part of any Lisp programming environment (such a function is called a *built-in* function). It is embedded into a Lisp system by means of the so-called *read-eval-print loop,* where an s-expression entered by the user is first *read* into the Lisp system (`READ` is also a built-in function). Then the Lisp interpreter is called via the call of EVAL to evaluate the s-expression, and the resulting s-expression is returned by printing it to the user's device (not surprisingly calling a built-in function PRINT). When the Lisp system is started on the computer, this read-eval-print loop is automatically started and signaled to the user by means of a specific Lisp prompt sign starting a new line. In this article we will use the question mark (?) as the Lisp prompt. For example,

```
? (+ 3 4)
7
```

means that the Lisp system has been started and the read-eval-print loop is activated. The s-expression (+ 3 4) entered by a Lisp hacker is interpreted by the Lisp interpreter as a call of the addition function and prints the resulting s-expression 7 at the beginning of a new line.

#### b. Evaluation

The Lisp interpreter operates according to the following three rules:

1. Identity: A number, a string, or the symbols T and NIL evaluate to themselves. This means that the value of the number 3 is 3 and the value of "house" is "house." The symbol T returns T, which is interpreted to denote the true value, and NIL returns NIL meaning false.
2. Symbols: The evaluation of a symbol returns the s-expression *associated* to it (how this is done will be shown below). Thus, if we assume that the symbol *NAMES* is associated to the list (JOHN MARY TOM), then evaluation of *NAMES* yields that list. If the symbol COLOR is associated with the symbol GREEN, then GREEN is returned as the value of COLOR. In other words, symbols are interpreted as variables *bound* to some values.

### 3. Lists

Every list is interpreted as a function call. The first element of the list denotes the function which has to be applied to the remaining (potentially empty) elements representing the *arguments* of that function. The fact that a function is specified before its arguments is also known as *prefix notation*. It has the advantage that functions can simply be specified and used with an arbitrary number of arguments. The empty list () has the s-expression NIL as its value. Note that this means that the symbol NIL actually has two meanings: one representing the logical false value and one representing the empty list. Although this might seem a bit odd, in Lisp there is actually no problem in identifying which sense of NIL is used.

In general, the arguments are evaluated before the function is applied to the values of the arguments. The order of evaluation of a sequence of arguments is left to right. An argument may represent an atom or a list, in which case it is also interpreted as a function call and the Lisp interpreter is called for evaluating it. For example, consider the following evaluation of a function in the Lisp system:

```
? (MAX 4 (MIN 9 8) 7 5)
8
```

Here, the arguments are 4, (MIN 9 8), 7, and 5, which are evaluated in that order before the function with the name MAX is applied on the resulting argument values. The first argument 4 is a number so its value is 4. The second argument (MIN 9 8) is itself a function call. Thus, before the third argument can be called, (MIN 9 8) has to be evaluated by the Lisp interpreter. Note that because we have to apply the Lisp interpreter for some argument *during* the evaluation of the whole function call, it is also said that the Lisp interpreter is called *recursively*. The Lisp interpreter applies the same steps, so the first argument 9 is evaluated before the second argument 8. Application of the function MIN then yields 8, assuming that the function is meant to compute the minimum of a set of integers. For the *outermost* function MAX, this means that its second argument evaluates to 8. Next the arguments 7 and 5 are evaluated, which yields the values 7 and 5. Now, the maximum function named MAX can be evaluated, which returns 8. This final value is then the value of the whole function call.

#### a. Quoting

Since the Lisp interpreter always tries to identify a symbol's value or interprets a list as a function call, how can we actually treat symbols and lists as data? For example, if we enter the list (PETER WALKS HOME), then the Lisp interpreter will immediately return an error saying something like ERROR: UNKNOWN FUNCTION PETER (the Lisp interpreter should be clever enough to first check whether a function definition exists for the specified function name before it tries to evaluate each argument). Or, if we simply enter HOUSE, then the Lisp interpreter will terminate with an error such as ERROR: NO VALUE BOUND TO HOUSE. The solution to this problem is quite easy: since every first element of a list is interpreted as a function name, each Lisp system comes with a built-in function QUOTE which expects one s-expression as argument and returns this expression without evaluating it. For example, for the list (QUOTE (PETER WALKS HOME)) QUOTE simply returns the value (PETER WALKS HOME), and for (QUOTE HOUSE) it returns HOUSE. Since the function QUOTE is used very often, it can also be expressed by the special character ’. Therefore, for the examples above we can equivalently specify ’(PETER WALKS HOME) and ’HOUSE.

#### b. Programs as Data

Note that QUOTE also enables us to treat function calls as data by specifying, for example, (QUOTE (MAX 4 (MIN 9 8) 7 5)) or ’(MAX 4 (MIN 9 8) 7 5). We already said that the Lisp interpreter is also a built-in unary function named EVAL. It explicitly forces its argument to be evaluated according to the rules mentioned above. In some sense, it can be seen as the opposite function to QUOTE. Thus, to explicitly require that a list specified as data to the Lisp system be interpreted as a function call, we can specify (EVAL ’(MAX 4 (MIN 9 8) 7 5)), which returns the value 8 as described above. In the same way, specifying (EVAL ’(PETER WALKS HOME)) will cause a Lisp error because Lisp tries to call a function PETER.

The main advantage of being able to treat programs as data is that we can define Lisp programs (functions) which are able to construct or generate programs such that they first build the corresponding list representation and then explicitly call the Lisp interpreter using EVAL in order to evaluate the just created list as a function. It is not surprising that due to this characteristic Lisp is still the dominant programming language in the AI area of genetic programming.

#### c. Assigning Values to Symbols

When programming real-life practical programs, one often needs to store values computed by some program to a variable to avoid costly recomputation of that value if it is needed in another program at some later time. In a purely functional version of Lisp, the value of a function only depends on the function definition and on the value of the arguments in the call. In order to make Lisp a practical language (practical at least in the sense that it can run efficiently on von Neumann computers), we need a way to assign values to symbols.

Common Lisp comes with a built-in function called SETQ. SETQ expects two arguments: the symbol (called the variable) to which a value is bound and an s-expression which has to provide the value. The Lisp interpreter treats the evaluation of SETQ in a special way, such that it explicitly supresses evaluation of SETQ’s first argument (the variable), but rather binds the value of SETQ’s second argument to the variable (to understand how Lisp internally binds a value to a symbol would require too many technical details which we cannot go into in this short article). The value of the second argument of SETQ is returned as the value of SETQ. Here are some examples:

```
? COLOR
ERROR: UNBOUND SYMBOL COLOR
? (SETQ COLOR ‘GREEN)
GREEN
? (SETQ MAX (MAX 3 2.5 1))
3
```

Note that SETQ actually changes the status of the Lisp interpreter because the next time the same variable is used, it has a value and therefore the Lisp interpreter will be able to return it. If this effect did not occur, then the Lisp interpreter would signal an error because that symbol would not be bound (cf. step 2 of the Lisp interpreter). Thus, it is also said that SETQ produces a *side effect* because it dynamically changes the status of the Lisp interpreter. When making use of SETQ one should, however, be aware of the fact that one is leaving the proper path of semantics of pure Lisp. SETQ should therefore be used with great care.

## B. The List Data Type

Programming in Lisp actually means defining functions that operate on lists, e.g., create, traverse, copy, modify, and delete lists. Since this is central to Lisp, every Lisp system comes with a basic set of primitive built-in functions that efficiently support the main list operations. We will briefly introduce the most important ones now.

### 1. Type Predicate

First, we have to know whether a current s-expression is a list or not (i.e., an atom). This job is accomplished by the function LISTP, which expects any s-expression EXPR as an argument and returns the symbol T if EXPR is a list and NIL if it is otherwise. Examples are [we will use the right arrow (⇒) for pointing to the result of a function call] the following:

```
(LISTP ’(1 2 3)) ⇒ T
(LISTP ’()) ⇒ T
(LISTP ’3) ⇒ NIL
```

### 2. Selection of List Elements

Two basic functions exist for accessing the elements of a list: CAR and CDR. Both expect a list as their argument. The function CAR returns the first element in the list or NIL if the empty list is the argument, and the function CDR returns the same list from which the first element has been removed or NIL if the empty list was the argument. For example,

```
(CAR '(A B C)) ⇒ A  (CDR '(A B C))
  ⇒ (B C)
(CAR '()) ⇒ NIL      (CDR '(A)) ⇒
  NIL
(CAR '((A B) C)) ⇒ (A B)  (CDR
  '((A B) C)) ⇒ C
```

By means of a sequence of CAR and CDR function calls, it is possible to traverse a list from left to right and from outer to inner list elements. For example, during evaluation of

```
(CAR (CDR '(SEE THE QUOTE)))
```

the Lisp interpreter will first evaluate the expression

```
(CDR '(SEE THE QUOTE))
```

which returns the list (THE QUOTE), which is then passed to the function CAR which returns the symbol THE. Here are some further examples:

```
(CAR (CDR (CDR '(SEE THE QUOTE))))
  ⇒ QUOTE
(CAR (CDR (CDR (CDR '(SEE THE
  QUOTE))))) ⇒ NIL
(CAR (CAR '(SEE THE QUOTE))) ⇒ ???
```

What will happen during evaluation of the last example? Evaluation of (CAR '(SEE THE QUOTE)) returns the symbol SEE. This is then passed as argument to the outer call of CAR. However, CAR expects a list as argument, so the Lisp interpreter will immediately stop further evaluation with an error such as ERROR: ATTEMPT TO TAKE THE CAR OF SEE WHICH IS NOT LISTP.

A short historical note: the names CAR and CDR are old fashioned because they were chosen in the first version of Lisp on the basis of the machine code operation set of the computer on which it was implemented (CAR stands for "contents of address register" and CDR stands for "contents of decrement register." In order to write more readable Lisp code, Common Lisp comes with two equivalent functions, FIRST and REST. We have used the older names here as it enables reading and understanding of older AI Lisp code.

## 3. Construction of Lists

Analogously to CAR and CDR, a primitive function CONS exists which is used to construct a list. CONS expects two s-expressions and inserts the first one as a new element in front of the second one. Consider the following examples:

```
(CONS 'A '(B C)) ⇒ (A B C)
(CONS '(A D) '(B C)) ⇒ ((A D) B C)
(CONS (FIRST '(1 2 3)) (REST '(1 2
  3))) ⇒ (1 2 3)
```

In principle, CONS together with the empty list suffice to build very complex lists, for example,

```
(CONS 'A (CONS 'B (CONS 'C '())))
  ⇒ (A B C)
(CONS 'A (CONS (CONS 'B (CONS 'C
  '())) (CONS 'D '()))) ⇒ (A (B C)
  D)
```

However, since this is quite cumbersome work, most Lisp systems come with a number of more advanced built-in list functions. For example, the function LIST constructs a list from an arbitrary number of s-expressions, and the function APPEND constructs a new list through concatenation of its arguments which must be lists. EQUAL is a function which returns T if two lists have the same elements in the same order, otherwise it returns NIL. For example,

```
(LIST 'A 'B 'C) ⇒ (A B C) (LIST
  (LIST 1) 2 (LIST 1 2 3)) ⇒ ((1)
  2 (1 2 3))
(APPEND '(1) (LIST 2)) ⇒ (1 2)
  (APPEND '(1 2) NIL '(3 4)) ⇒ (1
  2 3 4)
(EQUAL '(A B C) '(A B C)) ⇒ T
  (EQUAL '(A B C) '(A C B)) ⇒ NIL
```

## C. Defining New Functions

Programming in Lisp is done by defining new functions. In principle this means specifying lists in a certain syntactic way. Analogously to the function SETQ, which is treated in a special way by the Lisp interpreter, there is a special function DEFUN which is used by the Lisp interpreter to create new function objects. DEFUN expects as its arguments a symbol denoting the *function name,* a (possibly empty) list of *parameters* for the new function, and an arbitrary number of s-expressions defining the *body* of the new function. Here is the definition of a simple function named MY-SUM which expects two arguments from which it will construct the sum using the built-in function +:

```
(DEFUN MY-SUM (X Y)
  (+ X Y))
```

This expression can be entered into the Lisp system in the same way as a function call. Evaluation of a function definition returns the function name as value, but will create a function object as side effect and adds it to the set of function definitions known by the Lisp system when it is started (which is at least the set of built-in functions). Note that in this example the body consists only of one s-expression. However, the body might consist of an arbitrary sequence of s-expressions. The value of the last s-expression of the body determines the value of the function. This means that all other elements of the body are actually irrelevant, unless they produce intended side effects.

The parameter list of the new function MY-SUM tells us that MY-SUM expects exactly two s-expression as arguments when it is called. Therefore, if you enter (MY-SUM 3 5) into the Lisp system, the Lisp interpreter will be able to find a definition for the specified function name and then process the given arguments from left to right. When doing so, it binds the value of each argument to the corresponding parameter specified in the parameter list of the function definition. In our example, this means that the value of the first argument 3 (which is also 3 since 3 is a number which evaluates to itself) is bound to the parameter X. Next, the value of the second argument 5 is bound to the parameter Y. Because the value of an argument is bound to a parameter, this mechanism is also called CALL BY VALUE. After having found a value for all parameters, the Lisp interpreter is able to evaluate the body of the function. In our example, this means that (+ 3 5) will be called. The result of the call is 8, which is returned as result of the call (MY-SUM 3 5). After the function call is completed, the *temporary* binding of the parameters X and Y are deleted.

Once a new function definition has been entered into the Lisp system, it can be used as part of the definition of new functions in the same way as built-in functions are used, as shown in the following example:

```
(DEFUN DOUBLE-SUM (X Y)
  (+ (MY-SUM X Y) (MY-SUM X Y)))
```

which will double the sum of its arguments by calling MY-SUM twice.

Here is another example of a function definition, demonstrating the use of multiple s-expressions in the function body:

```
(DEFUN HELLO-WORLD () (PRINT ”HELLO
  WORLD!”) ’DONE)
```

This function definition has no parameter because the parameter list is empty. Thus, when calling (HELLO-WORLD), the Lisp interpreter will immediately evaluate (PRINT ”HELLO WORLD!”) and will print the string “Hello World!” on your display as a side effect. Next, it will evaluate the symbol ’DONE, which returns DONE as result of the function call.

## D. Defining Control Structures

Although it is now possible to define new functions by combining built-in and user-defined functions, programming in Lisp would be very tedious if it were not possible to control the flow of information by means of conditional branches perhaps iterated many times until a stop criterion is fulfilled. Lisp branching is based on function evaluation: control functions perform tests on actual s-expressions and, depending on the results, selectively evaluate alternative s-expressions.

The fundamental function for the specification of conditional assertions in Lisp is COND. COND accepts an arbitrary number of arguments. Each argument represents one possible branch and is represented as a list where the first element is a test and the remaining elements are actions (s-expressions) which are evaluated if the test is fulfilled. The value of the last action is returned as the value of that alternative. All possible arguments of COND (i.e., branches) are evaluated from left to right until the first branch is positively tested. In that case the value of that branch is the value of the whole COND function. This sounds more complicated than it actually is. Let us consider the following function VERBALIZE-PROP, which verbalizes a probability value expressed as a real number:

```
(DEFUN VERBALIZE-PROP (PROB-VALUE)
  (COND ((> PROB-VALUE 0.75) ‘VERY-
      PROBABLE)
         ((> PROB-VALUE 0.5)
           ’PROBABLE)
         ((> PROB-VALUE 0.25)
           ’IMPROBABLE)
         (T ’VERY-IMPROBABLE)))
```

When calling (VERBALIZE-PROP 0.33), the actual value of the argument is bound to the parameter PROB-VALUE. Then COND is evaluated with that binding. The first expression to be evaluated is ((> PROB-VALUE 0.75) ‘VERY-PROBABLE). > is a built-in predicate which tests whether the first argument is greater than the second argument. Since PROB-VALUE is 0.33, > evaluates to NIL, which means that the test is not fulfilled. Therefore, evaluation of this alternative branch is terminated immediately, and the next alternative ((> PROB-VALUE 0.5) ’PROBABLE) is evaluated. Here the test function also returns NIL, so the evaluation is terminated also. Next, ((> PROB-VALUE 0.25) 'IMPROBABLE) is evaluated. Applying the test function now returns T, which means that the test is fulfilled. Then all actions of this positively tested branch are evaluated and the value of the last action is returned as the value of COND. In our example, only the action 'IMPROBABLE has been specified, which returns the value IMPROBABLE. Since this defines the value of COND, and because the COND expression is the only expression of the body of the function VERBALIZE-PROP, the result of the function call (VERBALIZE-PROP 0.33) is IMPROBABLE. Note that if we enter (VERBALIZE-PROP 0.1), the returned value is VERY-IMPROBABLE because the test of the third alternative will also fail and the branch (T 'VERY-IMPROBABLE) has to be evaluated. In this case, the symbol T is used as the test which always returns T, so the value of this alternative is VERY-IMPROBABLE.

## E. Recursive Function Definitions

The second central device for defining control flow in Lisp is *recursive function definitions.* A function which partially uses its definition as part of its own definition is called *recursive.* Thus seen, a recursive definition is one in which a problem is decomposed into smaller units until no further decomposition is possible. Then these smaller units are solved using known function definitions, and the sum of the corresponding solutions form the solution of the complete program. Recursion is a natural control regime for data structures which have no definite size, such as lists, trees, and graphs. Therefore, it is particularly appropriate for problems in which a space of states has to be searched for candidate solutions.

Lisp was the first practical programming language that systematically supported the definition of recursive definitions. We will use two small examples to demonstrate recursion in Lisp. The first example is used to determine the length of an arbitrarily long list. The length of a list corresponds to the number of its elements. Its recursive function is as follows:

```
(DEFUN LENGTH (LIST)
(COND ((NULL LIST) 0)
      (T (+ 1 (LENGTH (CDR
      LIST))))))
```

When defining a recursive definition, we have to identify the base cases, i.e., those units which cannot be decomposed any further. Our problem size is the list. The smallest problem size of a list is the empty list. Thus, the first thing we have to do is to specify is a test for identifying the empty list and to define what the length of the empty list should be. The built-in function NULL tests whether a list is empty, in which case it returns T. Since the empty list is a list with no elements, we define the length of the empty list as 0. The next thing to be done is to decompose the problem size into smaller units so that the same problem can be applied to smaller units. Decomposition of a list can be done by using the functions CAR and CDR, which means that we have to specify what is to be done with the first element of a list and the rest until the empty list is found. Since we already have identified the empty list as the base case, we can assume that decomposition will be performed on a list containing at least one element. Thus, every time we are able to apply CDR to get the rest of a list, we have found one additional element which should be used to increase the number of the already identified list elements by 1. Making use of this function definition, (LENGTH '()) will immediately return 0, and if we call (LENGTH '(A B C)), the result will be 3, because three recursive calls have to be performed until the empty list can be determined.

As a second example, we consider the recursive definition of MEMBER, a function which tests whether a given element occurs in a given list. If the element is indeed found in the list, it returns the sublist which starts with the first occurrence of the found element. If the element cannot be found, NIL is returned. The following are example calls:

```
(MEMBER 'B '(A F B D E B C)) ⇒
  (B D E B C)
(MEMBER 'K '(A F B D E B C)) ⇒
  NIL
```

Similarly to the recursive definition of LENGTH, we use the empty list as the base case. For MEMBER, the empty list means that the element in question is not found in the list. Thus, we have to decompose a list until the element in question is found or the empty list is determined. Decomposition is done using CAR and CDR. CAR is used to extract the first element of a list, which can be used to check whether it is equal to the element in question, in which case we can directly stop further processing. If it is not equal, then we should apply the MEMBER function on the remaining elements until the empty list is determined. Thus, MEMBER can be defined as follows:

```
(DEFUN MEMBER (ELEM LIST)
  (COND ((NULL LIST) NIL)
        ((EQUAL ELEM (CAR LIST))
          LIST)
          (T (MEMBER ELEM (CDR
            LIST)))))
```

## F. Higher Order Functions

In Lisp, functions can be used as arguments. A function that can take functions as its arguments is called a *higher order function*. There are a lot of problems where one has to traverse a list (or a tree or a graph) such that a certain function has to be applied to each list element. For example, a *filter* is a function that applies a test to the list elements, removing those that fail the test. *Maps* are functions which apply the same function on each element of a list, returning a list of the results. High-order function definitions can be used for defining generic list traversal functions such that they abstract away from the specific function used to process the list elements.

In order to support high-order definitions, their is a special function, FUNCALL, which takes as its arguments a function and a series of arguments and applies that function to those arguments. As an example of the use of FUNCALL, we will define a generic function FILTER which may be called in the following way:

```
(FILTER '(1 3 -9 -5 6 -3) #'PLUSP)
  ⇒ (1 3 6)
```

PLUSP is a built-in function which checks whether a given number is positive or not. If so, it returns that number, otherwise NIL is returned. The special symbol # is used to tell the Lisp interpreter that the argument value denotes a function object. The definition of FILTER is as follows:

```
(DEFUN FILTER (LIST TEST)
  (COND ((NULL LIST) LIST)
        ((FUNCALL TEST (CAR LIST))
          (CONS (CAR LIST) (FILTER
              (CDR LIST) TEST)))
        (T (FILTER (CDR LIST)
          TEST))))
```

If the list is empty, then it is simply returned. Otherwise, the test function is applied to the first element of the list. If the test function succeeds, CONS is used to construct a result list using this element and all elements that are determined during the recursive call of FILTER using the CDR of the list and the test function. If the test fails for the first element, this element is simply skipped by recursively applying FILTER on the remaining elements, i.e., this element will not be part of the result list. The filter function can be used for many different test functions, e.g.,

```
(FILTER '(1 3 A B 6 C 4) #'NUMBERP)
  ⇒ (1 3 6 4)
(FILTER '(1 2 3 4 5 6) #'EVEN) ⇒
  (2 4 6)
```

As another example of a higher order function definition, we will define a simple mapping function, which applies a function to all elements of a list and returns a list of all values. If we call the function MY-MAP, then the definition looks like the following:

```
(DEFUN MY-MAP (FN LIST)
  (COND ((NULL LIST) LIST)
        (T (CONS (FUNCALL FN (CAR
             LIST)) (MY-MAP FN (CDR
LIST))))))
```

If a function DOUBLE exists which just doubles a number, then a possible call of MY-MAP could be

```
(MY-MAP #'DOUBLE '(1 2 3 4)) ⇒ (2
  4 6 8)
```

Often it is the case that a function should only be used once. Thus, it would be quite convenient if we could provide the definition of a function directly as an argument of a mapping function. To do this, Lisp supports the definition of LAMBDA expressions. We have already informally introduced the notation of LAMBDA expressions in Section II as a means of defining nameless or *anonymous* functions. In Lisp, LAMBDA expressions are defined using the special form LAMBDA. The general form of a LAMBDA expression is

```
(LAMBDA (parameter . . .)
  body . . .)
```

A LAMBDA expression allows us to separate a function definition from a function name. LAMBDA expressions can be used in place of a function name in a FUNCALL, e.g., the LAMBDA expression for our function DOUBLE may be

```
(LAMBDA (X) (+ X X))
```

For example, the above function call of MY-MAP can be restated using the LAMBDA expression as follows:

```
(MY-MAP #'(LAMBDA (X) (+ X X)) '(1
  2 3 4) ⇒ (2 4 6 8)
```

A LAMBDA expression returns a function object which is not bound to a function name. In the definition of MY-MAP we used the parameter FN as a function name variable. When evaluating the lambda form, the Lisp interpreter will bind the function object to that function name variable. In this way, a function parameter is used as a dynamic function name. The # symbol is necessary to tell Lisp that it should not only bind a function object, but should also maintain the bindings of the local and global values associated to the function object. This would not be possible by simply using the QUOTE operator alone (unfortunately, further details cannot be given here due to the space constraints).

### G. Other Functional Programming Languages Than Lisp

We have introduced Lisp as the main representative functional programming language (especially the widely used dialect Common Lisp) because it is still a widely used programming language for a number of AI problems such as Natural Language Understanding, Information Extraction, Machine Learning, AI planning, or Genetic Programming. Beside Lisp, a number of alternative functional programming languages have been developed. We will briefly mention two well-known members, viz. ML and Haskell.

Meta-Language (ML) is a static-scoped functional programming language. The main differences to Lisp are its syntax (which is more similar to that of Pascal) and a strict polymorphic type system (i.e., using strong types and type inference, which means that variables need not be declared). The type of each declared variable and expression can be determined at compile time. ML supports the definition of abstract data types, as demonstrated by the following example:

```
DATATYPE TREE   =     L OF INT
     |      INT * TREE * TREE;
```

which can be read as "every binary tree is either a leaf containing an integer or it is a node containing an integer and two trees (the subtrees)." An example of a recursive function definition applied on a tree data structure is shown in the following example:

```
FUN DEPTH(L _) = 1
 | DEPTH(N(I,L,R)) =
    1 + MAX(DEPTH L, DEPTH R);
```

The function DEPTH maps trees to integers. The depth of a leaf is 1 and the depth of any other tree is 1 plus the maximum of the depths of the left and right subtrees.

Haskell is similar to ML: it uses a similar syntax, it is also static scoped, and it makes use of the same type inferencing method. It differs from ML in that it is purely functional. This means that it allows no side effects and includes no imperative features of any kind, basically because it has no variables and no assignment statements. Furthermore, it uses a *lazy* evaluation technique, in which no subexpression is evaluated until its value is known to be required.

Lists are a commonly used data structure in Haskell. For example, [1,2,3] is the list of three integers 1, 2, and 3. The list [1,2,3] in Haskell is actually shorthand for the list 1:(2:(3:[])), where [] is the empty list and : is the infix operator that adds its first argument to the front of its second argument (a list). As an example of a user-defined function that operates on lists, consider the problem of counting the number of elements in a list by defining the function LENGTH:

```
LENGTH :: [A] -> INTEGER
LENGTH [] = 0
LENGTH (X:XS) = 1 + LENGTH XS
```

which can be read as "The length of the empty list is 0, and the length of a list whose first element is X and remainder is XS is 1 plus the length of XS." In Haskell, function invocation is guided by *pattern matching*. For example, the left-hand sides of the equations contain patterns such as [] and X:XS. In a function application these patterns are matched against actual parameters ([] only matches the empty list, and X:XS will successfully match any list with at least one element, binding X to the first element and XS to the rest of the list). If the match succeeds, the right-hand side is evaluated and returned as the result of the application. If it fails, the next equation is tried. If all equations fail, an error results.

This ends our short "tour de Lisp." We were only able to discuss the most important aspects of Lisp. Readers interested in more specific details should consult at least one of the books mentioned in the Bibliography. The rest of this article will now be used to introduce another programming paradigm widely used in AI programming, namely, Prolog.

## IV. LOGICAL PROGRAMMING IN PROLOG

In the 1970s an alternative paradigm for symbolic computation and AI programming arose from the success in the area of automatic theorem proving. Notably, the resolution proof procedure developed by Robinson (1965) showed that formal logic, particularly predicate calculus, could be used as a notation for defining algorithms and, therefore, for performing symbolic computations. In the early 1970s, Prolog (an acronym for *Programming in Logic*), the first logical-based programming language, appeared. It was developed by Alain Colmerauer, Robert Kowalski, and Phillippe Roussel. Basically, Prolog consists of a method for specifying predicate calculus propositions and a restricted form of resolution. Programming in Prolog consists of the specification of *facts* about objects and their relationships and *rules* specifying their logical relationships. Prolog programs are *declarative* collections of statements about a problem because they do not specify how a result is to be computed, but rather define *what* the logical structure of a result should be. This is quite different from imperative and even functional programming, where the focus is on defining *how* a result is to be computed. Using Prolog, programming can be done at a very abstract level quite close to the formal specification of a problem. Prolog is still the most important logical programming language. There are a number of commercial programming systems on the market which include modern programming modules, i.e., compiler, debugger, and visualization tools. Prolog has been used successfully in a number of AI areas such as expert systems and natural language processing, but also in such areas as relational database management systems or education.

## A. A Simple Prolog Program

Here is a very simple Prolog program consisting of two facts and one rule:

```
scientist(gödel).
scientist(einstein).
logician(X) :- scientist(X).
```

The first two statements can be paraphrased as "Gödel is a scientist" and "Einstein is a scientist." The rule statement says "X is a logician if X is a scientist." In order to test this program, we have to specify query expressions (or theorems) which Prolog tries to answer (or to prove) using the specified program. One possible query is

```
?- scientist(gödel).
```

which can be verbalized as "Is Gödel a scientist?" Prolog, by applying its built-in proof procedure, will respond with "yes" because a fact may be found which exactly matches the query. Another possible query verbalizing the question "Who is a scientist?" and expressed in Prolog as

```
?- scientist(X).
```

will yield the Prolog answer "X = gödel, X = einstein." In this case Prolog not only answers yes, but returns all bindings of the variable X which it finds during the successful proof of the query. As a further example, we might also query "Who is a logician?" using the following Prolog query:

```
?- logician(X).
```

Proving this query will yield the same set of facts because of the specified rule. Finally, we might also specify the following query:

```
?- logician(mickey-mouse).
```

In this case Prolog will respond with "no." Although the rule says that someone is a logician if he or she is also a scientist, Prolog does not find a fact saying that Mickey Mouse is a scientist. Note, however, that Prolog can only answer relative to the given program, which actually means "no, I couldn't deduce the fact." This property is also known as the *closed world assumption* or *negation as failure.* It means that Prolog assumes that all knowledge that is necessary to solve a problem is present in its database.

## B. Prolog Statements

Prolog programs consist of a collection of statements, also called *clauses,* which are used to represent both data and programs. The dot symbol is used to terminate a clause. Clauses are constructed from *terms.* A term can be a *constant* (symbolic names that have to begin with a lowercase letter, such as `gödel` or `eIn-Stein`), a *variable* (symbols that begin with an uppercase letter, such as `X` or `Scientist`), or a *structure.* Structures represent atomic propositions of predicate calculus and consist of a functor name and a parameter list. Each parameter can be a term, which means that terms are recursive objects. Prolog distinguishes three types of clauses: facts, rules, and queries. A *fact* is represented by a single structure, which is logically interpreted as a simple true proposition. In the simple example program above we already introduced two simple facts. Here are some more examples:

```
male(john).
male(bill).
female(mary).
female(sue).
father(john, mary).
father(bill,john).
mother(sue,mary).
```

Note that these facts have no intrinsic semantics, i.e., the meaning of the functor name `father` is not defined. For example, applying  common sense, we may interpret it as "John is the father of Mary." However, for Prolog, this meaning does not exist, it is just a symbol.

*Rules* belong to the next type of clauses. A rule clause consists of two parts: the *head* which is a single term and the *body* which is either a single term or a conjunction. A conjunction is a set of terms separated by the comma symbol. Logically, a rule clause is interpreted as an implication such that if the elements of the body are all true, then the head element is also true. Therefore, the body of a clause is also denoted as the *if* part and the head as the *then* part of a rule. Here is an example for a set of rule clauses:

```
parent(X,Y) :- mother(X, Y).
parent(X,Y) :- father(X, Y).
grandparent(X,Z) :- parent(X,Y),
  parent(Y,Z).
```

where the last rule can be read as "X is a grandparent of Z, if X is a parent of Y *and* Y is a parent of Z." The first two rules say "someone is a parent if it is the father or mother of someone else." The reason we treat the first two rules as a disjunction will become clear when we introduce Prolog's proof procedure. Before doing this, we shall introduce the last type of clause, the *query* clause (also called the *goal* clause). A query is used to activate Prolog's proof procedure. Logically, a query corresponds to an unknown theorem. It has the same form as a fact. In order to tell Prolog that a query has to be proven, the special query operator `?-` is usually written in front of the query. In the simple Prolog program introduced above, we have already seen an informal description of how a query is used by Prolog.

## C. Prolog's Inference Process

Prolog's inference process consists of two basic components: a search strategy and a unifier. The search strategy is used to search through the fact and rule database, while unification is used for pattern matching and returns the bindings that make an expression true.

The unifier is applied on two terms and tries to combine them both to form a new term. If unification is not possible, then unification is said to have *failed.* If the two terms contain no variables, then unification actually reduces to checking whether the terms are equal. For example, unification of the two terms

```
father(john,mary) and
  father(john,mary)
```

succeeds, whereas unification of the following term pairs will fail:

```
father(X,mary) and father(john,sue)
sequence(a,b,c) and sequence(a,b)
```

If a term contains a variable (or more), then the unifier checks whether the variable can be bound with some information from the second term, however, only if the remaining parts of the terms unify. For example, for the following two terms:

```
father(X,mary) and father(john,mary)
```

the unifier will bind `X` to `john` because the remaining terms are equal. However, for the following pair:

```
father(X,mary) and father(john,sue)
```

the binding would not make sense, since `mary` and `sue` do not match.

The search strategy is used to traverse the search space spanned by the facts and rules of a Prolog program. Prolog uses a *top-down, depth-first* search strategy. What does this mean? The whole process is quite similar to the function evaluation strategy used in Lisp. If a query Q is specified, then it may either match a fact or a rule. In case of a rule R, Prolog first tries to match the head of R, and if it succeeds, it then tries to match all elements from the body of R which are also called *subqueries.* If the head of R contains variables, then the bindings will be used during the proof of the subqueries. Since the bindings are only valid for the subqueries, it is also said that they are *local* to a rule. A subquery can either be a fact or a rule. If it is a rule, then Prolog's inference process is applied recursively to the body of such subquery. This makes up the top-down part of the search strategy. The elements of a rule body are applied from left to right, and only if the current element can be proven successfully is the next element tried. This makes up the depth-first strategy. It is possible that for the proof of a subquery two or more alternative facts or rules are defined. In that case Prolog selects one alternative A and tries to prove it, if necessary by processing subqueries of A. If A fails, Prolog goes back to the point where it started the proof of A (by removing all bindings that have been assigned during A's test) and tries to prove the next alternative. This process is also called *back-tracking.* In order to clarify the whole strategy, we can consider the following example query (using the example clauses introduced in the previous paragraph as Prolog's database):

```
?- grandparent(bill,mary).
```

The only clause that can match this query is the following rule:

```
grandparent(X,Z) :- parent(X,Y),
    parent(Y,Z).
```

and unification of the query with the rule's head will return the following bindings: `X = bill, Z =`

mary. In order to prove the rule, the two elements of the rule body have to be proven from left to right. Note that both rules share variables with the rule head, and, therefore, the bindings computed during the match of the head with the query are also available for the respective subqueries. Thus, the first subquery is actually instantiated as parent(bill,Y) and the second subquery is instantiated as parent(Y,mary). Now, to prove the first clause, Prolog finds two alternative parent rules. Let us assume that Prolog chooses the first alternative (in order to remember that more than one alternative is possible, Prolog sets a *choice point*),

```
parent(X,Y) :- mother(X, Y).
```

Unification of the subquery with the rule head is easily possible and will bind the X variable to the term bill. This *partially* instantiates the single body element as mother(bill,Y). Unfortunately, there are no facts in the database which validate this subquery. Because the unification of mother(bill,Y) fails, so does the whole rule. Then, Prolog back-tracks to the choice point where it selected the first possible parent rule and chooses the second alternative,

```
parent(X,Y) :- father(X, Y).
```

Unification of the (still active) subquery parent (bill,Y) will instantiate father(bill,Y). This time unification is possible, returning the binding Y = john. Now the first parent subquery of the grandparent rule has been proven and the actual variables are X = bill, Y = john, Z = mary. This instantiates the second element of the grandparent rule body to parent(john,mary) (note that the Z value had already been bound after the grandparent rule was selected). The same strategy is then applied for this subquery, and Prolog will find enough facts to prove it successfully. Since both body elements of the grandparent rule have been proven to be valid, Prolog concludes that the initial query is also true.

## D. Prolog Extensions

In order to use Prolog for practical programming, it comes with a number of extensions, e.g., list data structures; operators for explicitly controlling the traversal of the search space by a Prolog program (namely, the cut operator); and routines for IO interfaces, tracing, and debugging. We cannot describe all these extensions in the context of this short article. We will only briefly show how lists can be used in Prolog.

Prolog supports lists as a basic data structure using conventional syntax. The list elements are separated by commas. The whole list is delimited by square brackets. A list element can be an arbitrary term or a list itself. Thus, it is quite similar to the list structures in Lisp. Here is an example of a Prolog list:

```
[john, mary, bill]
```

The empty list is represented as []. In order to be able to create or traverse lists, Prolog provides a special construction for explicitly denoting the head and tail of a list. [X | Y] is a list consisting of a head X and a tail Y. For example, the above list could also be specified as

```
[john | mary, bill]
```

We will use the member predicate as an example of how lists are treated in Prolog. This predicate will determine whether a given element occurs in a given list. Using the above notation, an element is in a list if it is the head of that list or if it occurs somewhere in the tail of the list. Using this informal definition of the member predicate, we can formulate the following Prolog program (the symbol _ denotes an *anonymous* variable, used to tell Prolog that it does not matter which value the unifier binds to it):

```
member(Element,[Element | _]).
member(Element,[_ | List]) :-
    member(Element,List).
```

Assuming the following query:

```
?- member(a, [b,c,a,d]).
```

Prolog will first check whether the head of [b | c,a,d] is equal to a. This causes the first clause to fail, so the second clause is tried. This will instantiate the subquery member(a, [c,a,d]), which means that the first list element is simply skipped. Recursively applying member, Prolog tries to prove whether the head of [c | a,d] is equal to a which also fails, leading to a new subquery member(a,[a,d]) through instantiation of the second clause. The next recursive step will check the list [a | d]. This time, a is indeed equal to the head element of this list, so Prolog will terminate with “yes.”

## E. Constraint Logic Programming

Constraint logic programming (CLP) is a generalization of the (simple) Prolog programming style. In CLP, term unification is generalized to constraint solving. In constraint logic programs, basic components of a problem are stated as constraints (i.e., the structure of the objects in question) and the problem as a whole is represented by putting the various constraints together by means of rules (basically by means of definite clauses). For example, the following definite clause—representing a tiny fraction of a Natural Language (NL) grammar like English:

$$
\begin{array}{l}
\texttt{sign}(X_0) \leftarrow \\
\quad \texttt{sign}(X_1), \\
\quad \texttt{sign}(X_2), \\
\quad X_0 \ \texttt{syn cat} \doteq \texttt{s}, \\
\quad X_1 \ \texttt{syn cat} \doteq \texttt{np}, \\
\quad X_2 \ \texttt{syn cat} \doteq \texttt{vp}, \\
\quad X_1 \ \texttt{syn agr} \doteq X_2 \ \texttt{syn agr}
\end{array}
$$

expresses that for a linguistic object to be classified as an *S*(entence) phrase it must be composed of an object classified as an *NP* (nominal phrase) and by an object classified as a *VP* (verbal phrase) and the agreement information (e.g., person, case) between *NP* and *VP* must be the same. All objects that fulfill at least these constraints are members of *S* objects. Note that there is no ordering presupposed for *NP* and *VP* as is the case for NL grammarbased formalisms that rely on a context-free backbone. If such a restriction is required, additional constraints have to be added to the rule, for instance, that substrings have to be combined by concatenation. Since the constraints in the example above only specify necessary conditions for an object of class *S,* they express partial information. This is very important for knowledge-based reasoning, because in general we have only partial information about the world we want to reason with. Processing of such specifications is then based upon constraint solving and the logic programming paradigm. Because unification is but a special case of constraint solving, constraint logic programs have superior expressive power.

A number of constraint-based logic programming languages (together with high-level user interface and development tools) have been realized, e.g., CHIP or the Oz language, which supports declarative programming, object-oriented programming, constraint programming, and concurrency as part of a coherent whole. Oz is a powerful constraint language with logic variables, finite domains, finite sets, rational trees, and record constraints. It goes beyond Horn clauses to provide a unique and flexible approach to logic programming. Oz distinguishes between directed and undirected styles of declarative logic programming.

## V. OTHER PROGRAMMING APPROACHES

In this article, we have compared AI languages with imperative programming approaches. Object-oriented languages belong to another well-known programming paradigm. In such languages the primary means for specifying problems is to specify abstract data structures also called objects or classes. A class consists of a data structure together with its main operations, often called *methods.* An important characteristic is that it is possible to arrange classes in a hierarchy consisting of classes and subclasses. A subclass can inherit properties of its superclasses, which support modularity. Popular object-oriented languages are Eiffel, C++, and Java. The Common Lisp Object-Oriented System is an extension of Common Lisp. It supports full integration of functional and object-oriented programming. Recently, Java has become quite popular in some areas of AI, especially for intelligent agent technology, Internet search engines, or data mining. Java is based on C++ and is the main language for the programming of Internet applications. Language features that makes Java interesting from an AI perspective are its built-in automatic garbage collection and multi-threading mechanism.

With the increase of research in the area of Web intelligence, a new programming paradigm is emerging, viz. *agent-oriented programming.* Agent-oriented programming (AOP) is a fairly new programming paradigm that supports a societal view of computation. In AOP, objects known as agents interact to achieve individual goals. Agents can exist in a structure as complex as a global Internet or as simple as a module of a common program. Agents can be autonomous entities, deciding their next step without the interference of a user, or they can be controllable, serving as a mediary between the user and another agent. Since agents are viewed as living, evolving software entities, there seems also to emerge a shift from the more language programming point of view toward a more software platform development point of view. Here the emphasis is on system design, development platforms, and connectivity. Critical questions are then how the rich number of existing AI resources developed in different languages and platforms can be integrated with other resources making use of modern system development tools such as CORBA (Common Object Request Broker Architecture), generic abstract data type and annotation languages such as XML, and a standardized agent-oriented communication language such as KQML (Knowledge Query and Manipulation Language). So the future of AI programming might be less concerned with questions such as "what is the best suited programming paradigm?", but will have to find answers for questions such as "how can I integrate different programming paradigms under one umbrella?" and "what are the best communication languages for intelligent autonomous software modules?"

## SEE ALSO THE FOLLOWING ARTICLES

Engineering, Artificial Intelligence in • Evolutionary Algorithms • Expert Systems Construction • Industry, Artificial Intelligence in • Medicine, Artificial Intelligence in • Object-Oriented Programming • Programming Languages Classification

## BIBLIOGRAPHY

Charniak, E., Riesbeck, C. K., McDermott, D. V., and Meehan, J. R. (1980). *Artificial Intelligence Programming.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Clocksin, W. F., and Mellish, C. S. (1987). *Programming in Prolog.* Berlin: Springer-Verlag.

Keene, S. E. (1988). *Object-Oriented Programming in Common Lisp.* Reading, MA: Addison-Wesley.

Luger, G. F., and Stubblefield, W. A. (1993). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving,* 2nd ed. Redwood City, CA: Benjamin/Cummings.

Norvig, P. (1992). *Artificial Intelligence Programming.* San Mateo, CA: Morgan Kaufman.

Pereira, F. C. N., and Shieber, S. M. (1987). *Prolog and Natural Language Analysis,* CSLI Lecture Notes, Number 10. Stanford, CA: Stanford Univ. Press.

Sebesta, R. W. (1999). *Concepts of Programming Languages,* 4th ed. Reading, MA: Addison-Wesley.

Ullman, J. D. (1997). *Elements of ML Programming,* 2nd ed. Englewood Cliffs, NJ: Prentice-Hall.

Watson, M. (1997). *Intelligent Java Applications for the Internet and Intranets.* San Mateo, CA: Morgan Kaufman.

# Automata Theory

**Sergio de Agostino and Raymond Greenlaw**
*Armstrong Atlantic State University*

## GLOSSARY

**deterministic** Having an unambiguous state. The machine either has a unique next state or none at all.

**deterministic finite automaton (DFA)** A simple, theoretical machine with one read-only input tape that reads only left-to-right.

**deterministic pushdown automaton (DPDA)** A DFA that is extended by adding a stack data structure.

**deterministic Turing machine (DTM)** A DFA that is extended by adding a read/write work tape.

**$\Lambda$-transition** The ability of a machine to change state without reading from an input tape or advancing the input head.

**language accepted (by a DFA)** The set of all strings recognized by a DFA.

**nondeterministic** The machine may have a non-unique next state.

**nondeterministic finite automaton (NFA)** A simple, theoretical, nondeterministic machine with one read-only input tape that reads only left-to-right.

**nondeterministic Turing machine** A DFA that is extended by adding a read/write work tape and nondeterminism.

**nondeterministic pushdown automaton (NPDA)** A DPDA extended with nondeterminism.

**regular expression** A formalism that can be used to model languages recognized by DFAs.

**tape configuration** The contents of a tape and the position of the tape head on the tape.

**work tape** A tape that can be written-to and read-from, allowing storage of intermediate data.

## I. INTRODUCTION

A large part of theoretical computer science is dedicated to studying *computational models*—also known as idealized computers. Such machines consist of a finite control (processor) and a number of tapes (memory). These machines differ in terms of their number of tapes and the functionality of their tapes. Figure 1 depicts such an idealized model. The purpose of a computational model is to capture those aspects of computation that are relevant to the particular problem you wish to solve, while hiding the other aspects that are unimportant. One can think of a computational model as a custom machine designed for your particular needs. Several of the most important models are the *deterministic finite automaton,* the *nondeterministic finite automaton,* the *deterministic pushdown automaton,* the *nondeterministic pushdown automaton,* the *deterministic Turing machine,* and the *nondeterministic Turing machine.* Each of these models is significant in the theory of computation.

Before we can attempt to solve a specific problem by a machine, we must communicate it to the machine. We do this with a language. In very general terms, a language is a system of signs used to communicate information between one or more parties. Thus, the first step of understanding a problem is to design a language for communicating that problem to a machine. Since a problem requires an answer, the language has to handle both input and output communication. What is interesting is that from a theory-of-computation perspective, almost all prob-

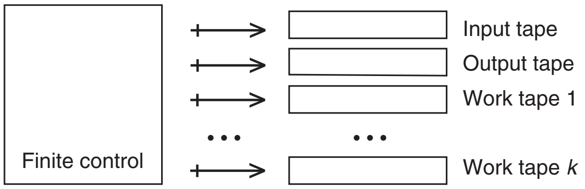


**Figure 1** Schematic of a finite state machine. The ↦ symbol denotes sequential access to a tape.

lems can be expressed just in terms of language *recognition.* That is, the language simply describes all the legitimate inputs and there is no output to communicate other than the binary value of YES or NO, depending on whether the input was in the language or not. A second common and powerful mechanism to describe languages is provided by the notion of *grammar.* We will also discuss how grammars relate to languages and automata.

We begin by presenting some background and then by defining the most basic computational models—the deterministic finite automaton and its nondeterministic variant. The other models can easily be described as enhanced versions of either the deterministic or nondeterministic finite automaton.

## II. BASIC CONCEPTS

An *alphabet* is a nonempty, finite set of symbols. A *string* is a sequence of symbols over an alphabet. The *empty string* consists of a string of 0 symbols and is denoted $\Lambda$. A *language* over an alphabet $\Sigma$ is any finite or infinite set of strings over $\Sigma$. Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The concatenation of two languages $L_1$ and $L_2$ is the set of all strings obtained by concatenating any element of $L_1$ with any element of $L_2$; specifically,

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define $L^n$ as $L$ concatenated with itself $n$ times, with the special case $L^0 = \{\Lambda\}$ for every language $L$. Finally, we define the *star closure* of $L$ as $L^* = \mathrm{L}^0 \cup L^1 \cup L^2 \cdots$, and the *positive closure* as $L^+ = L^1 \cup L^2 \cdots$.

Consider the language $L = \{\Lambda, 00, 0000, 000000, \ldots\}$ over the alphabet $\{0,1\}$. What would a program look like that accepts this language? First of all, what exactly do we mean by this question? Suppose a user inputs a given string $x$ over the alphabet $\{0,1\}$ to a program designed to accept this language. Then the program should respond YES if $x \in L$, and NO otherwise. This is really just answering a membership question. Notice, the language $L$ consists of all strings that contain an even number of 0's and no 1's. You could imagine a C program that keeps track of whether an even number of 0's has been encountered in the input and also whether or not a 1 has been seen. When the end of the input is reached, if an even number of 0's have been read and no 1 has been detected, then the program should respond YES and otherwise it should respond NO.

There is a useful way of diagraming such a program and this method is shown in Fig. 2; the picture is called a (*state*) *transition diagram.* The transition diagram is a directed graph enhanced with the >> symbol and some labeling on the nodes and edges. The >> symbol indicates the state that the machine starts from, called the *initial state.* This transition diagram consists of three nodes called *states* labeled $q_0$, $q_1$, and $q_2$; and six edges called *transitions.* Notice the edges are labeled by 0 or 1, symbols from the alphabet in consideration. The node with two circles, $q_0$, represents an *accepting state,* which coincidentally in this case is also the initial state.

Consider an "input" to the transition diagram such as 0000. If we trace through the figure on this input starting from state $q_0$, we go to state $q_1$ using up one 0, back to $q_0$ using up another 0, back to $q_1$ using the third 0, and finally back to $q_0$ at which point we have no more input remaining. The state we ended up in, $q_0$, is an accepting state so the input 0000 is *accepted.* That is, $0000 \in L$. In tracing the machine on input 10, from state $q_0$ we go to state $q_2$ using up the 1 and then stay in state $q_2$ using up the last symbol 0. Because $q_2$ does not have a double circle, it is not an accepting state; and therefore, the machine does not stop in an accepting state. The input 10 is *rejected* as it should be because $10 \notin L$. Automata theory formalizes these ideas.

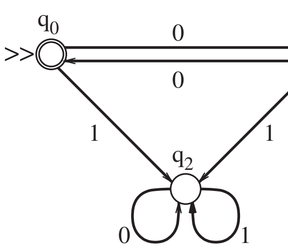


**Figure 2** A sample *state transition diagram* useful for representing a machine that accepts the language $\{\Lambda, 00, 0000, 000000, \ldots\}$.

## A. Tapes

In the example just sketched we did not say how the input was going to be represented. All of our machines store information as strings on tapes. These tapes are divided into *tape cells* or *tape squares,* or just *cells* or *squares* for short. Each square holds a single symbol. The tape has a special organization. By convention the tape always contains two special symbols, called the *left* and *right end markers,* and denoted by the < and > symbols, respectively. A tape begins with the left end marker, and ends with the right end marker. Between these two markers, all symbols present on the tape must come from the *data alphabet.*

The end markers are never part of the data alphabet. The data alphabet plus the end markers constitutes the *tape alphabet.* If the data alphabet is $\Sigma$ then we denote the tape alphabet (that is, $\Sigma \cup \{<,>\}$) by $\Sigma_T$. The *contents of a tape* is the string over the data alphabet that appears between the end markers. Thus a tape can contain the empty string if all it has is the left end marker followed immediately by the right end marker.

The input to a machine is a string, usually denoted $x$, over the data alphabet. The notation $x(i)$ is used to refer to the $i$th symbol of $x$, where $i$ ranges from 1 to the length of $x$. The input is placed on an *input tape.* The individual characters comprising $x$ appear in adjacent cells of the tape; the entire input is between end markers. The input tape is special in that it is *read-only.* Since the machine cannot write to the tape, nothing can ever be stored on the tape except the original input. Figure 3 provides an illustration of a sample input tape. In this case the input $x$ is the string 110101. The length of $x$ is six, and, for example, $x(1)$ equals 1 and $x(5)$ equals 0.

Machines access a tape via a *tape head.* Initially when the machine starts up, the tape head is always positioned to the first square immediately to the right of the left end marker. Thus if the tape with contents $x$ is nonempty, the tape head is over $x(1)$ so that the first symbol of $x$ can be read. If $x$ is the empty string, $\Lambda$, then the tape head will be positioned over the right end marker. Figure 4 illustrates the tape head for the input tape and its initial positioning.

Further constraints can apply to the motion of the tape head. For example, some machines can only read the input once while others can go over the input as many times as desired; we will specify each input access method in turn.

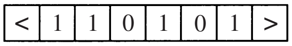


**Figure 3** An illustration of an input tape. < and > are left and right end markers, respectively.

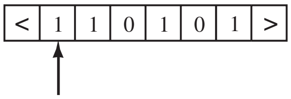


**Figure 4** An illustration of the initial position of the input tape head. At this point the machine would be reading a 1 off the input tape.

When we describe an ongoing computation of a machine, we must be able to describe the contents of a tape and the current position of the tape head. These two pieces of information constitute the *tape configuration.*

**Definition 1**

A *tape configuration* is a pair $[p,x]$. The head position $p$, with $0 \leq p \leq |x| + 1$, is a number indicating the position of the tape head on a tape with contents $x$.

- When $p = 0$ the tape head is over the left end marker,
- when $p = |x| + 1$ the tape head is over the right end marker, and
- otherwise the tape head is over the symbol $x(p)$.

The *current tape symbol* in a tape configuration is the symbol under the tape head, and is denoted $\sigma[p,x]$. The *remaining tape* in a tape configuration $[p,x]$ is the contents of the tape from under the tape head up to but not including the right tape marker. The remaining tape, denoted by $\rho([p,x])$, is the string

- $\Lambda$ if $p = |x| + 1$, and
- otherwise it is $x(p)x(p+1) \cdots x(|x|)$.

The *initial tape configuration* is $[1,x]$, also denoted by $\tau^I(x)$. The *final tape configuration* is $[|x| + 1, x]$, also denoted by $\tau^F(x)$.

In general, machines can also have *work tapes.* These are tapes that can be written to and read from, and so are used to store the intermediate results of a computation. All work tapes are initially blank. When the machine writes to a work tape, it writes a new symbol onto the tape square under the tape head, replacing the old contents of the square. If the tape square under the tape head is an end marker, the writing of a symbol also extends the tape by one cell, effectively moving the tape marker to the left or right depending on what end of the tape the head is at. If the tape head is somewhere between the end markers when the write of an end marker occurs, the result is that the tape is truncated so that the new contents of the tape are the squares between the new end marker and its match. Overwriting > with < or vice versa is not permitted.

## B. Finite Controls

Tapes provide us with a model of how we present a machine with input, store intermediate data, and generate output. Now we have to consider the machine itself. How do we program it? What are the instructions of the machine? How do we know what instruction to execute next? What is the syntax of an instruction? What are the semantics of an instruction, that is, what happens when you execute an instruction?

The *finite control* or *state transition function* is the key mechanism for defining all the machine models we study. It is the machine's program and defines how the machine computes. One of the most important features of the finite control is the fact that it must be *finite*. This limits the machine to a finite set of instructions; this fixed number of instructions is independent of the input.

Just like the program counter on a real machine, the current *state* of a finite control helps determine the next instruction to be executed by the machine. When an instruction is executed, it does any one or more of the following, depending on the type of machine: reads a symbol under a tape head, writes a symbol to the tape cell under a tape head, and moves a tape head. The instruction then determines the next state or states of the program and changes the current state.

If the next state of a particular machine is always unambiguous (there is either a unique next state, or none at all) then the machine is said to be *deterministic*. If there are circumstances during the execution of the machine in which the next instruction is ambiguous (there are two or more possible next states) then the machine is said to be *nondeterministic*. The finite control of a deterministic machine is usually denoted $\delta$. The finite control of a nondeterministic machine is usually denoted by $\Delta$. The precise details of the state transition function for a machine depend on other specifications of the machine such as how many tapes it has, and their input/output capabilities. In general, finite-state control, tape-memory machines can be classified according to the answers to the following questions:

1. Is the machine deterministic or nondeterministic?
2. Are state transitions labeled with a single symbol from $\Sigma_T$, or a string from $\Sigma_T^*$ and is $\Lambda$ allowed as a label? Must there be a transition for every possible symbol for every possible state?
3. How many tapes are there, how do the heads move on the tapes, and which ones are read-only?

# III. DETERMINISTIC FINITE AUTOMATA (DFAs)

The deterministic finite automaton or DFA is a very simple machine. It has one read-only input tape, with the restriction that the tape head can only move from left to right and can never change direction. The DFA has no other tapes. The finite control allows a DFA to read one input symbol from the input tape and then based on the machine's current state, it may change state. As part of each computational step of a DFA, the input tape head is automatically repositioned one square further to the right and is ready for reading the next input symbol.

For example, in Fig. 2 the states are represented by the circles. One part of the finite control corresponding to Fig. 2 is the *transition* $\delta(q_0,0) = q_1$. That is, when in state $q_0$ and reading a 0 the machine transfers to state $q_1$. The input head is then automatically moved one square to the right (moving to the right of the right end marker causes the machine to fail). The transition $\delta(q_0,1) = q_2$ specifies that while in state $q_0$ and reading a 1 transfer to state $q_2$. Again the input head is automatically moved one square to the right. The remainder of the finite control for the transition diagram shown in Fig. 2 is specified similarly. The finite control is shown in its entirety in tabular form in Table I. Each row in such a table represents one possible transition of $M$. This table is called the *transition table* of $M$. Note that there are no entries in the table to indicate what the next state should be when the input head is over an end marker. In such a case where there is no next state the machine simply stops.

We are now ready to present the formal definition of a DFA. The description of the DFA is presented as a five-tuple so that the order of the components is fixed.

### Definition 2

A deterministic finite automaton *(DFA)* is a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ with the components specified as follows:

**Table I** A Convenient Method for Representing the Transition Function of a DFA

| Transition number | State | Input symbol | New state |
|---|---|---|---|
| 1 | $q_0$ | 0 | $q_1$ |
| 2 | $q_0$ | 1 | $q_2$ |
| 3 | $q_1$ | 0 | $q_0$ |
| 4 | $q_1$ | 1 | $q_2$ |
| 5 | $q_2$ | 0 | $q_2$ |
| 6 | $q_2$ | 1 | $q_2$ |

Note: The transition table for the DFA presented in Fig. 2 is shown here. The transitions are numbered for convenience but this numbering is not part of the finite control.

1. $Q$: A finite, nonempty set of *states.*
2. $\Sigma$: The *data alphabet* and its induced *tape alphabet* $\Sigma_T = \Sigma \cup \{<,>\}$.
3. $\delta$: The *transition function* or *finite control* is a function
$$\delta : Q \times \Sigma_T \mapsto Q.$$
4. $q_0$: The *initial state* or *start state,* $q_0 \in Q$.
5. $F$: The set of *accepting states,* $F \subseteq Q$.

The set of states is denoted $Q$. Note that $Q$ is finite and nonempty.

The data alphabet is denoted $\Sigma$. These are the symbols that can occur on the input tape between < and >. End markers are not allowed as data symbols. The tape alphabet $\Sigma_T$ is the set of all possible symbols that appear on the tape, and so it is $\Sigma$ union the set $\{<,>\}$.

We defer the description of $\delta$ for the moment.

The initial state is denoted $q_0$. This is a special state in $Q$ and is the state from which $M$ begins executing. Note the initial state is *not* expressed as a set like the other components in the definition.

$F$ is the nonempty set of accepting states. These special states are used by a DFA to signal when it accepts its input, if in fact it does. When the machine stops in a nonaccepting state this signifies the input is rejected. The notion of *acceptance* is described formally in Definition 6.

Where does the input tape appear in the definition? The tape is utilized in the transition function $\delta$. The domain of $\delta$ is $Q \times \Sigma_T$ so elements in the domain of $\delta$ are ordered pairs. That is, $\delta$ takes a state and a symbol from the input tape (possibly an end marker). Note, the more complex models we present later make useful transitions on the end markers. The restrictions placed on the DFA do not allow it to take advantage of the end markers. Therefore, we only show $\delta$ being defined on $Q \times \Sigma$ in our examples. A typical argument to $\delta$ would be (0,1). Using standard function notation we would write $\delta((0,1))$ to signify $\delta$ being applied to its arguments. To simplify notation, we drop the "extra" set of parentheses keeping in mind that the arguments to $\delta$ are really ordered pairs. So, for example, we write $\delta(0,1)$. The range of $\delta$ is $Q$.

Suppose $q \in Q$, $a \in \Sigma_T$, and $\delta(q, a) = q'$, where $q' \in Q$. This is called a *transition* of $M$. This transition moves $M$ from state $q$ into state $q'$ on reading an $a$, and the input head is then moved one square to the right. In Fig. 2 transitions were represented by edges between states and labeled with input tape symbols. Since $\delta$ is a function, DFAs behave deterministically. Another way of saying this is that the machine has only one "choice" for its next transition, just like a typical C program must execute a unique next instruction. The complete specification for the DFA shown in Fig. 2 is given below.

### Example 1

Formal specification of a DFA.

The five-tuple for the DFA M shown in Fig. 2 is as follows:

$M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_0\})$, where $\delta$ is defined as in the transition table shown in Table I or equivalently expressed as

$$\{(q_0, 0, q_1), (q_0, 1, q_2), (q_1, 0, q_0), (q_1, 1, q_2), (q_2, 0, q_2), (q_2, 1, q_2)\}.$$

Here we have written the function $\delta : Q \times \Sigma_T \mapsto Q$ as triples in $Q \times \Sigma_T \times Q$.

In order to describe a computation of a DFA we need to be able to specify snapshots of the machine detailing where the machine is in its computation. What are the important ingredients in these snapshots? They are the configuration of the input tape and the current state of $M$. Such a snapshot is called a *configuration* of the machine.

### Definition 3

A *configuration* of a DFA $M = (Q, \Sigma, \delta, q_0, F)$ on input $x \in \Sigma$ is a two-tuple $(q, [p, x])$, where $q \in Q$ and $[p,x]$ is a configuration of the input tape. The *initial configuration* of $M$ on input $x$ is the configuration $(q_0, [1, x])$, or equivalently $(q_0,\tau^I(x))$. We use $C_0$ to denote the initial configuration when $M$ and $x$ are understood. For machine $M$, the set of all possible configurations for all possible inputs $x$ is denoted by $\mathcal{C}(M)$.

How can we utilize the notion of configuration to discuss the computation of a DFA? They help us define the *next move* relation, denoted $\vdash_M$, as shown in the following.

**Definition 4**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Let $\mathcal{C}(M)$ be the set of all configurations of $M$. Let $C_1 = (q_1, [p_1, x])$ and $C_2 = (q_2, [p_2, x])$ be two elements of $\mathcal{C}(M)$. $C_1 \vdash_M C_2$ if and only if $p_2 = p_1 + 1$ and there is a transition $\delta\,(q_1, \sigma[p_1, x]) = q_2$. The relation $\vdash_M$ is called the *next move, step,* or *yields* relation.

Notice $\vdash_M$ is a relation defined on configurations. This means $\vdash_M \subseteq \mathcal{C}(M) \times \mathcal{C}(M)$. Since $\delta$ is a function, $\vdash_M$ is also a function. Definition 4 is saying that configuration $C_1$ yields configuration $C_2$ if there is a transition from $C_1$ that when executed brings $M$ to the new configuration $C_2$.

As an example consider the DFA, call it $M$, whose transition function was depicted in Table I. The initial configuration of $M$ on input $x = 0011$ is $(q_0, [1, 0011])$. Applying transition 1 from Table I, we see

$$(q_0, [1, 0011]) \vdash_M (q_1, [2, 0011]).$$

The machine read a 0 and moved to state $q_1$. Continuing this *trace* (formally defined in Definition 5), we obtain the following series of configurations:

$$\begin{aligned}(q_1, [1, 0011]) &\vdash_M (q_0, [2, 0011\}) \text{ (by transition 3)}\\ &\vdash_M (q_2, [3, 0011]) \text{ (by transition 2)}\\ &\vdash_M (q_2, [4, 0011]) \text{ (by transition 6)}\end{aligned}$$

We say the DFA *halts* when there is no next state or when the machine moves off the end of the tape. This can occur whenever the state transition function is undefined. A *halting configuration* of a DFA is a configuration $C_h = (q, [p,x]) \in \mathcal{C}(M)$ with the property that $\delta\,(q, \sigma[p,x])$ is undefined. If the DFA halts when there is no more input left to process, that is, it is in a configuration $C = (q, \tau^F(x))$ then we say that the DFA is in a *final configuration*. That is, the DFA is in a configuration $C_h = (q, [p, x]) \in \mathcal{C}(M)$ with the property that $p = |x| + 1$.

The relation $\vdash_M$ was defined to aid in assisting with the descriptions of computations. But $\vdash_M$ stands for only one step. We would like to discuss computations of varying lengths including length zero.

**Definition 5**

Let $M$ be a DFA with next move relation $\vdash_M$. Let $C_i \in \mathcal{C}(M)$, for $0 \leq i \leq n$. Define $\vdash_M^*$ to be the reflexive, transitive closure of the relation $\vdash_M$. $C_0$ *yields* or *leads to* $C_n$ if $C_0 \vdash_M^* C_n$. A *computation* or *trace* of $M$ is a sequence ofconfigurations related by $\vdash_M$ as follows: $C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_n$. This computation has *length n* or we say it has *n steps*. Sometimes, we write $C_0 \vdash_M^n C_n$ to indicate a computation from $C_0$ to $C_n$ of length $n$.

Notice that on an input $x$ of length $n$, a DFA will run for at most $n + 1$ steps. If the state transition function is defined on every state and data symbol then the DFA will process its entire input. For the four step computation traced above, we can write

$$(q_0, [1{,}0011]) \vdash_M^* (q_2, [5, 0011]) \quad \text{or}$$

$$(q_0, [1, 0011]) \vdash_M^4 (q_2, [5, 0011])$$

with $(q_2, [5, 0011])$ the final configuration.

We would like to describe the computational capabilities of DFAs in terms of the languages they accept. First, we need to define what it means for a DFA to *accept* its input. The idea is simply that the machine reads all of its input and ends up in an accepting state.

**Definition 6**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and $q \in Q$. $M$ *accepts* input $x \in \Sigma^*$ if

$$(q_0, \tau^I(x)) \vdash_M^* (f, \tau^F(x)),$$

where $f \in F$. This computation is called an *accepting computation.* A halting configuration $(q, \tau^F(x))$ is called an *accepting configuration* of $M$ if $q \in F$. If $M$ does not accept its input $x$, then $M$ is said to reject $x$. The computation of $M$ on input $x$ in this case is called a *rejecting computation,* and $M$ was left in a *rejecting configuration.*

$M$ begins computing in its initial state, with the input tape head scanning the first symbol of $x$, and $x$ written on the input tape. If $M$ reads all of $x$ and ends in an accepting state, it accepts. It is important to note that $M$ reads its input only once and in an *on-line* fashion. This means $M$ reads the input once from left to right and then must decide what to do with it. $M$ cannot go back and look at the input again. In addition, even though $M$ can sense the end of the input by detecting the > marker, this is only useful if $M$ can reverse directions on the input tape. Thus $M$ must be prepared to make a decision about accepting or rejecting assuming that the input might be exhausted after the symbol just read.

As an example, the DFA with the transition function as shown in Fig. 2 accepts the input $x = 0011$ since $q_0 \in F$ and $(q_0, [1, 0011]) \vdash^* (q_0, [5{,}0011])$. We can now define the *language accepted* by a DFA $M$. Informally, this is simply the set of all strings accepted by $M$.

### Definition 7

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The *language accepted* by $M$, denoted $L(M)$, is $\{x \mid M \text{ accepts } x\}$. The union of all languages accepted by DFAs is denoted $L_{\mathbf{DFA}}$. That is,

$$L_{\mathbf{DFA}} = \{L \mid \text{there is a DFA } M \text{ with } L = L(M)\}.$$

The DFA shown in Fig. 2 accepts the language $\{\Lambda$, 00, 0000, 000000, . . .$\}$. It follows that this language is in $L_{\mathbf{DFA}}$. Let us look now at a typical application of DFAs.

### Example 2

Application of DFAs involving searching a text for a specified pattern.

DFAs are useful for pattern matching. Here we consider the problem of searching for a given pattern $x$ in a file of text. Assume our alphabet is $\{a, b, c\}$. This example can easily be generalized to larger alphabets. To further simplify the discussion let $x$ be the string abac. The techniques used here can be applied to any other string $x$. Formally, we want to build a DFA that accepts the language

$$\{s \mid s \in \{a, b, c\}^* \text{ and } s \text{ contains the pattern } abac\}.$$

The idea is to begin by hard coding the pattern $x$ into the states of the machine. This is illustrated in Fig. 5A. Since the pattern *abac* has length four, four states are needed in addition to the initial state, $q_0$, to remember the pattern. Think of each state as signifying that a certain amount of progress has been made so far in locating the pattern. So, for example, on reaching state $q_2$ the machine remembers that *ab* has been read.

We can only reach state $q_4$ if we have read the pattern *abac* so $q_4$ is the only accepting state required. The next step is to fill in the remaining transitions on other characters in the alphabet. The complete DFA is shown in Fig. 5B. Notice how in the figure there are some edges with more than one label. This simply means that the corresponding transition can be applied when reading any one of the symbols labeling the transition.

We now explain how the extra transitions were added by examining state $q_3$. The following methodology can be applied in a similar fashion to the other states. From state $q_3$ on reading a "*c*," we enter the accepting state specifying that the pattern was indeed found; this is why state $q_4$ is an accepting state. From state $q_3$ on reading an "*a*," we transition back to state $q_1$. This is because the "*a*" could be the start of the pattern *abac*. That is, we can make use of this "*a*." If we read a "*b*" from the state $q_3$, then we need to transition all the way back to state $q_0$. The "*b*" nullifies all of the progress we had made and we must now start over from the beginning.

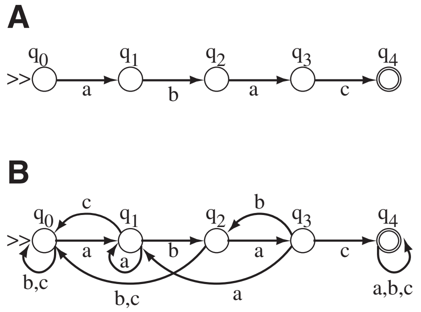


**Figure 5** Steps in constructing a DFA to a recognize a pattern $x$ in a file of text. In this case corresponding to Example 2, $x$ equals *abac*. Part (A) shows how to begin by hard coding the pattern into the machine's states. Part (B) shows the complete DFA.

The complete description of the DFA for recognizing strings containing the pattern $x$ equals *abac* over the alphabet $\{a, b, c\}$ is $(\{q_0, q_1, q_2, q_3, q_4\}, \{a, b, c\}, \delta, q_0, \{q_4\})$, where $\delta$ is as shown in Table II. One point worth noting is that once a pattern is found (that is, the first time an accepting state is entered), the text editor can notify the user of the pattern's location rather than continuing to process the remainder of the file. This is usually what text editors do.

**Table II** The Transition Table for the DFA Described in Example 2 and Shown in Fig. 5

| State | Input symbol | New state |
|---|---|---|
| $q_0$ | $a$ | $q_1$ |
| $q_0$ | $b$ | $q_0$ |
| $q_0$ | $c$ | $q_0$ |
| $q_1$ | $a$ | $q_1$ |
| $q_1$ | $b$ | $q_2$ |
| $q_1$ | $c$ | $q_0$ |
| $q_2$ | $a$ | $q_3$ |
| $q_2$ | $b$ | $q_0$ |
| $q_2$ | $c$ | $q_0$ |
| $q_3$ | $a$ | $q_1$ |
| $q_3$ | $b$ | $q_0$ |
| $q_3$ | $c$ | $q_4$ |
| $q_4$ | $a$ | $q_4$ |
| $q_4$ | $b$ | $q_4$ |
| $q_4$ | $c$ | $q_4$ |

## IV. NONDETERMINISTIC FINITE AUTOMATA (NFAs)

In this section we define the nondeterministic finite automata (NFAs). A DFA being deterministic has only one computational thread. However, an NFA, because any given configuration may have many possible next configurations, cannot be described by a single computational thread. An NFA computation should be visualized as many superimposed simultaneous threads. But an NFA is not a *parallel computer*—it does not have any ability to run simultaneous computations. Instead, one can imagine the NFA behaving as follows: if the problem the NFA is solving has a solution, then the simultaneous threads will collapse into a single unique thread of computation that expresses the solution. If the NFA cannot solve the problem, the threads collapse into failure.

It is obvious that an NFA is not a machine that one can build directly. So why is it worth considering? Here are three reasons. The first is simply that this model has more expressive power than the DFA in the sense that it is easier to design NFAs than DFAs for some languages, and such NFAs usually have fewer states than the corresponding DFA. A second reason is that the abstract concept of nondeterminism has proved very important in theoretical computer science. Third, although NFA are more expressive when it comes to programming them, it turns out that any language that can be accepted by an NFA can also be accepted by a DFA. We prove this result via simulation in Section V.

Nearly all of the basic definitions about DFAs carry over to NFAs. Let us mention the enhancements to a DFA that yield the NFA model and then look at some examples. The new features in order of descending importance are the use of *nondeterminism,* the use of Λ-*transitions,* and the use of transitions on arbitrary strings.

*Nondeterminism* means that the machine could potentially have two or more different computations on the same input. For example, in Fig. 6 we show a portion of an NFA. In this NFA from state $q_0$ on reading an *a,* the machine could go to either state $q_1$ or state $q_2$. This behavior is nondeterministic and was not allowed in the DFA. In our examples we will see that this feature is very useful for designing NFAs.

A Λ-*transition* allows the machine to change state without reading from the input tape or advancing the input head. It is useful to think of such a transition as a jump or goto. Why would such a jump be useful? As an example suppose we want to recognize the language $\{a\}^* \cup \{b\}^*$ over the alphabet $\{a, b\}$.The NFA shown in Fig. 7 accepts this language. Since NFAs, like DFAs, only get to read their input once, the two Λ-transitions start two threads of computation. One thread looks for an input that is all *a*'s, the other looks for an input that is all *b*'s. If either thread accepts its input, then the NFA stops and accepts. Thus we can accept (formally defined in this section) $\{a\}^* \cup \{b\}^*$ very easily; the design is also conceptually appealing. Notice that without using Λ-transitions the machine needs three accepting states.

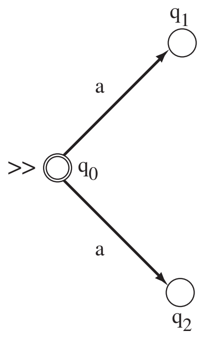


**Figure 6** A partial NFA. Notice from state $q_0$ there is a choice of either state $q_1$ or state $q_2$ on input *a.*

A DFA for accepting the language $\{a\}^* \cup \{b\}^*$ is shown in Fig. 7B. This machine has more states, transitions, and accepting states and it is more complex. It turns out that at least four states are needed for any DFA that accepts the language $\{a\}^* \cup \{b\}^*$.

Now let us look at the third enhancement to DFAs. By use of arbitrary transitions on strings we mean that a transition can be labeled with any string in Σ*. Essentially, this means an NFA is allowed to read more than one input symbol at a time (or none). How might this feature prove useful? Coupled with nondeterminism this enhancement allows us to design simpler machines. As an example recall the DFA presented in Fig. 5 that accepted the language {*x* | *x* ∈ {*a, b, c*}* and

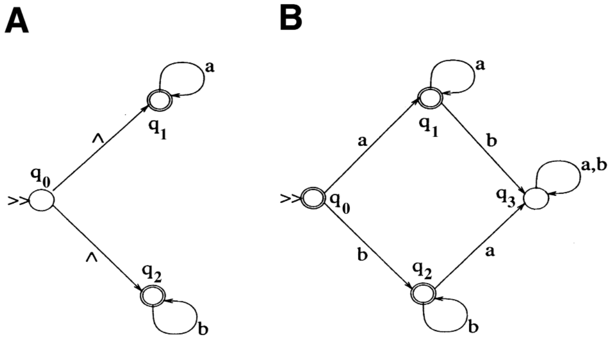


**Figure 7** Part (A) shows an NFA for accepting the language $\{a\}^* \cup \{b\}^*$. Part (B) depicts the smallest DFA for accepting the same language.

$x$ contains the pattern *abac*}. An NFA for accepting this same language is shown in Fig. 8. Until we formally define computations and acceptance for NFAs, think of this machine as gobbling up symbols unless it encounters the pattern *abac* in which case it jumps to an accepting state and then continues to gobble up symbols. We have reduced the five-state DFA from Fig. 5 to a two-state NFA using this new feature.

Rather than go through all of the definitions presented for DFAs again for NFAs, we highlight the changes in defining NFAs.

### Definition 8

A *nondeterministic finite automaton (NFA)* is a five-tuple $M = (Q, \Sigma, \Delta, q_0, F)$ that is defined similarly to a DFA except for the specification of the transitions. The *transition relation* $\Delta$ is a finite subset of $Q \times \Sigma_T^* \times Q$.

Notice $Q \times \Sigma_T^* \times Q$ is an infinite set of triples but we require $\Delta$ to be finite.

The new specification of transitions handles all of the enhancements that were discussed above. Since we now have a relation instead of a function, the machine can be nondeterministic. That is, for a given state and symbol pair it is possible for the machine to have a choice of next move. In Fig. 6, for example, the two transitions $(q_0, a, q_1)$ and $(q_0, a, q_2)$ are shown. Of course, in a DFA this pair of transitions would not be allowed.

Since $\Delta \subseteq Q \times \Sigma_T^* \times Q$ this model incorporates $\Lambda$-transitions and arbitrary string transitions. For examples, in the NFA shown in Fig. 7A the two $\Lambda$-transitions $(q_0, \Lambda, q_1)$ and $(q_0, \Lambda, q_2)$ are shown and in Fig. 8 the transition from state $q_0$ to $q_1$ is $(q_0, abac, q_1)$.

Finally, since $\Delta$ is a relation that is not total, there can be state symbols pairs for which $\Delta$ is not defined. In Fig. 7A the machine does not have a transition out of state $q_0$ on either $a$ or $b$. So, in the full representation of $\Delta$ there simply are no transitions $(q_0,a,q)$ nor $(q_0,b,q)$ for any $q \in Q$. If a thread ever entered such a state, the thread would terminate.

Nearly all of the other definitions for DFAs carry over with very little modification. For example, $\vdash$ still relates configurations but now we might have $C_1 \vdash C_2$ and $C_1 \vdash C_3$, where $C_2 \neq C_3$; a situation that was not possible in a DFA. One definition we need to rethink is that for acceptance. Since NFAs are nondeterministic, there may be several possible computation threads on the same input. We say an input is accepted if *at least one* thread leads to acceptance.

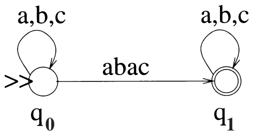


**Figure 8** An NFA for accepting the language $\{x \mid x \in \{a, b, c\}^*$ and $x$ contains the pattern *abac*$\}$.

### Definition 9

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. *M accepts* input $x \in \Sigma^*$ if $(q_0, \tau^I(x)) \vdash_M^* (f, \tau^F(x))$, for some accepting state $f \in F$. Such a computation is called an *accepting computation*. Computations that are not accepting are called *rejecting computations*. The *language* accepted by *M*, denoted $L(M)$, is $\{x \mid M \text{ accepts } x\}$. The union of all languages accepted by NFAs is denoted $L_{\mathbf{NFA}}$. That is,

$$L_{\mathbf{NFA}} = \{L \mid \text{there is an NFA } M \text{ with } L = L(M)\}.$$

On a given input an NFA may have both accepting and rejecting computations. If it has at least one accepting computation, then the input is accepted. That is, the input is accepted if at least one thread leads to an accepting state. The language accepted by an NFA consists of all strings that the NFA accepts. Let us consider an example of two possible computations of the NFA $M$ shown in Fig. 8 on input *abaca*.

The first is

$$\begin{aligned}(q_0, [1, abaca]) &\vdash_M (q_0, [2, abaca]) \\ &\vdash_M (q_0, [3, abaca]) \\ &\vdash_M (q_0, [4, abaca]) \\ &\vdash_M (q_0, [5, abaca]) \\ &\vdash_M (q_0, [6, abaca])\end{aligned}$$

and the second is

$$\begin{aligned}(q_0, [1, abaca]) &\vdash_M (q_1, [5, abaca]) \\ &\vdash_M (q_1, [6, abaca]).\end{aligned}$$

Clearly, the two computations are very different. In the first one we use up all of the input in five steps but do not end in an accepting state. Thus, this is an example of a rejecting computation. In the second case we use up all of the input in two steps and do end in an accepting state. Thus, the latter computation is accepting. Since there was an accepting computation, the input *abaca* is accepted by $M$ and $abaca \in L(M)$. To prove that an input is accepted by an NFA, one only needs to demonstrate that a *single* accepting computation exists. However, to argue that an NFA does not accept a given string, one must show that *all* possible computations are rejecting. This is usually more difficult.

## V. EQUIVALENCE OF DFAs AND NFAs

NFAs possess many features DFAs do not. These enhancements simplify the design of NFAs for certain languages. Do these new features actually make the NFA a more powerful model in the sense that it can accept languages that *no* DFA can? That is, is there some language $L$ that an NFA can accept that no DFA can? Surprisingly, DFAs and NFAs accept exactly the same class of languages. We prove this theorem below. Our treatment of this result is more detailed than that of other similar equivalences described in this article. The intention is to present one detailed argument to the reader. The specifics for other equivalences can be found in the references.

Theorem 1

*The language classes $L_{\mathbf{DFA}}$ and $L_{\mathbf{NFA}}$ are equal.*

*Proof.* ($L_{\mathbf{DFA}} \subseteq L_{\mathbf{NFA}}$) Suppose $L \in L_{\mathbf{DFA}}$. Then there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = L$. The idea is simply to view $M$ as an NFA. Define an NFA $M' = (Q, \Sigma, \Delta, q_0, F)$, where if $\delta(q,a) = q'$ then $(q,a,q') \in \Delta$. We claim that $L(M) = L(M')$. It is easy to see that $(q_0, \tau^I(x)) \vdash^*_M (f, \tau^F(x))$, where $f \in F$ if and only if $(q_0, \tau^F(x)) \vdash^*_{M'} (f, \tau^F(x))$. This simply says the machines have the same transitions, which is of course how $M'$ was defined. Since $L(M') = L$, this shows $L \in L_{\mathbf{NFA}}$ and we can conclude that $L_{\mathbf{DFA}} \subseteq L_{\mathbf{NFA}}$.

($L_{\mathbf{NFA}} \subseteq L_{\mathbf{DFA}}$) Suppose $L \in L_{\mathbf{NFA}}$. Then there exists an NFA $M = (Q, \Sigma, \Delta, q_0, F)$ such that $L(M) = L$. We will construct a DFA $M_3$ such that $L(M_3) = L$. The simulation of $M$ will take place in three stages. In each stage a new machine will be constructed that is equivalent to $M$ but is more like a DFA than in the previous phase. In the third stage the result is in fact a DFA. The first stage involves eliminating transitions of the form $(q, y, q')$, where $|y| > 1$. Stage two involves eliminating $\Lambda$-transitions. In the third stage nondeterminism is removed. From $M$ we construct $M_1$, from $M_1$ we define $M_2$, and from $M_2$ we build the desired DFA $M_3$. Figure 9 illustrates the process. Since we will show $L(M_3) = L(M)$, this is enough to complete the proof.

**Constructing $\mathbf{M_1}$:** The idea in building $M_1$ is to add new states to $M$ so that strings $y$ labeling transitions, with $|y| > 1$, can be split up into single symbol transitions as required in a DFA. The reader can think of splicing in new states in the transition diagram for any edge labeled by a string of length more than one. The five-tuple for $M_1$ is given by $M_1 = (Q_1, \Sigma, \Delta_1, q_0, F)$. The new state set $Q_1$ consists of the states from $Q$ and the additional states that we need to splice in. The relation $\Delta_1$ is defined in terms of $\Delta$ except that transitions on strings of length more than one need to be replaced by a new set of equivalent transitions. The algorithm shown in Fig. 10 describes exactly how $Q_1$ and $\Delta_1$ are constructed.

The following three facts imply that $L(M) = L(M_1)$: the set of accepting states in $M_1$ is the same as in $M$, all transitions in $\Delta$ are in $\Delta_1$ except for those on strings of length greater than one, and transitions on strings of length greater than one in $\Delta$ were replaced by transitions in $\Delta_1$ that carried out the same function.

**Constructing $\mathbf{M_2}$:** The second stage in the construction requires that we eliminate $\Lambda$-transitions from $M_1$. In the process we will define a new NFA $M_2 = (Q_2, \Sigma, \Delta_2, q_0, F_2)$. In this case $Q_2$ equals $Q_1$ and $F_2 = F \cup \{q' \mid (q', [1,\Lambda]) \vdash^*_{M_1} (f, [1,\Lambda])$ for some $f \in F\}$. This says that any state in $M_1$ from which we can reach an accepting state without using input becomes an accepting state in $M_2$. Since we are not consuming input, the empty tape configuration $[1, \Lambda]$ is sufficient. The algorithm shown in Fig. 11 shows precisely how $\Delta_2$ is constructed. The idea is to eliminate all $\Lambda$-transitions from $\Delta_1$. We replace any combination of $\Lambda$-transitions and a single transition on one symbol in $\Delta_1$ by a transition involving a single symbol in $\Delta_2$.

We now argue that $L(M_1) = L(M_2)$. Suppose $x \in L(M_1)$. Then $(q_0, \tau^I(x)) \vdash^*_{M_1} (f, \tau^F(x))$ for some $f \in F$. This computation may involve $\Lambda$-transitions. Any series of $\Lambda$-transitions that are followed by a transition in which an individual symbol is read can be replaced by a single transition of $M_2$ resulting in $M_2$ having the same configuration as $M_1$. Any combination of $\Lambda$-transitions that occur after $x$ has been completely read lead from some state $\tilde{q}$ to $f$. Because of the way in which $F_2$ was defined, we see $\tilde{q} \in F_2$ and so $x \in L(M_2)$. This shows $L(M_1) \subseteq L(M_2)$. A related argument can be used to show that $L(M_2) \subseteq L(M_1)$ essentially by re-

| $M$ | $\mapsto$ | $M_1$ | $\mapsto$ | $M_2$ | $\mapsto$ | $M_3$ |
|---|---|---|---|---|---|---|
| original NFA | | eliminate transitions of the form $(q, y, q')$, where $\lvert y\rvert > 1$ | | eliminate $\Lambda$-transitions | | eliminate nondeterminism resulting in a DFA |

**Figure 9** Illustration of the construction carried out in Theorem 1.

```
// initialize
$Q_1 \leftarrow Q$;
$\Delta_1 \leftarrow \Delta$;
// replace transitions of strings of length two or more
for each transition $(q, y, q') \in \Delta$ with $|y| > 1$ do
    $\Delta_1 \leftarrow \Delta_1 - \{(q, y, q')\}$;
    // suppose $|y| = k$ for some $k > 1$
    // let $q_1, \ldots, q_{k-1}$ be new states not in $Q_1$
    $Q_1 \leftarrow Q_1 \cup \{q_1, \ldots, q_{k-1}\}$;
    $\Delta_1 \leftarrow \Delta_1 \cup \{(q, y(1), q_1), (q_1, y(2), q_2), \ldots, (q_{k-1}, y(k), q')\}$;
```

**Figure 10** Constructing $M_1$.

versing the steps in the argument just presented. All this says is that we did not make $M_2$ accept more strings than $M_1$.

Notice if there was some state that involved only $\Lambda$-transitions in $M_1$, it is possible that after applying stage two of subset construction that this state becomes disconnected or *unreachable*. An unreachable state is one that no longer plays any role in the strings accepted by the machine since it can no longer be used in a computation. This is because there is no way to enter an unreachable state by a computation starting from the initial state. Before proceeding to stage three, we discard unreachable states from $M_2$.

**Constructing $\mathbf{M_3}$:** The third stage in the construction is to eliminate nondeterminism from $M_2$ in forming $M_3$. The idea is to consider all the possible threads of computation that $M_2$ could have active. Each thread is in a single well-defined state, which is based on the nondeterminism and the transitions that were chosen in the past by the thread. The current states over all possible threads will be compressed into a single state in $M_3$. In other words the states in $M_3$ will be the power set of the states in $M_2$ (after having removed unreachable states).

Rather than complicate notation further let us continue to refer to $Q_2$ and $F_2$ (possibly different since unreachable states may have been removed) using the same notation. We define the new machine as follows: $M_3 = (Q_3, \Sigma, \delta_3, \{q_0\}, F_3)$, where $Q_3 = 2^{Q2}$ and $F_3 = \{Q' \mid Q' \in Q_3 \text{ and } Q' \cap F_2 \neq \emptyset\}$, and $\delta_3$ is formed as described in the algorithm depicted in Fig. 12.

```
// initialize $\Delta_2$
$\Delta_2 \leftarrow \emptyset$;
// eliminate $\Lambda$-transitions
for each state $q \in Q_1$ do
    for each symbol $a \in \Sigma$ do
        $\Delta_2 \leftarrow \Delta_2 \cup \{(q, a, q') \mid (q, [1, a]) \vdash^*_{M_1} (q', [2, a])\}$;
```

**Figure 11** Constructing $M_2$.

```
$\delta_3 \leftarrow \emptyset$;
for each state $Q' \in Q_3$ do
// states of $M_3$ are sets of states of $M_2$
    for each symbol $a \in \Sigma$ do
        $R \leftarrow \emptyset$;
        // determine the set of possible states $M_2$ could be in
        for each state $q_2 \in Q_2$ do
            if $(q', a, q_2) \in \Delta_2$ for some $q' \in Q'$
            then $R = R \cup \{q_2\}$;
        // notice $R \in Q_3$
        $\delta_3 \leftarrow \delta_3 \cup \{(Q', a, R)\}$;
```

**Figure 12** Constructing $M_3$.

It is important to observe that the states of $M_3$ are sets of states of $M_2$. That is, $M_3$'s states consist of all possible subsets of states from $M_2$. The idea behind the algorithm shown in Fig. 12 is best explained via a simple example. Suppose, for example, that from state $q$, $M_2$ could on an $a$ go to either state $q_1$ or state $q_2$. That is, $M_2$ from configuration $(q, [1,as])$ for any string $s$ can make a nondeterministic move—either change to configuration $(q_1,[2,as])$ or to configuration $(q_2, [2,as])$. In $M_3$ then we want to end up in configuration $(\{q_1, q_2\},[2,as])$ since $M_2$ could be in either one of these states after reading the $a$. This is why we need sets to represent the states of $M_3$.

It is easy to see that $M_3$ is a DFA. The last statement in the code shown in Fig. 12 dictates that exactly one transition is added to the relation $\delta_3$ for each symbol and each state. Thus $\delta_3$ is a function. We must argue that $L(M_3) = L(M_2)$. Using the transitivity of equality, this will imply that $L(M_3) = L(M)$. So, this will complete the entire proof of the theorem.

First, we prove that $L(M_2) \subseteq L(M_3)$. Suppose $x \in L(M_2)$, with $n = |x|$. Then there exists a computation $(q_0,[1,x]) \vdash^*_{M_2} (f, [n+1,x])$ for some $f \in F_2$. Since $M_2$ has no $\Lambda$-transitions and no transitions on strings of length more than one, this computation passes through exactly $n - 1$ states. The if statement of the algorithm shown in Fig. 12 adds the appropriate state to the set $R$, and then in the last step of the algorithm the appropriate transition is added to $\delta_3$ keeping track of all the possible states that $M_2$ could be in. Thus for each step of the computation in $M_2$ involving a transition $(q', a, q_2)$, there are corresponding sets of states $Q'$ with $q' \in Q'$ and $Q_2$ with $q_2 \in Q_2$ in $M_3$, and a transition $(Q', a, Q_2)$ in $\Delta_3$. Therefore,

$$(\{q_0\}, [1,x]) \vdash^n_{M_3} (F',[n+1,x]),$$

where $f \in F'$. This shows that $x \in L(M_3)$, so $L(\mathrm{M}_2) \subseteq L(M_3)$. By a related argument that essentially reverses the steps in this one, we can prove that $L(M_3) \subseteq L(M_2)$.

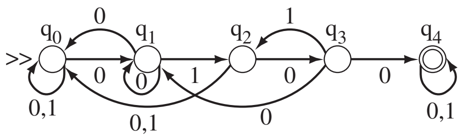


**Figure 13** A sample NFA.

It is instructive to trace an NFA on an input to examine how the set of states that the NFA can be in evolves. In Fig. 13 we show a five-state NFA. Consider this NFA on the input 010. On reading the first 0, the machine can occupy states $q_0$ or $q_1$. On reading the 1, the machine can occupy states $q_0$ or $q_2$. Finally, on reading the last 0, the machine can occupy states $q_0$, $q_1$, or $q_3$. Since none of these states are accepting, the machine rejects input 010.

## VI. EQUIVALENCE OF DFAs WITH OTHER MODELS

We call a language *regular* if it belongs to $L_{\mathbf{DFA}}$. Therefore, every regular language can be described by some DFA or some NFA. In this section we look at other ways of representing regular languages.

### A. Regular Expressions

One way of describing regular languages is via the notion of *regular expressions*. The notation for regular expressions involves a combination of strings of symbols from some alphabet $\Sigma$, parentheses, and the operators $+$, $\cdot$, and $*$.

Definition 10

We construct *regular expressions* by applying the following rules:

1. $\varnothing$, $\Lambda$, and $a \in \Sigma$ are all (primitive) regular expressions.
2. If $r_1$ and $r_2$ are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, $r_1^*$, and $(r_1)$.
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in step 2.

The next definition describes the languages that can be represented by regular expressions.

Definition 11

The *language $L(r)$ denoted by any regular expression $r$* is defined by the following rules: $\varnothing$ is a regular expression denoting the empty set; $\Lambda$ is a regular expression denoting $\{\Lambda\}$; for every $a \in \Sigma$, $a$ is a regular expression denoting $\{a\}$; if $r_1$ and $r_2$ are regular expressions, then so are $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, $L(r_1 \cdot r_2) = L(r_1)L(r_2)$, $L((r_1)) = L(r_1)$, and $L(r_1^*) = (L(r_1))^*$.

It is intuitively reasonable that for every regular language $L$, there exists a regular expression $r$ such that $L$ equals $L(r)$. In fact, every regular language has an associated NFA and it can be seen that the inputs of all the accepting threads from the initial state to any final state are generated by a regular expression. On the other hand, if $r$ is a regular expression then $L(r)$ is regular. This follows from the fact that $L_{\mathbf{DFA}}$ is *closed* under union, concatenation, and star closure. By closed we mean that you can take any languages in the class, apply these operations to them, and the resulting language will still be in the class. It is relatively easy to build the corresponding finite automata by working with NFAs.

### B. Grammars

We present the definition of another model of computation, namely a grammar, in this section.

Definition 12

A *grammar* $G$ is defined as a quadruple $G = (N, \Sigma, S, P)$, where

- $N$ is an alphabet called the set of *nonterminals.*
- $\Sigma$ is an alphabet called the set of *terminals,* with $\Sigma \cap N = \varnothing$.
- $S \in N$ is the *start variable.*
- $P$ is a finite set of *productions* of the form $x \to y$, where $x \in (N \cup \Sigma)^+$ and $y \in (N \cup \Sigma)^*$.

Given three strings $w, u, v \in (N \cup \Sigma)^*$ such that $w = uxv$, we say that the production $x \to y$ is applicable to $w$. By applying the production to $w$ we obtain a new string $z = uyv$. We say that $w$ derives $z$ and denote this with $w \Rightarrow z$. If $w_1 \to w_2 \Rightarrow \cdots \Rightarrow w_n$, we say $w_1$ derives $w_n$ and denote this with $w_1 \Rightarrow^* w_n$. The set $L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$ is the language generated by $G$.

A third way of representing regular languages is by means of certain simple grammars, called *regular grammars.*

**Definition 13**

A *regular grammar* is a four-tuple $G = (N, \Sigma, P, S)$, where $N$ is a nonempty set of *nonterminals,* $\Sigma$ is an alphabet of *terminals,* $S \in N$ is the *start symbol,* and $P$ is a set of *productions* of the form $x \rightarrow y$, where $x \in N$ and $y \in \Sigma^* N \cup \Sigma^*$.

Notice that productions in a regular grammar have at most one nonterminal on the right-hand side and that this nonterminal always occurs at the end of the production. The grammar *generates* a string of terminals starting from $S$ and then by repeatedly applying productions in the obvious way until no nonterminals remain.

Let $M = (Q, \Sigma, \Delta, q_0, F)$ be an NFA accepting a language $L$. It is easy to see that a regular grammar $G = (N, \Sigma, S, P)$ generating $L$ can be defined by setting $V = Q$, $S = q_0$, and putting in $P$ the production $q_i \rightarrow a_j q_k$ if $(q_i, a_j, q_k) \in \Delta$ and the production $q_k \rightarrow \Lambda$ if $q_k \in F$.

It is also easy to see that every regular grammar $G = (N, \Sigma, P, S)$ has a corresponding NFA $M = (N \cup \{f\}, \Sigma, \Delta, S, \{f\})$, where $\Delta$ is formed as follows:

1. For every production of the form $A \rightarrow xB$, where $A, B \in N$ and $x \in \Sigma^*$, the transition relation $\Delta$ contains $(A,x,B)$.
2. For every production of the form $A \rightarrow x$, where $A \in N$ and $x \in \Sigma^*$, the transition relation $\Delta$ contains $(A,x,f)$.

Whenever we define a language family through a grammar, we are interested in knowing what kind of automaton we can associate with the family. This will give us an indication of how efficiently the language can be recognized. Programming language syntax is usually specified using a grammar. The parser for the language is an implementation of the corresponding automaton.

## VII. PUSHDOWN AUTOMATA AND BEYOND

### A. Introduction

Regular languages have broad application in computer science, but many useful languages are not regular and so cannot be accepted by DFAs and NFAs. For example, no DFA can accept the language consisting of nested, balanced parentheses, i.e. $\{(^i)^i \mid i \geq 0\} = \{\Lambda, (), (()), ((())), \ldots\}$. Obviously, a language such as this is an important one from the perspective of programming languages. The reason DFAs cannot accept a language like the nested, balanced parentheses language is because they do not have any way of storing information other than in a finite set of states.

The *deterministic pushdown automaton* (DPDA) and the *nondeterministic pushdown automaton* (PDA) considered in this section are extensions of the DFA and NFA, respectively. The models are extended by adding a *pushdown* (or *stack*) data structure. Stacks provide the machines with the ability to write and store information for later retrieval. The pushdown automata allow us to accept a richer class of languages than the finite automata and are useful for parsing program code. In Section VII.D  we briefly explore an extension of these models called the *Turing machine.*

The deterministic (nondeterministic) pushdown automata can be viewed as a DFA (respectively, NFA) that has a stack added to it. Its finite control can read symbols from the stack as well as having the current state and input tape symbol to base its next move on. The next move will result in a possible state change and some manipulation of the stack. We allow the DPDA to access more than one symbol from the stack at a time analogously to NFAs.

Before formally defining the pushdown automaton let us ask the following question: how could a DFA augmented with a stack be used to recognize the language $\{(^i)^i \mid i \geq 0\}$? Intuitively, we could use the stack to store left parentheses. For each right parenthesis we encounter a left parenthesis that could be popped off the stack. If we run out of right parentheses exactly when the stack is empty, then we know that the parentheses are balanced. Naturally, we have to make sure that the parentheses are in the correct order too; we have to be careful not to accept strings like $(())()$. Let us now describe how a stack behaves, define the DPDA formally, and then return to a complete description for a DPDA that accepts the language of balanced parentheses.

We will implement the stack by adding an extra tape to our finite automaton model. This extra work tape, called the stack, will be writable (unlike the input tape). However, there will be restrictions on how this tape can be manipulated. Its initial configuration is $[1,\Lambda]$, the *empty stack.* The right end tape mark is the *bottom of stack marker,* and the stack tape head will be the *top of stack pointer.* The stack can be written to in only two circumstances.

1. The head can move left one square onto the $<$ mark and then write any symbol from the data alphabet $\Sigma$. This symbol is thus pushed onto the stack and becomes the new topmost symbol. This operation is called a *basic push.*

2. If the head is not over the > mark, then a < mark can be written and the head advanced one square to the right. This results in the top symbol being popped off the stack. This operation is called a *basic pop*.

These two basic stack operations describe how to push a single symbol and pop an individual symbol off of the stack. To implement a push $x$ operation, $|x|$ basic push operations are required. To perform no operation on the stack, a basic push operation of < is executed. To implement the instruction pop $y$, $|y|$ basic pop operations are executed; each one removing the next symbol from $y$ off the stack. To implement the pop $y$, push $x$ combination requires $|y|$ basic pops followed by $|x|$ basic pushes. Armed with these preliminaries we are now ready to define our stack machines.

## B. Deterministic PDAs

The formal definition of a deterministic pushdown automata is given below.

**Definition 14**

A deterministic pushdown automaton *(DPDA)* is a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ that is a DFA augmented with a stack. All components are defined similarly to a DFA, see Definition 1, except for the transition function that must incorporate the stack.

δ: The *transition function* or *finite control* is a (partial) function

$$\delta : Q \times \Sigma_T \times \Sigma_T^* \mapsto Q \times \Sigma_T.$$

The important points to notice about δ are that it is finite and a *partial* function. A partial function is a function that may be undefined for some elements of its domain. Although the domain of δ is infinite, δ being finite implies it is only defined for a fixed number of triples. The restrictions on how reads and writes to the stack are performed (described in the last section) disallows strings in $\Sigma_T^*$ like $a$<< that do not obey the rules for representing strings on tapes.

Let us look at a few examples of transitions in order to make the behavior of a DPDA clear. Let $q, q' \in Q$, $a \in \Sigma$, and $x,y \in \Sigma^*$. The transition $\delta(q, a, \Lambda) = (q', x)$ means read an $a$ from the input tape, change from state $q$ to $q'$, and push the string $x$ onto the stack. As described previously the push $x$ is actually implemented via $|x|$ basic push operations. The transition $\delta(q, a, y) = (q', \Lambda)$ means read an $a$ from the input tape, change from state $q$ to $q'$, and pop the string $y$ off the stack. As described earlier the pop $y$ is actually implemented via $|y|$ basic pop operations. More generally, the transition $\delta(q, a, y) = (q',x)$ means read an $a$ from the input tape, change state from $q$ to $q'$, and replace the string $y$ on the top of the stack by the string $x$—think of a pop $y$ followed by a push $x$ being executed in one step. The actual implementation of this operation requires $|y|$ basic pops followed by $|x|$ basic pushes. From now on we will focus on the high-level pushes and pops, which in the DPDA and PDA require only one step.

All of our definitions about finite automata carry over in a natural manner to pushdown automata. The one item that needs further clarification is the *acceptance of a string*. For acceptance we will require that the DPDA end up in an accepting state after reading all of its input, as we did for DFAs, but also require that the stack be empty. As promised we construct a DPDA to accept the language of nested, balanced parentheses.

**Example 3**

A DPDA to accept $\{(^i)^i \mid i \geq 0\}$.

As mentioned earlier the key idea is to store all left parentheses on the stack and then pop them off as each one matches a right parenthesis. Define $M = (\{q_0, q_1\}, \{(,)\}, \delta, q_0, \{q_0, q_1\})$, where δ is as given in Table III.

The first transition is used to push ('s on the stack; the second transition is used to match the first) with the last (; the third transition is to continue matching)'s with ('s. Let us consider some computations of $M$. We generalize the notation presented for DFAs by adding a third component to represent the stack configuration. Consider $M$ on the input string (()). We obtain the following computation:

$$\begin{aligned}(q_0, [1,(())], [1,\Lambda]) &\vdash_M (q_0, [2,(())], [1,(]) \\ &\vdash_M (q_0, [3,(())], [1,((]) \\ &\vdash_M (q_1, [4,(())], [1,(]) \\ &\vdash_M (q_1, [5,(())], [1,\Lambda])\end{aligned}$$

or

$$(q_0, [1,(())], [1,\Lambda]) \vdash_M^4 (q_1, [5,(())], [1,\Lambda]).$$

Therefore,

$$(q_0, [1,(())], [1,\Lambda]) \vdash_M^* (q_1, [5,(())], [1,\Lambda])$$

and since $q_1$ is an accepting state, all input has been read, and the stack is empty we have $(()) \in L(M)$. We have generalized $\vdash$, $\vdash^*$, and $\vdash^i$ to relate configurations of DPDAs.

Observe

$$(q_0, [1,\Lambda], [1,\Lambda]) \vdash_M^* (q_0, [1,\Lambda],[1,\Lambda])$$

**Table III** The Transition Table for the DPDA Described in Example 3

| Transition number | State | Input symbol | Stack pop | New state | Stack push |
|---|---|---|---|---|---|
| 1 | $q_0$ | ( | $\Lambda$ | $q_0$ | ( |
| 2 | $q_0$ | ) | ( | $q_1$ | $\Lambda$ |
| 3 | $q_1$ | ) | ( | $q_1$ | $\Lambda$ |

and since $q_0$ is an accepting state, $\Lambda \in L(M)$ as it should be since $\Lambda \in \{(^i)^i \mid i \geq 0\}$.

Consider $M$ on the input string ()(). We get the following computation:

$$(q_0, [1,()()], [1,\Lambda]) \vdash_M (q_0, [2,()()], [1,(])$$
$$\vdash_M (q_1, [3,()()], [1,\Lambda])$$

Notice that no further transitions of $M$ can be applied. Although $q_1$ is an accepting state and the stack is empty, because we have not read all of the input $()() \notin L(M)$. This is the correct behavior for $M$ since $()() \notin \{(^i)^i \mid i \geq 0\}$. It is not hard to verify that $L(M) = \{(^i)^i \mid i \geq 0\}$.

In the next section we define the nondeterministic version of a PDA.

## C. Nondeterministic PDA

Let us begin by presenting the definition of a nondeterministic pushdown automaton.

**Definition 15**

A nondeterministic pushdown automaton (PDA) is a five-tuple $M = (Q, \Sigma, \Delta, q_0, F)$ that is defined similarly to the DPDA except for the specification of the transitions.

The *transition relation* $\Delta$ is a finite subset of

$$Q \times \Sigma_T^* \times \Sigma_T^* \times Q \times \Sigma_T^*.$$

The main change from the DPDA to the PDA is, of course, the addition of nondeterminism. Thus from some configurations in a PDA it is possible that the machine could follow several different threads. The PDA can also perform a transition without reading input or by reading one or more input symbols. The manner in which the stack can be accessed remains the same. Other concepts defined for the DPDA can be defined analogously for the PDA. We can define corresponding language classes for PDA as we did for finite automata.

**Definition 16**

$L_{\mathbf{DPDA}}$ ($L_{\mathbf{PDA}}$) represents the class of languages accepted by DPDAs (respectively, PDAs). That is,

$$L_{\mathbf{DPDA}} = \{L \mid \text{there exists a DPDA } M \text{ with } L = L(M)\}$$

and

$$L_{\mathbf{PDA}} = \{L \mid \text{there exists a PDA } M \text{ with } L = L(M)\}.$$

## D. Inequivalence of DPDAs and PDAs

Are the language classes $L_{\mathbf{DPDA}}$ and $L_{\mathbf{PDA}}$ the same? In the case of finite automata the corresponding classes, $L_{\mathbf{DFA}}$ and $L_{\mathbf{NFA}}$, are equal. In the present case the language classes are not equal. That is, $L_{\mathbf{DPDA}} \subseteq L_{\mathbf{PDA}}$ but the reverse is not true. Let us look at an example of a language accepted by a PDA but not by any DPDA. We will only give an informal argument as to why the language is not accepted by any DPDA. However, the example should make it clear why some languages accepted by PDAs cannot be accepted by DPDA. In addition, the example will illustrate how to program a PDA.

**Example 4**

A language accepted by a PDA but not by any DPDA.

Consider the language $L = \{0^i1^i \mid i \geq 0\} \cup \{0^i \mid i \geq 0\}$. We will construct a PDA $M$ to accept $L$ and argue informally why no DPDA can accept $L$. Define $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \Delta, q_0, \{q_1, q_2\})$, where $\Delta$ is as given in Table IV.

Notice that transition 3 adds nondeterminism to the machine since this transition can be applied at any time that we are in state $q_0$. By entering state $q_1$, one can think of the machine as having guessed that the input is going to consist solely of 0's. That is, another thread is started.

**Table IV** The Transition Table for the PDA Described in Example 4

| Transition number | State | Input symbol | Stack pop | New state | Stack push |
|---|---|---|---|---|---|
| 1 | $q_0$ | 0 | $\rangle$ | $q_0$ | $0\rangle$ |
| 2 | $q_0$ | 0 | 0 | $q_0$ | 00 |
| 3 | $q_0$ | $\Lambda$ | $\Lambda$ | $q_1$ | $\Lambda$ |
| 4 | $q_0$ | 1 | 0 | $q_2$ | $\Lambda$ |
| 5 | $q_1$ | $\Lambda$ | 0 | $q_1$ | $\Lambda$ |
| 6 | $q_2$ | 1 | 0 | $q_2$ | $\Lambda$ |

Let us consider some computations of $M$. Consider $M$ on the input 000. The following computation is an accepting computation:

$$\begin{aligned}(q_0, [1{,}000],[1,\Lambda]) &\vdash_M (q_0, [2{,}000], [1,0])\\ &\vdash_M (q_0, [3{,}000], [1,00])\\ &\vdash_M (q_0, [4{,}000], [1,000])\\ &\vdash_M (q_1, [4{,}000], [1,000])\\ &\vdash_M (q_1, [4{,}000], [1,00])\\ &\vdash_M (q_1, [4{,}000], [1,0])\\ &\vdash_M (q_1, [4{,}000], [1,\Lambda])\end{aligned}$$

This computation abbreviated

$$(q_0, [1{,}000], [1,\Lambda]) \vdash_M^7 (q_1, [4{,}000], [1,\Lambda])$$

is accepting since all the input was read, $q_1$ is an accepting state, and the stack ends empty. $M$ also has rejecting computations on input 000. For example, one thread is

$$\begin{aligned}(q_0, [1{,}000], [1,\Lambda]) &\vdash_M (q_0, [2{,}000], [1,0])\\ &\vdash_M (q_0, [3{,}000], [1,00])\\ &\vdash_M (q_1, [4{,}000], [1,00])\\ &\vdash_M (q_1, [4{,}000], [1,0])\\ &\vdash_M (q_1, [4{,}000], [1,\Lambda])\end{aligned}$$

Since the input was not completely read, this is not an accepting computation even though $q_1$ is an accepting state and the stack is empty. As usual with nondeterministic models, if there is at least one accepting computation, then the string is accepted. Thus $000 \in L(M)$.

Why it is not possible to accept $L$ using a DPDA? Intuitively, this is because the two separate components of $L$ force the DPDA to make a choice. The DPDA like the DFA must always move its tape head to the right after reading a symbol on the input tape. Thus a DPDA cannot make use of the information that the end of input has been detected. Without knowing if the input has ended, a DPDA must effectively guess whether the string will be in the language $\{0\}^*$ or in the language $\{0\}^*\{1\}^*$ with a matching number of 0's and 1's. Transition 3 in $M$ is what allows us to create the necessary threads in the PDA. Since a DPDA cannot handle both of these cases, the language cannot be accepted by any DPDA. Note, if the DPDA did not need to always move the input head one square to the right, it would be able to accept $L$ as follows: push 0's until it sees a nonzero character. If the symbol is > then empty the stack while leaving the input head stationary and accept. If not, the symbol is a 1. So, match the input 1's with 0's that are on the stack. If the number of 1's and 0's matches, then accept, and otherwise reject.

**Theorem 2**

*The language classes $L_{\mathbf{DPDA}}$ and $L_{\mathbf{PDA}}$ are not equal.*

The pushdown automata are not completely general models of computation since, for example, no PDA can accept the language $\{a^ib^ic^i \mid i \geq 0\}$. We should point out that a language can be accepted by a PDA if and only if it is generated by a *context-free grammar*. These grammars have a more general form of production than the regular grammars mentioned earlier.

## E. Turing Machines

Interestingly, simply adding a read/write *work tape* to a DFA is sufficient to produce a machine, called a *Turing machine*, which is as powerful as any other computing device. We present its definition below.

**Definition 17**

A *deterministic Turing machine* (DTM) is a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ that is defined similarly to a DFA, except we allow two-way read-only access to the input tape and incorporate a work tape. The *transition function* or *finite control* is a (partial) function

$$\delta : Q \times \Sigma_T \times \Sigma_T \mapsto Q \times \{-1, 0, +1\} \times \Sigma_T \times \{-1, 0, +1\}$$

The current state of the DTM, and the current configurations of its input and work tapes, are used by δ to compute the next state and to manipulate the tapes. δ takes a state, a symbol from the input tape alphabet, and a symbol from the work tape alphabet as arguments. It generates a four-tuple that indicates the next state of the finite control, a change in position of the input tape head, a symbol to be written to the work tape head, and a change in position of the work tape head. A −1 moves left one cell, a +1 moves right one cell, and a 0 means do not move.

As with any of the automata, a Turing machine starts in its initial state with an input written on the input tape. It then goes through a sequence of steps controlled by the transition function δ. During this process, the contents of any cell on the tape may be examined and changed many times. Eventually, the whole process may terminate, either by entering a *halt state* or reaching a configuration for which δ is not defined. Languages accepted by a Turing machine are called *recursively enumerable*. A language is recursively enumerable if and only if is generated by a (general) *grammar*. Those recursively enumerable languages accepted by Turing machines that halt for any given input are called *recursive* and form a proper subset of the set of recursively enumerable languages. As with each of the other automaton, we can define the nondeterministic version of a Turing machine.

**Definition 18**

A *nondeterministic Turing machine* (NTM) is a five-tuple $M = (Q, \Sigma, \Delta, q_0, F)$ that is defined similarly to a DTM, except for the specification of the transitions. The *transition relation* $\Delta$ is a finite subset of

$$Q \times \Sigma_T \times \Sigma_T \times Q \times \{-1, 0, +1\} \times \Sigma_T \times \{-1, 0, +1\}$$

It can be shown that any nondeterministic Turing machine can be simulated by a deterministic Turing machine.

## VIII. SUMMARY

In this article we have covered the basic ideas in automata theory. The references provide additional details, and many other interesting applications and results of the theory.

## ACKNOWLEDGMENTS

The material in this article has been recast from Greenlaw and Hoover (1998). We are grateful to Jim Hoover for allowing us to include the material here. Thanks to the referees for their valuable suggestions; they helped to improve this article.

## SEE ALSO THE FOLLOWING ARTICLES

Decision Theory • Future of Information Systems • Game Theory • Information Theory • Machine Learning • Systems Science • Uncertainty

## BIBLIOGRAPHY

Greenlaw, R., and Hoover, H. J. (1998). *Fundamentals of the theory of computation.* San Francisco, CA: Morgan Kauffmann Publishers.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to automata theory, languages, and computation,* 2nd Edition, Addison-Wesley.

Lewis, H. R., and Papadimitriou, C. H. (1998). *Elements of the theory of computation,* 2nd edition. Englewood Cliffs, NJ: Prentice Hall.

Sipser, M. (1997). *Introduction to the theory of computation.* Boston, MA: PWS Publishing Company.

Sudkamp, T. A. (1991). *Languages and machines,* 2nd edition. Reading, MA: Addison-Wesley.

Wood, D. (1987). *Theory of computation.* New York: John Wiley & Sons.

# Benchmarking

**Bengt Karlöf**
*Karlöf Consulting*

## GLOSSARY

**benchlearning** Combines the efficiency aspects of benchmarking with the learning organization. It thereby combines hard and soft issues in making people learn about what is important for the success of the operation.

**benchmarking** A management method deriving from the land surveying term "benchmark" which is a point fixed in three dimensions in the bed rock. Benchmarking means calibrating your efficiency against other organizations, getting the inspiration and building on other peoples experiences.

**causality** Illustrates the cause and effect logic that is important to understand in benchmarking. Why is someone performing better and how do they do that?

**cross-industry benchmarking** Means that you take out a function or a process in a company and compare it to the corresponding organizational units in a company in another industry. Take for instance billing in the telecom industry that can be benchmarked with billing in credit card operations or energy companies.

**efficiency** The function of value and productivity. Value is in turn utility (or quality) in relation to price. Productivity means the cost of producing and delivering a unit of something. Efficiency thereby includes effectiveness and productivity.

**internal benchmarking** Surprisingly often learning does not take place across a decentralized structure with many similar units. Internal benchmarking has the purpose of creating cross learning.

**strategic benchmarking** A looser and more inspirational version of benchmarking to enhance creativity in strategy processes.

**strategy** A pattern of decisions and actions in the present to secure future success and exploit opportunities.

## I. INTRODUCTION

Benchmarking is a widely used concept, but one that is often misinterpreted and insufficiently used. Although the simple definition of benchmarking is to make a comparison between parts of or the entire operation, there is much more to it. This article sorts out the correct meaning and application of benchmarking. Based on several years of experience working with the method, this article also shares the most important lessons learned from the practical use of benchmarking. This article views benchmarking from a management point of view and does not have a specific information systems approach.

Benchmarking derives its force from the logical trap that it springs on those who oppose change. The psychology that makes benchmarking so effective compared to most other methods can be summed up as a reversal of the burden of proof. It works like this: In normal circumstances, those who want to change something have to show proof of *why* it should be changed. Application of advanced benchmarking shifts the burden of proof to the conservatives, who have to show proof of why the change should *not* be made.

## II. THEORY AND REVIEW OF BENCHMARKING

### A. What Is Benchmarking?

*Benchmark* is a term used in surveying. It means a fixed point, often marked by drilling a hole in bedrock and painting a ring around it, with a defined location in three dimensions (altitude, latitude, and longitude, not time, mass, and energy) from which the locations of other points can be determined. Etymologically the word can be traced back to the British weaving industry in the 18th century. The original meaning is uncertain, but is in any case irrelevant to the use of the word in a management context. In management, the terms *benchmark* and *benchmarking* are used as a metaphor for criteria of efficiency in the form of correct key indicators, but above all they refer to a process in an organization that leads to more efficient operation in some respect, better quality, and/or higher productivity.

The word *benchmark* has been used in functional parts of companies to denote calibrated key indicators or other comparable quantities. In the former sense, the word has been used since the 1960s for comparison of production costs in computer operation. In the advertising business it has been used in comparing quality/price relationships between comparable products.

Nothing is new under the sun. That also applies to benchmarking. The practice of acquiring knowledge from colleagues has been customary in the medical profession ever since the days of Ancient Egypt, and a similar phenomenon has been observable in Japan since the Imperial Meiji Restoration of the 1860s. The origin of benchmarking in its modern, deliberately applied form is, however, generally ascribed to Rank Xerox, which in 1978 was severely shaken by price competition from Japanese manufacturers of small and medium-sized office copiers. This competition was initially dismissed as dumping or faulty arithmetic, but then a fact-finding expedition was sent to Japan and found, to its dismay, that the Japanese had managed to reduce their production costs far below what Rank Xerox then considered feasible; their own targets for productivity improvement turned out to be hopelessly inadequate. IBM likewise adopted benchmarking a decade later, but on a scale that was far too limited; their benchmarking was between their own production units and with other mainframe computer manufacturers.

The most important aspects of current benchmarking can be summarized as follows:

1. A complete and correct description of the processes and activities that create value-adding performance
2. Correct and accepted comparison with another party—a good example
3. In-depth understanding of causality, i.e., of the differences in work organization, skills, etc. and so on, that explain differences in performance. (In short: why, why, why?)
4. Reengineering of work organization and routines and development of new skills to make operations more efficient; inspiration from, not imitation of, the partner
5. A goal-related, result-rewarded process of change that uses benchmarking as the starting point for an institutionalized search for new examples for continuity

Benchmarking is thus essentially a dynamic, improvement-oriented process of change, not a static comparison of key indicators which at worst are not calibrated and therefore not comparable. This last must be particularly emphasized, because benchmarking has often been wrongly applied using uncalibrated indicators in which nobody believes, therefore making them entirely worthless.

New methods, usually with catchy names, are constantly being launched in the field of management. The danger of a label like benchmarking is that it may be filed away in the same pigeonhole with many other concepts whose effectiveness is only a fraction of that which benchmarking possesses. So it is important for the reader to make this distinction in order to appreciate the usefulness of the method. Its potential lies perhaps not so much in the quick and easy boost to efficiency that it offers as in the institutionalized learning process, which is hereinafter termed Benchlearning. That name is a registered trademark and stands for organizational skill development.

Benchmarking can be advantageously defined in terms of what it is *not,* namely:

1. Key indicator analysis
2. Competition analysis
3. Imitation

It may, however, be useful to emphasize a few side effects that often tend to become the principal effects of benchmarking:

1. Definition of the content of one's own work
2. Determination of measurement parameters

3. Sound learning, which generates a demand for even more knowledge

Benchmarking is almost always initiated by the management of a company or organization or unit thereof. The aim is almost always to improve efficiency, leading to higher quality or productivity. The mistake is often made of taking certain things for granted, for example, that processes and activities are defined and that parameters exist for measuring them. Such is in fact seldom the case, and this necessitates a thorough review of the situation, which performs the highly useful function of creating awareness of what the tasks are and how they relate to the company's corporate mission.

Institutionalized learning is seldom if ever the focus of a benchmarking process. Its psychosocial consequences are thus not the original reason for starting the project. In the best case they come as a pleasant surprise—provided, of course, that all concerned have been given time to reflect, make their own discoveries, and solve their problems. The one thing that is more crucial than anything else to success in any organized enterprise is *efficiency*. This concept is therefore considered in detail next.

## B. Efficiency: Technocracy, Software, and Reality

Half in jest, I often define a technocrat as a person who approaches technical or economic problems from a strictly rational standpoint, unmoved by human, cultural, or environmental considerations.

Semantically, the term *efficiency* is an exceptionally vague one. Most people associate it with productivity in the sense of "working harder and faster." In the line of reasoning we shall follow here, productivity is just one axis of the efficiency graph. Productivity means doing things right, i.e. making or delivering products and services of a given quality at the lowest possible unit cost.

Let us begin by looking at an efficiency graph (Fig. 1), on which the following discussion is based. As the graph shows, efficiency is a function of value and productivity. This means that an organization can achieve high efficiency either by working on its productivity to lower its production costs, thereby enabling it to offer lower prices and thus greater value, or by concentrating on quality, which increases the numerator of the value fraction and offers the options of either greater volume or higher prices.

Conversely, inefficiency can take two forms. One is when the value (price/quality ratio) offered to the market is acceptable, but low productivity leads to losses when the product is sold at the price the market is willing to pay. That, until recently, was the situation of several automakers like Saab, Volvo, and Mercedes-Benz. The market was willing to pay a given price for a car of a given design, but unfortunately the cost of making the car—materials, components, and labor—was far too high.

In the automotive industry they use assembly time in hours as an approximation of productivity. An unprofitable factory with a sellable model can thus define its short- and medium-term problem as one of productivity and try to reduce the cost (or assembly time) per unit, assuming that the situation has been correctly analyzed. This brings us to one of the key questions regarding the graph, along with two others:

1. Where are we on the graph compared to our competitor or whomever we are comparing ourselves with?

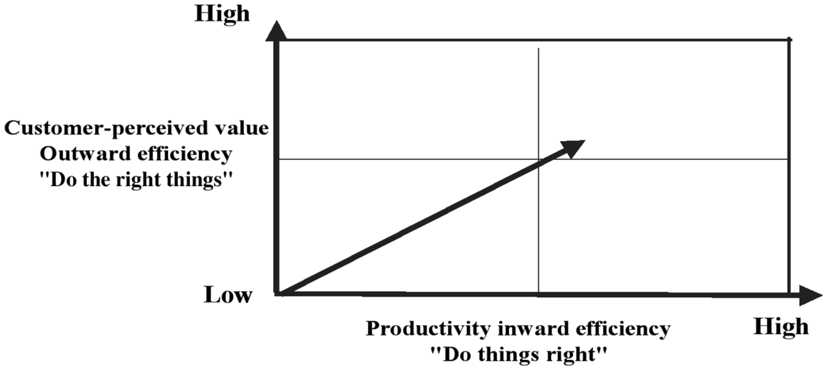


**Figure 1** The efficiency graph.

2. Given our position, in which direction should we move on the graph?
3. How quickly do we need to do it, i.e., how much time do we have?

Sometimes you can get halfway to solving the problem simply by asking the right question. If Saab, for example, has produced a model that the market is unwilling to buy in sufficient numbers at a price that is impossible to achieve, there is no point in messing about with assembly time and productivity. There are two analytical parameters, value and productivity. The foregoing example illustrates a situation in which value is judged to be satisfactory and productivity is the problem. The reverse situation, alas, is much more common and more treacherous, i.e., one in which productivity is high but the value is too low or, even worse, has never been tested on a market.

The most obvious examples of the latter situation are to be found in production of tax-financed public services. Productivity in the public sector in, for instance, my own country, Sweden, is often remarkably high. Measured in terms of processing times for inspection of restaurants or issuing of building licenses or passports, Swedish civil servants are outstandingly productive compared to their counterparts in other countries of which I have experience, such as the United States, the United Kingdom, France, or Germany. The problem, just as in companies, is that their products and services have never been put to the market test. This frequently results in production of things for which there would be no demand if they were offered on a free market where the customers could make their own choices. The situation is obscure but probably even worse in countries that also suffer from low productivity in the public sector. Many of us have noticed this since joining the European Union (EU), which has meant that yet another publicly financed system has been superimposed on those we already had.

The efficiency graph raises some central issues in a benchmarking context:

1. How do we define what we produce?
2. What is the cost per unit produced?
3. Who uses what we produce?
4. By what criteria do users judge what we produce and deliver?

Productivity in administration and production of services is a subject that has received scant attention in the literature. The question of manufacturing productivity, on the other hand, has been discussed in detail ever since the heyday of F. W. Taylor in the beginning of the 20th century. The reason, of course, is that direct material and labor costs have historically been the dominant components in the selling price of a product. John Andersson has put it this way:

> There used to be 14 blacksmiths for every clerk. Now there are 14 clerks for every blacksmith.

The sharply rising proportion of distributed costs has prompted growing interest in distributed costs in general and administrative overhead in particular. This is one reason why ABC (activity-based costing) analysis and other aids to calculation have been developed. In an era of mass production and mass consumption, productivity in manufacturing was a prime consideration. It was more important to have a car, any kind of car, than no car at all. The same applied to refrigerators, shoes, etc.

Most of what the economist Torstein Veblen has called *conspicuous consumption* takes place in modern Western societies. That term refers to values which in Abraham Maslow's hierarchy of needs are classified as self-realizing. These developments have led to a growth of interest in value theory. There is a shortage of literature, and indeed of general knowledge, about the value axis of the efficiency graph.

Value is a subjective phenomenon. Perhaps that is why analysts, regarding it as vague and lacking in structure, have paid so little attention to it. Adam Smith, the father of economic science, tried to explain value in terms of the amount of work that went into a product; but he never succeeded in explaining why a glass of water, which is so useful, is worth so little while a bag of diamonds, which is of no practical use, is worth so much.

It was the Austrian economist Hermann Heinrich Gossen (1810–1858) who came to consider how precious water becomes to a thirsty traveler in a desert. That thought led him to formulate the *theory of marginal utility,* as follows:

> The marginal utility of an article is the increment to total utility provided by the most recently acquired unit of that article.

Gossen's simple observation was that the parched traveler in the desert would be willing to trade a sizable quantity of diamonds for a liter of water, because the utility of the water far exceeded that of the diamonds.

Value theory and its corollaries regarding quality will command increasing attention from leaders of enterprises. This is especially true of leaders of units that have a planned-economy relationship to their "customers."

In most cases it is desirable to specify the value axis by constructing a value graph with quality as the abscissa and price as the ordinate, as shown in (Fig. 2).

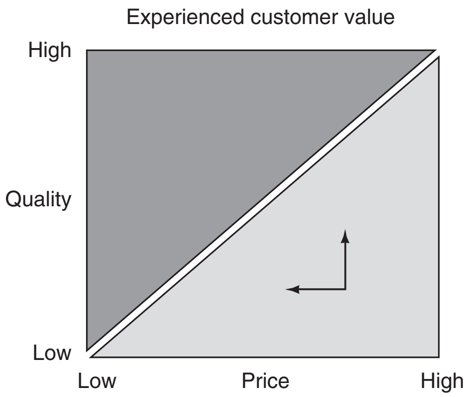


**Figure 2** The value graph.

Quality stands for all the attributes of products and services that together configure the offering. Price is the sacrifice the customer must make in order to take advantage of the offering. What Veblen calls *conspicuous consumption* is a deviation, such as when somebody buys a status-symbol car like a Jaguar at a price far in excess of reasonable payment for performance. In this case, the price functions as a signal of quality to the rest of the world. A consequence of this is that the volume of sales does not increase if the price is reduced. Conspicuous consumption situations are fairly common nowadays, occurring in connection with all prestige brands. Fake Rolex watches made in Thailand do not encroach on the market for real Rolex watches, because they are sold to people who would never dream of paying what a genuine Rolex costs.

In the case of internal deliveries between departments in the planned-economy environment that prevails within a company, the price consists of a production cost that is seldom paid directly by the receiving unit. In this case, the price does not represent a sacrifice on the buyer's part. The quality of the delivery, however, can be judged. These are very important considerations with regard to productivity measurements in planned-economy systems, as well as in total quality management (TQM) and similar schemes aimed at monitoring efficiency in units of companies and organizations.

In the application of benchmarking it is of course extremely important to get into the habit of considering both axes of the efficiency graph, i.e., value (quality in relation to price) and productivity (cost per unit). The highest price that a product or service can command is determined by the value that customers put on it. The lowest price that the supplier can charge without going broke is determined by productivity, because no company can stay in business for long if its production costs are higher than its revenues. Productivity, therefore, influences the customer's perception of value in that higher productivity makes it possible to offer a lower price and thus higher value.

Quality, on the other hand, usually costs money, when we are talking about customer-perceived quality. More space between the seats in an aircraft means fewer passengers and therefore a higher cost per passenger-kilometer. In medical services, increased consultation time per patient is likewise a quality parameter that adds to the cost of providing the service. Improving quality in the other sense, i.e., reducing the frequency of rejects in production, does not normally increase costs but reduces them instead. This is done by tightening the production flow and eliminating the cost of rework by taking proactive measures (Fig. 3).

The concept of efficiency is central not only to benchmarking, but also to all forms of enterprise and organization, so a full understanding of the meaning of efficiency is essential.

## C. Categories of Benchmarking

According to the individual situation, benchmarking can be divided into a number of categories and extremes can be contrasted with each other. Some of the more important ones are as follows:

- Strategic and operative benchmarking
- Internal and external benchmarking
- Qualitative and quantitative benchmarking
- Same industry and cross-industry benchmarking
- Benchmarking in a leading or supporting role
- Benchmarking for better performance or for world-class performance

The purpose is instructional, i.e., to make the reader aware of angles that enable benchmarking to be done more effectively.

### 1. Strategic and Operative Benchmarking

One of the great advantages of benchmarking is that the method can be applied to both strategic and operative issues. The dividing line is by no means sharp. Strategy is defined here as "action taken at the present time to ensure success in the future." Strategy thus aims at achieving good results not only now but next year and the year after that too, though the term is sometimes loosely used of any issue that is of major importance, regardless of the time frame.

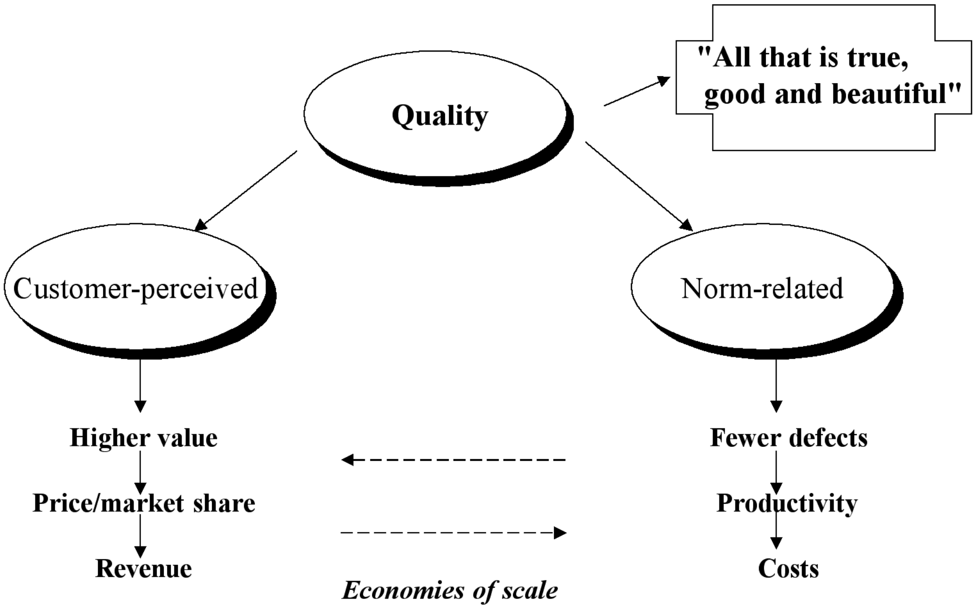


**Figure 3** Quality.

Operative management, on the other hand, has to do with all the day-to-day problems that must be solved right now to ensure that deliveries can be made to customers without a hitch. Strategy is always important, but seldom acute. Operative problems, on the other hand, may be both important and acute. (You can read more about this in *Conflicts of Leadership,* Bengt Karlöf, Wiley, 1996, and *Strategy in Reality,* Bengt Karlöf, Wiley, 1997.)

A rough distinction between strategic and operative issues is made in Fig. 4. As a strategic instrument, benchmarking can perform the function of identifying new business opportunities or seeking areas where dramatic improvements can be made. Identification of the latter is made harder in many organizations by the absence of competition, and in such cases benchmarking performs a strategic function.

Probably the most widespread application of benchmarking is as an operative instrument to identify ways to improve operations. The aim of functional or process benchmarking is to seek and find good models for operative improvements. The great challenge

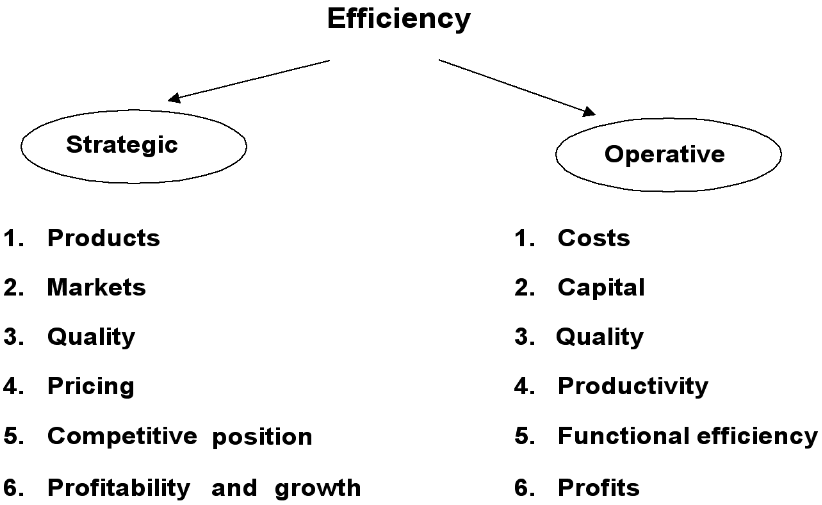


**Figure 4** Strategic and operative efficiency.

of the future will be to apply benchmarking in all parts of the business that operate under planned-economy rules. The conclusion from this is that the principal application of benchmarking will be operative. That statement is not intended in any way to belittle the importance of the strategic angle, but because competition is endemic to business, and because competition can be regarded as an ongoing form of benchmarking, one can safely conclude that all the planned-economy parts of an organization will have a strong motive to find indicators of their efficiency through benchmarking.

## 2. Internal and External Benchmarking

Benchmarking will be imperative in organizations that contain a number of similar production centers within themselves. Internal benchmarking means making comparisons between similar production units of the same organization or company. It may sometimes seem strange that organizations exposed to the full force of competition do not use benchmarking to the extent that one might expect. This applies, for example, to banks and airlines. Learning how to improve efficiency within one's own company ought to come first, indeed it should be self-evident.

External benchmarking raises numerous questions. Many people seem to assume that benchmarking must involve comparisons with competitors, but this is not necessarily so. Of all the projects I have worked on, only a vanishingly small minority have involved competitors. Where this has happened, the subjects of study have been processes at an early stage in the value-added chain, like production and project engineering, which are not regarded as competitively critical.

Almost all industries offer opportunities for benchmarking in which the competitive situation is not a sensitive issue. A construction firm in England, for example, may pick one in North America with which it is not in competition. A European airline may benchmark its medium-haul operations against an airline in Southeast Asia, where the degree of competition is negligible.

A step-by-step procedure often follows the sequence of

- Internal comparisons
- Benchmarking within the same industry
- Good examples outside the industry

I would like to emphasize that companies should start by picking partners whose operations are as closely comparable as possible to avoid straining people's ability to recognize similarities. Later you can gradually move further afield to study situations that may differ more from your own, but where even greater opportunities for improvement may be found. In many cases there is a tendency to regard internal benchmarking as simpler. This is true in some respects, but not in others. The instinct to defend one's territory and not give anything away can be an obstacle to contacts with colleagues in other parts of the organization.

A combination of internal and external benchmarking has frequently proved fruitful. This means seeking both internal and external reference points simultaneously and thus opening a wider field to the participants, giving them insights into situations other than those with which they are already familiar. Variety is valuable in itself in enhancing the instructiveness of benchmarking; the participants are stimulated and inspired by glimpses of working environments that differ somewhat from their own.

## 3. Qualitative and Quantitative Benchmarking

The object of any organized activity is to create a value that is greater that the cost of producing it. This applies to staff work just as much as to selling and production, and it applies to all kinds of organizations regardless of mission or ownership. In some cases quality and productivity may be exceedingly difficult to measure. The personnel department of a company, regardless of what that company does, is likewise expected to produce a value greater than the cost of producing it, but in such a department it may be far from easy to answer the questions that touch on efficiency:

1. What do we produce and deliver?
2. What does it cost per unit?
3. Who evaluates what we deliver?
4. On what criteria is the evaluation based?

A personnel department normally exists in a planned-economy environment, operating on money allocated by top management and lacking customers who have a free choice of supplier and pay for what they get out of their own pockets. This makes the questions hard to answer, but does not make them any the less relevant. The processes of personnel or human resource management are fairly easy to define and structure; the hard part lies in defining who the buyers of the department's services are and what set of criteria should be used to dimension the resources allocated to the department.

Qualitative benchmarking may be preferable in such cases, i.e., a description of processes and activities followed by a comparison with the way similar things are done in another organization that serves as a good example. This is usually called descriptive or qualitative benchmarking, and in many cases it is quite as effective as the quantitatively oriented kind. Qualitative benchmarking can often be supplemented by measurement of attitudes, frequency studies of typical cases, and isolation of customized production. The latter includes the making of studies and other work of a nonrepetitive nature.

## 4. Same Industry or Same Operation?

Some of the most important questions here have already been dealt with under the heading of internal and external benchmarking. One client in the telecom business said: "The telecom industry is tarred with the monopoly brush all over the world, so we must look for examples outside the industry." The same is true of many European companies and other structures that are largely shielded from competition. In such cases it is advisable to seek examples from other industries. Experience shows that it is this type of exercise that reveals the widest performance gaps, and thus the greatest opportunities for improvement. You can however look around in your own industry if good examples are available at a sufficient geographical remove that the awkward question of competition does not arise. European airlines can go to the United States, and firms in the building trade can undoubtedly find useful objects of comparison all over the world.

Some examples of cross-industry benchmarking are shown in Table I. They include some well-known and illustrative examples of how inspiration can come from quite unexpected places. The recognition factor—how easy it is to relate what you find to your own experience—determines how far you can safely move outside your own familiar field of business.

## 5. Benchmarking in a Leading or Supporting Role

Benchmarking can be run as an adjunct to other methods and processes of change or the methods and processes can be run as adjuncts to benchmarking. In a number of cases, business process reengineering (BPR) has been clarified and made more manageable by the addition of benchmarking. In other cases process definitions and measurement criteria may be the object of benchmarking. The same applies to Total Quality Management (TQM), just-in-time (JIT), *kaizen* (continual improvement), lean production, and other approaches. The educational aspect of benchmarking helps to identify both the need for change and opportunities for improvement. The ranking order should be determined by the kind of change desired. If uncertainty prevails on that point, running an exploratory benchmarking exercise can be useful. That is usually an excellent way to find out *what* needs to be changed.

If you go on from there to consider *how* and *why* something needs to be changed, benchmarking can provide a source of inspiration as part of a broader program of change. It may also be used as the principal instrument for improvement, preferably in conjunction with the learning that takes place when benchmarking is further developed into Benchlearning. That method takes full advantage of the institutional learning process that benchmarking can generate. This approach is particularly suitable where the object is to combine business development with skill development in parts of companies.

It is, of course, impossible to predict all the situations of change that may arise, but it is always advisable to try benchmarking as a powerful educational accessory to other schemes in cases where it is not the principal instrument of change.

**Table I** Examples of Cross-Industry Benchmarking

| Global best practices: Take a look outside your own industry | |
|---|---|
| **Key process** | **Good examples** |
| Defining customer needs and customer satisfaction | Toyota (Lexus), British Airways, American Express |
| Dealing with customers' orders | DHL, Microsoft |
| Delivery service | UPS (United Parcel Service), Electrolux, Atlas Copco |
| Invoicing and debt collection | American Express, Singapore Telecom |
| Customer support | Microsoft, Word Perfect |

### 6. Benchmarking for Better Performance or for World-Class Performance

Sometimes an organization must strive for world-class performance to secure its own existence and survival—as for example in the case of nuclear power station builders, international accountancy chains, or credit card operators. The rule of thumb is that the more concentrated an industry is, the more important it is to achieve world-class performance in the organization as a whole and all of its parts. There is, however, an unfortunate tendency to strive for peak performance without regard to the company's frame of reference or the competition it faces. The aim should be to seek an optimum rather than a maximum solution, i.e., to gradually realize the potential for improvement that is achievable through effort and determination.

To some extent it may be useful to discuss the world-class aspect, especially in Europe where pressure of competition is low. Many organizations suffer from an odd combination of overconfidence and an institutional inferiority complex. One often encounters the attitude that our public service, despite lack of competition, is of course the best of its kind in the world and that there is nothing else that can be compared to it. The primary weakness of this line of reasoning is that it is probably not true, and a secondary one is that it assumes there is nothing to be learned anywhere, which is never true.

The discussion, if we can manage to conduct it unencumbered by considerations of prestige, often leads to the opposite conclusion by revealing a number of imperfections that not only indicate potential areas of improvement but also betray a sense of inferiority in important respects. Low pressure of competition encourages an overconfident attitude that is often only a thin veneer in the performance-oriented world of today.

Willingness to learn is obviously a good thing. Such an attitude is characteristic of successful organizations; the trouble is that the very fact of long-term success makes it very difficult to maintain.

The literature on benchmarking generally recommends world-class performance as the aim. That is all very fine if you can get there, but the road may be a long one. If the shining goal of the good example is too far off, the excuse-making reflex sets in and presents a serious obstacle to the process of change. The ideal is to be able to find a few partners at different removes from your own organization; this maximizes the force for change, enabling good practice to be studied in various organizations and to serve as a source of inspiration for improving your own. Figure 5 illustrates how the creativity of one's own organization can translate inspiration from a partner into performance that surpasses the latter's. In the ideal situation, this leads to a self-sustaining process of improvement, Benchlearning.

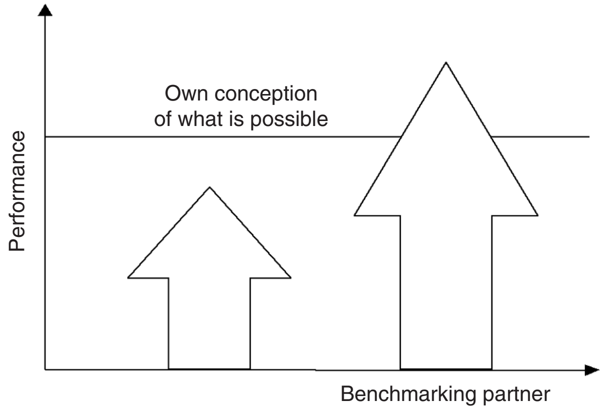


**Figure 5** Inspiration from a benchmarking partner. [From Karlöf, B., and Östblom, S. (1993). *Benchmarking.* Chichester: Wiley.]

When you set out to organize and mount a benchmarking project, you should consider a number of questions that may give rise to problems at the implementation stage, e.g. between consultant and client:

1. Who does the necessary research?
2. Who is the customer?
3. How many people are involved and in what capacities?
4. Who should be included in the project group?
5. What can be classed as a "good example"?
6. What experts need to be consulted in what areas (e.g., ABC analysis)?
7. What special analyses need to be made (e.g., frequency studies and time measurements)?
8. How much money is available for the project?
9. What kind of skills and experience are needed on the project team?
10. Organize a seminar on success factors and pitfalls!

If you pay close attention to these questions and answer them properly, you will minimize the risk of running into difficulties along the road.

## III. METHODOLOGY OF BENCHMARKING

After some 10 years of learning lessons from benchmarking, I have now adopted a 10-step methodology.

How the process is divided up is actually just a teaching device. An experienced benchmarker can make do with fewer steps because he or she is familiar with what is included in each of them.

## 1. Explanation of Method, Leadership, and Participation

People with extensive experience in benchmarking tend to seriously underestimate how difficult and time consuming it is to explain the method and get people to understand it. The natural way to structure the explanation is to use the lessons related here and the structure of the method shown below:

1. A detailed explanation of the benchmarking method with time for questions and discussion
2. The concept of efficiency as a basis for diagnosis, measurement, and comparison
3. Definitions of concepts, terminology, and methods that are recognized by all members of the group
4. A survey of any special circumstances that may motivate a departure from the 10 steps presented here

## 2. Choosing What to Benchmark

The natural place to start is with units whose efficiency is known to be below par. The units concerned may have administrative functions like finance or information technology, but they can just as easily be line functions like aircraft maintenance or luggage handling. In the telecommunications field it may be invoicing or productivity in access networks. In insurance companies it may be claims adjustment, in banks creditworthiness assessment, and so on. In every industry there are production units of a generic nature like IT as well as industry-specific processes like claims adjustment (insurance), clinical testing (pharmaceuticals), etc. Generic support processes can often be compared across industrial demarcation lines, whereas more specific processes call for a closer degree of kinship. To sum up, we are looking at a situation in which efficiency is clearly in need of improvement.

## 3. Business Definition and Diagnosis

A considerable proportion of the time devoted to benchmarking is spent in defining the work content of the business, or in what I have called *diagnosis.* Put brutally, the statement may sound like an extremely harsh judgment. What we are actually saying is that people have failed to define what they are doing and that they therefore do not know their jobs. Sadly, that brutal statement is corroborated by repeated and unanimous experience. Just as learning is such an important side effect of benchmarking that it contends for the title of principal effect, business definition or diagnosis is so important that it deserves to be highlighted as the special value that benchmarking often creates, even though that was not the original intention.

## 4. Choosing Partners

Choosing a partner is naturally an essential feature of benchmarking. In a structure with multiple branches, the choice might seem to be obvious—the partners are right there in your own organization. Experience shows, unfortunately, that even if the identity of the partners is obvious, the difficulties involved in making contact and collaborating are actually greater than with an external partner. As a result of decentralization, often carried to extremes, there are a lot of people in an organization who regard definition as their own prerogative and therefore see no reason to harmonize process descriptions, measurement criteria, etc. So the partnership contact that looks so easy may turn out to be fraught with great difficulty.

## 5. Information Gathering and Analysis

We are now ready to proceed to one of the core areas of benchmarking: the collection of information and the analysis of descriptive and numerical quantities. The descriptive quantities are things like work organization, skill development, and other areas relevant to the comparison that cannot be expressed in figures.

## 6. Inspiration from Cause and Effect

Benchmarking indicates not only *what* can be improved, but also *how* and *why*. This is what we call a cause-and-effect relationship, causality or *why, why, why?* Even in studies that use calibrated key indicators, the important explanations of differences in performance are often overlooked. You do not have to go down to the level of detail of describing every manual operation, but you should realize that if the drivers in your partner company take care of routine checks that have to be made by skilled technicians in your own company, that goes a long way toward explaining differences in performance.

### 7. Follow-Up Visit

The follow-up visit qualifies as a separate step because it needs to be taken into account at the planning stage and in initial discussions with partners. If you fail to schedule at least one follow-up visit as part of the program, your partner may feel neglected. One or more follow-up visits are a perfectly normal feature of benchmarking.

The main items on the agenda of the follow-up visit are

1. Checking figures and measurements
2. Testing hypotheses to explain gaps
3. Identifying new explanations of why differences in performance exist
4. Discussing obstacles to improvement

### 8. Reengineering

The word *reengineering* is used here to denote changes in processes or flows, activities, work organization, system structures, and division of responsibilities. Benchmarking may be supplemented by other points of reference. Two such points of reference may be well worth considering in some cases because of their great instructive value:

1. Own history (experience curve)
2. Zero-base analysis

### 9. Plans of Action and Presentation

The reason why these two apparently disparate items have been lumped together under one head is that they are interdependent. The substance of the plan of action influences the form in which it is presented and vice versa. Planning the changes prompted by a benchmarking study is no different from any other kind of project work. The benchmarking element, however, simplifies the job of planning as well as the presentation, and the changes will be easier to implement and better supported by facts than in projects of other kinds.

The plan of action covers the following principal parameters:

1. Strategies and goals
2. Studies
3. Activities
4. Responsibility
5. Time
6. Resources
7. Results

### 10. Change and Learning

Change management is much more difficult and demands much more energy than is commonly supposed at the outset. Even when the message has been received, understood, and acknowledged, a tremendous amount of work still remains to be done.

## IV. BENCHMARKING—PITFALLS, SPRINGBOARDS, AND OBSERVATIONS

Leaders of business know by experience that good examples, well chosen and presented, have great educational value. They have also learned to discount the value of key indicators that nobody believes in and that only provoke the excuse-making reflex in the organizations and individuals concerned. In addition, they understand the logic of shifting the burden of proof: Nobody wants to make a fool of himself by rejecting changes when there is hard evidence to show that others have already done so and that the results have been successful. Anybody who did reject them would be insulting her own intelligence.

There are, however, innumerable kinds of less blatant behavior that can detract from the effect, but there are also ways of finding shortcuts that lead much more quickly to the goal of improvement. What follows is a list—not in any particular order, but based on experience—of some of the pitfalls, springboards, and observations.

## A. Pitfalls of Benchmarking

### 1. The Effect of Benchmarking Is Binary

Thoroughness in calibrating numerical data and consistency in the descriptive parts of the analysis are essential. The requirements here are full comparability, elimination of noncomparable factors, and acknowledgment by the people concerned that the highest possible degree of comparability has in fact been achieved. If those people go through the same intellectual process that you have gone through and accept the comparability of the findings, that effectively disarms the excuse-making reflex. If, in addition, you have been really thorough about gathering data and can show that you have considered and measured all of the relevant factors, you will avoid the yawning pitfall you can otherwise fall into—the pitfall of lack of calibration, comparability, and acceptance.

## 2. Beware of Distributed Costs

There used to be 14 smiths for every clerk. Now there are 14 clerks for every smith. With IT as an enabler, with a shrinking proportion of direct costs and a growing proportion of overheads, it has become increasingly hard to assign cost elements to a specific operation. This difficulty can be overcome by using either ABC analysis or indicators that exclude overheads to secure acceptance of cost distribution and comparability. I have had personal experience of situations where certain cost elements were charged to a higher level of management in one unit than in another. Much to our embarrassment, this was pointed out to us when we made our presentation, and we had to go back and revise some of the material.

## 3. Do Not Save Everything for the Final Report

One way to avoid the embarrassment of being caught out in errors of comparability is to present conclusions in the form of a preliminary report that is expected to be adjusted prior to the final presentation. This defuses the emotional charge in the analysis and enables a kinder view to be taken of any errors.

## 4. Do Not Be a Copycat

A lot of people with no firsthand experience of benchmarking think it means copying the behavior of others. That is, of course, true to the extent that good practice is well worth following. But the real aim of benchmarking is *inspiration* for your own creativity, not *imitation* of somebody else. Benchmarking is intended to promote creativity, which can be defined as the ability to integrate existing elements of knowledge in new and innovative ways. What benchmarking does is to supply the necessary elements of knowledge.

## 5. Do Not Confuse Benchmarking with Key Indicators

Let me recapitulate the levels of benchmarking:

1. Uncalibrated key indicators that nobody believes in and that therefore have no power to change anything
2. Calibrated key indicators unsupported by understanding of the differences in practice and motivation that explain differences in performance (*why* and *how*)
3. Calibrated key indicators supported by understanding of *why* and *how*. (This is real benchmarking, and it leads to spectacular improvements)
4. Learning added to benchmarking, which then evolves into Benchlearning

The greatest danger to the future destiny and adventure of benchmarking is that it may degenerate into bean-counting with little or no calibration. So let us be quite clear about what benchmarking really means.

## 6. Complacency Is a Dangerous Enemy

The attitude that “we know it all, know best and can do it ourselves” is a cultural foundation stone of many organizations that reject benchmarking as an instrument for their own improvement. Such organizations react vehemently with the excuse-making reflex when benchmarking is applied to them. The danger of complacency is greatest at the outset. When benchmarking is introduced in an organization with a culture of complacency, it is often dismissed with scorn.

Many successful organizations have applied benchmarking too narrowly and consequently have run into structural crises. Rank Xerox, where the benchmarking method originated, had set productivity improvement targets that proved hopelessly inadequate. IBM benchmarked between its own units, but not to any significant degree outside its own organization, being overconfident in its own superiority. Uninterrupted success is an obstacle to the learning and improvement of efficiency that benchmarking can offer.

## 7. Benchmarking Is Not the Same Thing as Competition Analysis

I have concerned myself very little with what is sometimes called *competitive benchmarking*. This is because benchmarking is not at all the same thing as competition analysis, which traditionally consists of charting

1. Market shares
2. Financial strength
3. Relative cost position
4. Customer-perceived quality

These are highly interesting items of information, but they do not constitute benchmarking. If benchmarking is done with a competitor as the partner, it must be done in areas where the competitive interface is small. Some industries, the automotive industry for example, practice what they call *co-opetition*—a mixture of cooperation and competition. The farther back you go along the integration chain, the easier it is to collaborate. Conversely, it is very hard to collaborate at the front end where you are directly competing for customers.

### 8. Do Not Let Your Own Administrators Take Charge of New Methods

In the 1980s companies set up quality departments. These have gradually expanded their domains to include productivity aspects like BPR and, of course, benchmarking. Having projects run by your own staffers may look like a cheap way of doing it, but is seldom effective. In fact, I have never seen an in-house benchmarking project that was really successful. The resources allocated are inadequate, the project is a secondary assignment for the people in charge of it, and the method is not fully understood. It is the line management that must be motivated and act as the driving force, and it is desirable to retain outside consultants for three reasons:

1. The project must be somebody's primary responsibility, which is never the case when the project manager is recruited internally
2. Benchmarking calls for specialized knowledge that few employees possess
3. People from inside the company are suspected of having their own corporate political agenda, which reduces their credibility

So make sure that somebody has the primary responsibility. That probably means bringing in an outside consultant.

### 9. Benchmarking Risks Being Viewed as a Management Fad

The fact that the whole field of management is a semantic mess is aggravated by the confusion caused by all the new methods that burst on the scene from time to time, only to fade away again. We once ran a poll asking a large number of respondents to draw a life-cycle curve for benchmarking as a method. This was done in relation to other approaches like BPR, TQM, lean production, reinvention of the corporation, JIT, conjoint analysis, and so on. Though the interviewees had no special preference for benchmarking, their verdict was that it would stand the test of time. Figure 6 shows the predicted life-cycle curve for benchmarking as a management technique compared to those of other unnamed methods. This claim must of course be made in an article about benchmarking, but there is much independent evidence that testifies to the staying power of the method.

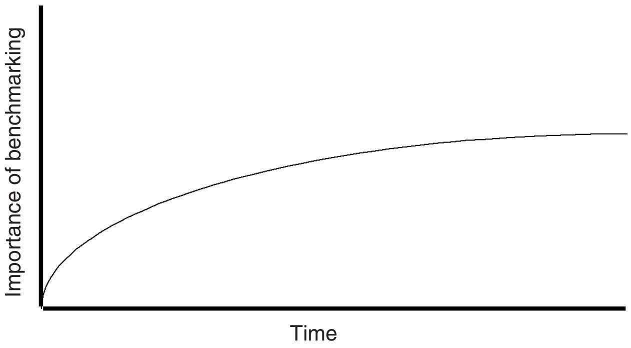


**Figure 6** Life cycle of benchmarking.

### 10. Our Banana Is the Best in the Bunch

All organizations prefer to make their own inventions rather than give the credit to somebody else. A European telecom company spent about $70 million on developing its own invoicing system instead of adopting an existing system from Australia that met the same specifications. The tendency to reject good examples is thus not entirely a matter of lack of comparability in the material, but may also be attributable to the "not invented here" syndrome.

## B. Springboards for Benchmarking

Springboards (success factors) for benchmarking are necessarily a mirror image of the pitfalls. Let us combine the success factors with some pertinent observations that are crucial to the success of a benchmarking project.

### 1. Get Staff Really Involved

To ensure success, the organizational units and individuals affected by the project must be given the opportunity to go through the same intellectual process as the project management. That will predispose them to accept the findings of the benchmarking study, which in turn will make it easier to take the step from thought to deed. Improvements in understanding, behavior, and results will all be accelerated if people are allowed to take an active part. So allocate a share of the resources to getting them really involved.

## 2. Make Special Measurements to Acquire New Information

There is never enough information in existing reporting systems. Efforts to improve efficiency nowadays are directed mainly at administrative work, and this calls for time management studies, floor traffic studies, descriptions of sales and other processes, and so on. Special measurements have double value in that they both get people involved and provide new information that is often interesting and helpful for purposes of change management. You may sometimes experience problems in persuading your partner to make the necessary measurements, but this is seldom an insurmountable obstacle. If you start with the two parameters of efficiency—quality and productivity—you can usually find innovative approaches, methods, and measurements that make a significant contribution to the project.

## 3. Benchmarking Contributes to Both Gradual and Quantum-Jump Change

The magnitude of the improvements that can ultimately be achieved is impossible to judge in advance. Experience shows, however, that although you can sometimes find hitherto unsuspected possibilities for making major breakthroughs, you will usually have to be patient and take many small steps that will ultimately lead to great gains in efficiency. Radical change management is often prompted by a crisis situation in which the very existence of the business is threatened. Benchmarking, on the other hand, involves a constant search for good examples to serve as models for the modest improvements that will enable you to avoid crises, i.e., to be proactive instead of reactive.

## 4. The Whole Conceals Inefficiencies in the Parts

A commercial company is evaluated by its bottom line. Other types of organizations have their own criteria for success. But if you look at the component parts of the organization—departments like accounting, personnel, and IT or processes like sales and aftermarket—you may find a grave lack of efficiency. There may thus be a substantial potential for improvement even where the overall result appears to be good. In the future, every head of department and process owner must be prepared to answer the question: “How do you know that your operation is efficient?”

Benchmarking is an unsurpassed instrument for detecting inefficiencies concealed in parts of the whole.

## 5. Benchmarking Is a Surrogate for Market Economy

A user of goods or services inside a company or organization lacks the freedom of choice between suppliers that exists in a market economy. The trouble is not only that the value of an internal delivery is not measured against a price, but also that what is delivered may not in fact be necessary at all. Units of organizations often justify their existence and growth by being productive, but the value of what they produce is not assessed. Benchmarking helps you make that assessment.

## 6. Benchmarking Encourages Performance-Oriented Behavior

Relations between individuals and units in large organizations are often, alas, governed by power rather than performance. Because the actual performance of a unit is so hard to measure, managers tend to devote more attention to matters of form and irrelevant factors than to things that really contribute to the success of the company as a whole. A person may be judged by his or her style, connections, appearance, or career status for lack of ways to measure actual performance and contribution to achieving the company’s aims.

It has become increasingly evident that benchmarking appeals to strongly performance-oriented people, by which I mean people who reckon success in terms of getting results. In this they differ from power-oriented and relation-oriented people, who are motivated by other considerations.

## 7. Proceed Step by Step

What we call the cascade approach means starting with a broad, shallow comparison (low level of resolution) and then taking a closer look at those areas that appear to offer the greatest opportunities for improvement. The first step in the cascade approach is exploratory benchmarking, a study that aims at identifying the areas with the greatest potential for improvement. Such a study seldom explains the reasons for differences in performance, but it does suggest where you should start digging deeper to look for reasons. This step-by-step approach is particularly recommended in cases where there are no obvious candidates for improvement.

## 8. Benchmarking Encourages Learning in Decentralized Systems

Organizations and companies with a multiple-branch structure and delegated profit-center responsibility often lack a mechanism for system-wide learning. One petrol station does not learn from another, nor one bank branch or retail outlet from another.

One of the obvious applications of benchmarking in the future will be to enable units of decentralized systems to learn from each other. This will take advantage of the organization's accumulated knowledge—an advantage that ought to be self-evident but is often overlooked. Failure to make use of knowledge from individual units of the organization ("advantages of skull") is astonishingly widespread.

### 9. Benchmarking Unites Theory and Practice

When the individuals in a working group or process are made to lift their gaze from their day-to-day tasks and consider their performance objectively, they form a theory about their own work. This has proved to be of great help in motivating organizations. People *like* to unite theory with practice because it makes them feel motivated. That in turn leads them to put more energy into their work, which results in a win–win situation where the employer benefits from increased energy and efficiency, while the employees are happier in their work. The function of theory is to lay a foundation for good practice. Benchmarking combines the theory of what the job in hand is really about with concrete inspiration from good examples to optimize the prospects for successful change management.

### 10. Benchmarking Benefits the Business and the Individual

Traditional training, especially in Scandinavia, focuses on the individual. People are sent off on courses to improve their qualifications. This is not always compatible with the employer's desire for better performance and efficiency. In the worst case the individual concerned may quit and use his qualifications to get a job with a competitor.

Benchmarking ensures that the business as a whole benefits from learning; knowledge becomes an asset of the organization rather than the individual. That way the skills stay in the company and the working unit even if individuals depart. This reconciles the requirements of a benevolent personnel policy with management's need for an efficiently run business.

The intellectual simplicity of the benchmarking method may be a pitfall in itself. There is a great risk of underestimating the problems. So if you are contemplating a project, use the foregoing points as a checklist to help you avoid the mistakes that others have made before you and to take advantage of the lessons they have learned.

## V. SUMMARY

Benchmarking is widely known and used, and has been for a long time. The most important elements in benchmarking as of today are as follows:

- A complete and correct description of the processes and activities that create value-adding performance
- Correct and accepted comparison with another party—a good example
- In-depth understanding of causality, that is, of the differences in work organization, skills, and so on, that explain differences in performance (In short: why, why, why?)
- Reengineering of work organization and routines and development of new skills to make operations more efficient; inspiration from, not imitation of, the partner
- A goal-related, result-rewarded process of change that uses benchmarking as the starting point for an institutionalized search for new examples for continuity

One key factor when it comes to benchmarking is efficiency, which in its right sense is a function of value and productivity. Related to this are issues of how to define what is produced, what the cost per unit is, who are the users of the product, and by what criteria those users evaluate the product.

Benchmarking can be divided into a number of categories of which some are strategic or operative benchmarking, internal or external benchmarking, and qualitative or quantitative benchmarking. Furthermore, benchmarking can be conducted within the same industry or cross-industry. Benchmarking can have a leading or supporting role and have the goal of better performance or world-class performance.

From long experience of working with benchmarking, several important and not instantly obvious issues have emerged. The method is simple and appealing, but its very simplicity has proved deceptive. Some advice to potential benchmarkers is to beware of distributed costs, to get inspired instead of copying, and not to confuse benchmarking with comparison of uncalibrated key indicators or competition analysis.

Benchmarking can be further developed into the method of Benchlearning, which also incorporates a continuous learning process in the organization with numerous positive side effects.

## SEE ALSO THE FOLLOWING ARTICLES

Cost/Benefit Analysis • Executive Information Systems • Goal Programming • Operations Management • Project Management Techniques • Quality Information Systems • Reengineering • Resistance to Change, Managing • Total Quality Management and Quality Control

## BIBLIOGRAPHY

Briner, W., Geddes, M., and Hastings, C. (1990). *Project leadership*. New York: Van Nostrand Reinhold.

Camp, R. C. (1989). *Benchmarking: The search for industry best practices that lead to superior performance*. Milwaukee, WI: ASQC Industry Press.

Edenfeldt Froment, M., Karlöf, B., and Lundgren, K. (2001). *Benchlearning*. Chichester: Wiley.

Gordon, G., and Pressman, I. (1990). *Quantitative decision-making for business,* 2nd ed. Upper Saddle River, NJ: Prentice-Hall International.

Karlöf, B. (1996). *Conflicts of leadership*. Chichester: Wiley.

Karlöf, B. (1997). *Strategy in reality*. Chichester: Wiley.

Karlöf, B., and Östblom, S. (1993). *Benchmarking—A signpost to excellence in quality and productivity*. Chichester: Wiley.

Peters, T. J., and Waterman, R. H., Jr. (1982). *In search of excellence: Lessons from America's best run companies*. New York: Harper Collins.

Spendolini, M. J. (1992). *The benchmarking book*. Amacom.

Thompson, A. A., Jr., and Strickland, A. J., III (1990). *Strategic management—Concepts and cases,* 5th ed. Homewood, IL: Irwin.

# Business-to-Business Electronic Commerce

**Jae Kyu Lee**
*Korea Advanced Institute of Science and Technology*

## GLOSSARY

**agent-based commerce** Electronic commerce that is assisted by the buyer agents and seller agents according to the mutually agreed language, message format, and protocol.

**B2B EC** Electronic commerce between businesses.

**B2C EC** Electronic commerce between business sellers and individual consumers.

**data warehouse** Data repository that stores the historical transactions to mine the meaning patterns useful for decision making.

**e-hub** The electronic hub which plays the roles (at least one of the roles) of e-marketplace, supply chain information coordinator, application service providers to enable the B2B EC.

**e-marketplace** Electronic marketplaces that display the electronic catalogs of suppliers, take orders, support the order fulfillment and perhaps the electronic payments.

**e-procurement** Electronic procurement system for buyer company, which needs to integrate with the external e-marketplaces as well as the buyer's corporate information system such as ERP.

**electronic supply chain management** Management of the supply chain on-line to reduce the inventory to minimum by sharing the information between partnered buyers and sellers.

**exchange** A type of e-marketplace where many buyers trade with many sellers.

**internet-based EDI** Electronic data interchange that is implemented on the Internet in a secure and economical manner.

**virtual corporation** An organization composed of several business partners, sharing costs and resources to produce goods and services more effectively and efficiently.

**XML** eXtensible Markup Language that can be browsed to human and also comprehensible by software agents.

**BUSINESS-TO-BUSINESS ELECTRONIC COMMERCE** (B2B EC) has emerged as the largest pie in EC. However, its definitions as well as perspectives on the current status and future prospects are not unequivocal. Therefore this article describes the necessary factors that can enhance the effectiveness and efficiency of the electronic commerce between companies. Then the current key activities in the B2B EC area are reviewed one by one with the discussion of its relationship with others. Key activities included are B2B e-marketplaces (seller, intermediary, or buyercentric ones); on-line service applications for business; the electronic procurement management system and its integration with external e-marketplaces; evolution of the EDI platform onto the Internet; collaboration by supply chain management; the shared data warehouse and virtual corporation; agent-based commerce between the buyer agents and seller agents; XML (eXtensible Markup Language) standards and their integration with electronic data interchange (EDI) and agents. This article provides a good understanding about activities in the B2B EC area, and explicates the converging phenomena among the separate activities toward enhanced managerial performance.

## I. INTRODUCTION

Electronic Commerce (EC) has been launched with the web-based e-marketplaces for individual consumers. As the large portion of buyers become businesses, it has become necessary to contrast business-to-business (B2B) EC with business-to-consumer (B2C) EC. According to the forecasts, B2B EC is expected to grow to $1330.9 billion by the year 2003, and continues to be the major share of the EC market. Since B2B EC has evolved from B2C EC, there are common points between them, but the distinction has become enlarged as the technologies for B2B EC develop.

Theoretically, B2B EC pursues the maximized values in terms of quality and delivery time, efficient collaboration, and reduced search and transaction costs. According to Handfield and Nichols (1999), full-blown B2B applications will be able to offer enterprises access to the following sorts of information about associated companies:

- *Product:* Specifications and prices in e-catalogs, and sales history in the data warehouse
- *Customer, sales, and marketing:* Sales history, forecasts, and promotion for one-to-one customer relationship management (CRM)
- *Suppliers and supply chain:* Key contractors, product line and lead times, quality, performance, inventory, and scheduling for effective supply chain management
- *Production process:* Capacities, commitments, and product plans for virtual corporations
- *Transportation:* Carriers, lead times, and costs for just-in-time (JIT) delivery management
- *Competitors:* Benchmarking, competitive product offerings, and market share

In practice, key activities in the B2B EC community are

1. The B2B e-marketplaces emerged with the same architecture used for B2C, although they handle different contents that business requires. Dell and Cisco are good examples of this category of direct marketing.
2. A large number of *vertical e-marketplaces* (usually called *exchanges* because in most cases there are many buyers and sellers involved) are widely established. In the B2B e-marketplaces, the buyers play the key role for market making, while in the B2C e-marketplaces sellers control the e-marketplaces.
3. Large buyers build their own e-marketplaces to handle their tenders in their servers, and to integrate the procurement process with their virtually *internalized e-marketplaces.*
4. *E-procurement systems,* which support the internal procurement management and its integration with the external e-marketplaces, begin to emerge.
5. The platform for *EDI* moves onto the Internet to reduce its implementation cost. So the secure extranet has become popular among associated companies.
6. To reduce the inventory and capacity, the information is shared among aligned partners. To support such collaboration, the third-party *electronic supply chain management (SCM) hub* service appeared. Big retailers opened their own data warehouse to their suppliers as a private e-hub on the Internet.
7. The third-party JIT delivery service becomes tightly coupled with suppliers.
8. To reduce the transactional burden between buyers and sellers, the software agents are actively under development. These agents will open the next stage of agent-based commerce.
9. Sellers can trace the behaviors of on-line customers precisely, and store this information in the data warehouse. So web mining through the data warehouse for customer relationship management has become popular for identifying relevant customers.

To grasp the activities of, and to design the B2B EC, the entities that we have to consider are selling companies, buying companies, intermediaries, e-marketplaces, e-SCM hubs, deliverers, a secure network platform on the Internet, protocol of B2B EC, and a back-end information system that should be integrated with the external e-marketplaces.

B2B EC is still in its infant stage; so many aspects have emerged from different backgrounds, as listed above. However, in the future such efforts should converge under a unified framework. This implies that there are ample research opportunities to open the future of B2B EC. The remaining sections describe each of the activities and interactions among them, and foresee future prospects.

## II. TAXONOMY OF B2B E-MARKETPLACES

The B2B EC can be best understood by categorizing who controls the e-marketplaces: the seller, intermediary, or buyer. Some e-marketplaces are called exchange. Most commonly agreed definition of exchange is the e-marketplaces which assist the matching of multiple buyers and multiple sellers. But the term may be used with different implication depending upon the purpose of authors. Let us study the characteristics of each type of e-marketplace contrasting them with the B2C e-marketplace. The best B2B web sites for 15 industries can be found in The Net-Marketing 200 (www.btobonline.com/netMarketing).

## A. Seller-Centric E-Marketplaces

The *seller-centric e-marketplaces* are the first type of e-marketplaces that can be used by both consumers and business buyers as depicted in Fig. 1. In this architecture, the features of e-catalog, e-cart, on-line ordering, and on-line payment schemes can be used for both B2B and B2C e-marketplaces. The only difference between them is their contents. While B2C handles the consumer items, B2B handles industrial items. However, some items like computers and printers are necessary in both sectors. So the seller-centric e-marketplace and direct marketing will never disappear even though the B2B EC evolves.

The seller-centric e-marketplace is the easiest way to build B2B e-marketplaces. Successful cases of this kind are manufacturers like Dell and Cisco, and retailers like Amazon and Ingram Micro. All of them support e-catalog, and some support auction services. For instance, Ingram Micro (www.ingram.com) opened the auction site only to the existing customers to sell off the seller's surplus goods at a deep discount. The e-marketplace may consist of pure on-line players or a combination of on- and off-line (click-and-mortar) businesses. In the early stage of seller-centric e-marketplaces, there is no distinction between the architectures of B2B and B2C EC.

It is reported that Dell sold more than 50% of computer products on-line, and 90% to business buyers. Cisco sold more than $1 billion worth of routers, switches, and other network interconnection devices to business customers through the Internet. The Internet covers 90% of sales and 80% of customer support, and the operating cost is reduced by 17.5%. In this way, Cisco recorded the gross margin of 64.4% in 2000. The seller-centric architecture will continue to exist as long as the supplier has a superb reputation in the market and a group of loyal online customers.

The benefits to online sellers are

- Global single contact point
- Updating the e-catalog is easy, on-line and consistent
- Reduced operating cost
- Enhanced technical support and customer service
- Reduced technical support staff cost
- Reduced catalog and service software distribution cost

However, the problem with the seller-centric e-marketplaces is that the buyers' order information is stored in each seller's server and it is not easy to integrate the information with the individual buyer's corporate information system. To handle this problem, the e-procurement system emerged which tries to integrate the internal procurement management

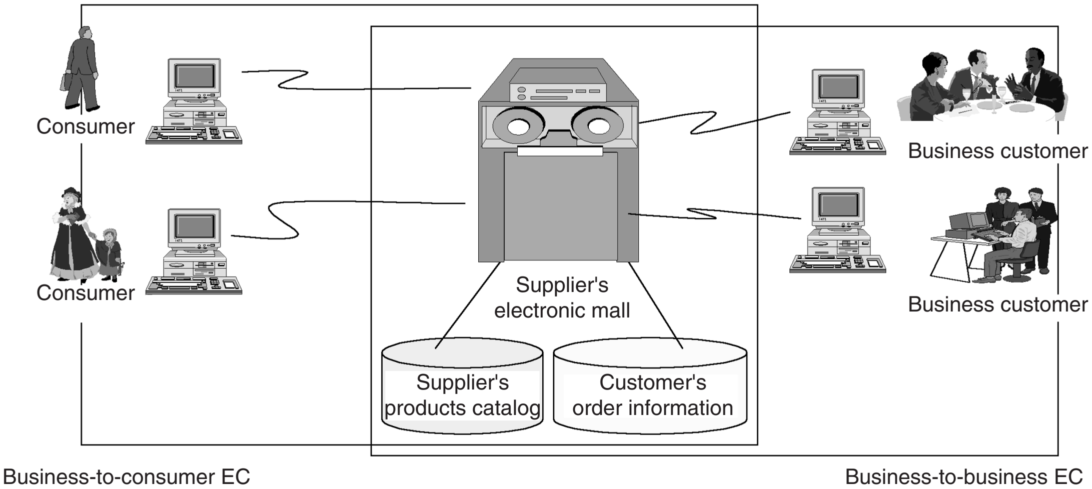


**Figure 1** Seller-centric $B_2B$ e-marketplace architecture.

process with the external e-marketplaces as explained in Section IV. When the internal information system is developed by the ERP (enterprise resource planning) system, the ERP system should be integrated with the e-marketplaces as well. For these reasons, the need of the buyer-centric e-marketplace is enlarged, as explained in Section II.C.

## B. Intermediary-Centric E-Marketplaces

The second type of e-marketplace is the *intermediary-centric e-marketplace*. The definition between the *e-tailer (electronic retailer)* and *intermediary* is blurred. A reasonable criterion of distinction seems to occur if the on-line site operator takes the orders and fulfills them. The possession of inventory cannot distinguish them precisely because many e-tailers do not keep inventory although they guarantee the order fulfillment. The intermediary-centric architecture can also be used for both B2B and B2C EC. The only difference here again is the items handled.

Examples of this kind are B2B portal directory service, comparison aid, auction to business, and exchange. A distinctive feature of the B2B intermediary e-marketplace is that in many cases buyers are involved in market making, usually to make sure that the pertinent items are appropriately included and to lead the standard of integration with their e-procurement systems. Therefore most B2B intermediaries are vertical e-marketplaces. They support e-catalog, search, auction, and exchange.

### 1. Vertical E-Marketplaces

At the moment, the active industries in *vertical e-marketplaces* are computers, electronics, chemicals, energy, automotive, construction, financial services, food and agriculture, health care, life science, metals, and telecom. Typical sites in these industries are as follows:

1. Computers and electronics: FastParts, PartMiner.com, PcOrder, and TechnologyNet
2. Chemicals and energy: AetraEnergy, Bloombey, CheMatch, ChemConnect, e-Chemicals, Enermetrix.com, Energy.com, HoustonStreet, and Industria
3. Automotive: Covisint, Cole Hersee Co.
4. Construction: Deere & Co., Bidcom, BuildNet, and Cephren
5. Financial Services: IMX Exchange, Muniauction, Pedestal, and Ultraprise
6. Food and Agriculture: Monsato Co., efdex, Floraplex, FoodUSA, and Gofish.com
7. Health Care and Life Sciences: Baxter Healthcare Corp., BioSuppliers.com, Chemdex, Neoforma, and SciQuest
8. Metals: e-Steel, iSteelAsia, MaterialNet, MetalShopper, and MetalSite
9. Telecom: Arbinet, Band-X, RateXchange, Simplexity.com, Telezoo, and The GTX.com

Since each industry has a different level of product fitness and readiness for digital marketing, the potential benefit of EC will not be the same. According to Forrester Research, the opportunity of B2B e-marketplaces can be categorized as in Fig. 2. According to this analysis, computers, electronics, shipping, and warehousing have the highest (above 70%) potential of e-marketplace saturation, while the heavy industry and aerospace industry have the lowest potential. So the selection of right industry and items are very important for the successful implementation of vertical e-marketplaces. In addition, the business partnership is critical for the success of this business.

### 2. Horizontal E-Marketplaces

The *horizontal e-marketplaces* support the aggregation of vertical e-marketplaces and the acquisition of common items and services that are necessary in all industries like MRO (maintenance, repair, and operations) items. The types of horizontal e-marketplaces are as follows:

1. Vertical e-marketplace aggregators: Typical aggregators are VerticalNet, Ventro, and Internos
2. Procurement solution provider initiative: A group of horizontal e-marketplaces are built under the initiative of e-procurement solution providers like Ariba, Commerce One, Clarus, Works.com, Suppplyaccess, PurchasePro, Peregrine/Harnbinger, and Unibex
3. MRO suppliers and services: The sites that handle the MRO items are MRO.com and W.W.Grainger
4. Industrial products: supplyFORCE
5. Surplus goods and equipment management: TradeOut, FreeMarkets, and AssetTrade
6. Information technology: Interwar
7. Sales and marketing: Art Technology Group, Broadvision, Calico, Firepond, and SpaceWorks.

Some *B2B auction* sites like FreeMarkets.com, Earmarked (www.fairmarket.com), and A-Z Used Computers (www.azuc.com) belong to the intermediary-centric B2B e-marketplaces.

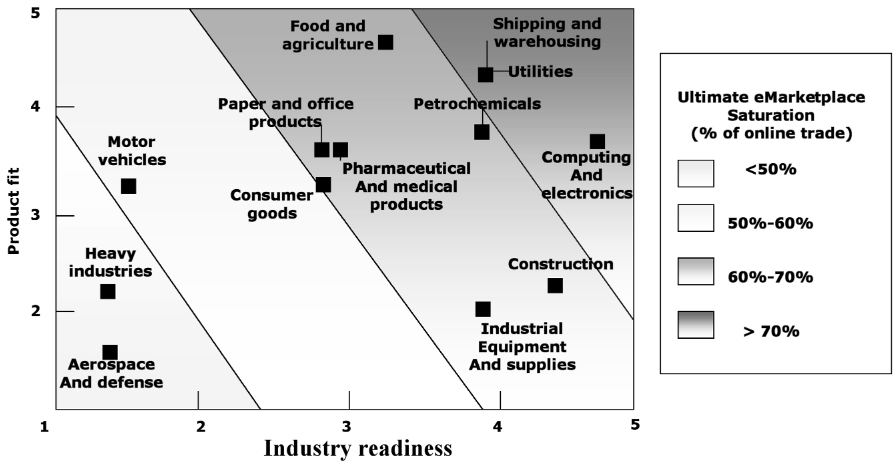


**Figure 2** The e-marketplace opportunity index. [From *The eMarketplace Opportunity,* Forrester Research. http://europa.eu.int/comm/enterprise/ict/e-marketplaces/presentation_homs.pdf. With permission.]

## C. Buyer-Centric E-Marketplaces

To buying businesses, the seller-centric e-marketplaces are convenient places to search one by one, however, they are inconvenient in a sense that they are segmented. The ordered information should be stored in the seller's server, therefore the buying company has to manually type in the order information in its procurement system. Thus, big buyers are compelled to build the buyer's own e-marketplace to manage the purchasing process efficiently.

Types of buyer-centric e-marketplaces are buyer initiated bidding sites, internalized e-marketplaces, and buyer's-coalition procurement sites. Buyer's bidding sites may evolve opening the site to the other potential buyers. Internalized e-marketplaces need to link and maintain consistency with the external e-marketplaces, and may open to outside buyers. This service can evolve to e-procurement service providers.

### 1. Buyer's Bidding Site

In this model, a buyer opens an e-market on its own server and invites potential suppliers to bid on the announced request for quotations. A successful example is the GE TPN case (tpn.geis.com). This model can offer a great sales opportunity to committed suppliers. However, as the number of such sites increases, suppliers will not be able to trace all such tender sites. At this stage, only very prominent buyers can take full advantage of this approach. Such government-driven sites are CBDNet (cbdnet.access.gpo.gov), GPO (www.access.gm.gov), COS (cos.gdb.org), EPIN (epin1.epin.ie), and GSD (www.info.gov.hk.gsd).

By building the buyer's bidding site, buyers can obtain the following benefits:

- Identifying and building partnerships with new suppliers worldwide
- Strengthening relationships and streamline sourcing processes with current business partners
- Rapidly distributing information and specifications to business partners
- Transmitting electronic drawings to multiple suppliers simultaneously
- Cutting sourcing cycle times and reducing costs for sourced goods
- Quickly receiving and comparing bids from a large number of suppliers to negotiate better prices

However, small companies cannot justify the cost of building their own bidding site for purchase, so the third party service has emerged. Examples of such sites are BIDCAST (www.bidcast.com), BIDLINE (www.bidline.com), BIDNET (www.bidnet.com), Federal Marketplace (www.fedmarket.cim), and GOVCON (www.govcon.com). Each site announces thousands of requests for bids.

By exploring the opportunities, suppliers in the buyer's bidding sites can take advantage of boosted sales, expanded market reach, lowered cost for sales and marketing activities, shortened selling cycle, improved sales productivity, and a streamlined bidding process. As the number of buyers increases, the burden of facilitating contacts between many buyers and many sellers becomes more serious. Manual handling of a high volume of transactions will not be possible let alone economical. So the software agents that work for buyers and sellers become essential, as described in Section VII.

### 2. Internalized E-Marketplaces

Another type of buyer-centric e-marketplace is the *internalized e-marketplace*. In this architecture, the internal e-catalog is accessible by the employees and the final requisitioners can order directly on-line. Then the order will be processed on the e-marketplace seamlessly and the procurement decision and ordering process can be tightly coupled with the internal workflow management systems, enhancing the efficiency of the procurement process.

In this architecture, the procurement department defines the scope of products and invites the preoffered prices. The posted prices will be stored in the internal database. MasterCard International developed the procurement card system, which allows the requisitioner to select goods and services from its own e-catalog containing more than 10,000 items.

An obstacle of this approach is the maintenance of the internal e-catalog so that it is consistent with the external e-marketplaces. For this purpose, the buyer's directory should be able to be coordinated in accordance with the change of e-marketplaces. Further research for this challenging issue can be found in Joh and Lee (2001).

### 3. Buyers-Coalition Procurement Sites

If every buyer builds its own internal e-marketplace, suppliers will be scared by complexity that they have to follow. So common buyers need to coalesce to reduce the complexity and incompatibility among buyers. The best example of this kind can be found in the automobile industry. GM, Ford, and DaimlerChrysler performed the joint project Automotive Network Exchange (ANX) to provide a common extranet platform to 30,000 suppliers. Then each of them started to build an independent procurement company, but instead they merged and built a common procurement company, Covisint. This case is a typical example of coalition among competitors and the moving the procurement activities to outsource.

## III. ON-LINE SERVICES TO BUSINESSES

In the e-marketplaces, the majority of items are hard goods. However, in cyberspace, service is more effective because the service can be completed on-line without any physical delivery. So let us review the status of online services to business.

- *Travel and tourism services:* Many large corporations have special discounts arranged with travel agents. To further reduce costs, companies can make special arrangements, which enable employees to plan and book their own trips on-line. For instance, Carlson Travel Network of Minneapolis provides an agentless service to corporate clients like General Electric. The GE employees can fill out the application at their intranet system.
- *Internal job market on the intranet:* Many companies conduct an internal electronic job market site on the intranet. Openings are posted for employees to look at, and search engines enable managers to identify talented people even if they were not looking actively for a job change. This is an example of intraorganizational e-commerce.
- *Real estate:* Since business real estate investment can be very critical, web sites cannot replace the existing agents. Instead, the web sites help in finding the right agents. However, some auctions on foreclosed real estate sold by the government may be opened on-line only to business.
- *Electronic payments:* Firm-banking on the Internet is an economical way of making business payments. The electronic fund transfers (EFT) using the financial EDI on the Internet is the most popular way that businesses use. The payment fee on the Internet is cheaper than other alternatives.
- *On-line stock trading:* Corporations are important stock investors. Since the fees for on-line trading are very low (as low as $14.95) and flat regardless the trading amount, the on-line trading brokerage service is a very attractive option for business investors.
- *Electronic auction to business bidders:* Some electronic auctions are open only to dealers. For instance, used cars and foreclosed real estate sold by the government are open only to dealers. The comprehensive list of auction sites is available in www.usaweb.com/auction.html.
- *On-line publishing and education:* On-line publishing is not the monopolistic asset of business. However, the subscribers of certain professional magazines are only for businesses. The on-demand electronic education programs can provide a useful training opportunity to busy employees.
- *On-line loan and capital makers:* Business loans can be syndicated on-line from the lending companies. IntraLink provides a solution for the service, and BancAmerica offers IntraLoan's matching service to business loan applicants and potential lending corporations. Some sites like www.garage.com provide information about venture capital.

- *Other on-line services:* Businesses are the major users of on-line consulting, legal advice, health care, delivery request, electronic stamping, escrowing, etc.

## IV. E-PROCUREMENT MANAGEMENT

As we have observed with the buyer-centric B2B e-marketplaces, one of the most important goals of B2B EC is effective procurement. So let us study the B2B EC from the evolutionary view of e-procurement management.

### A. Requisite for Effective and Efficient Procurement Management

All around the world, purchase and supply management (P&SM) professionals now advocate innovative purchasing as a strategic function to increase profit margins. Some of the tactics used in this transformation process are volume purchases, buying from approved suppliers, selecting the right suppliers, group purchasing, awarding business based on performance, improving quality of existing suppliers, doing contract negotiation, forming partnership with suppliers, and reducing paper work and administrative cost.

What many organizations fail to understand is that a fundamental change in businesses' internal processes must be implemented to maximize the full benefits of procurement reengineering. The two critical success factors which most organizations overlook are cutting down the number of routine tasks and reducing the overall procurement cycle through the use of appropriate technologies such as workflow, groupware, and ERP packages, as well as the B2B e-marketplaces. By automating and streamlining the laborious routine of the purchasing function, purchasing professionals can focus on more strategic purchases achieving the following goals:

- Reducing purchasing cycle time and cost
- Enhancing budgetary control
- Eliminating administrative errors
- Increasing buyers' productivity
- Lowering prices through product standardization and consolidation of buys
- Improving information management on suppliers and pricing
- Improving the payment process

To implement an effective and efficient e-procurement system, B2B EC needs to support either the integration of e-procurement systems with external e-marketplaces, or the integration of e-marketplaces with ERP packages.

### B. Integration of E-Procurement Systems with External E-Marketplaces

A group of e-procurement solution providers develops architectures that can support the integration of the buyer sites with external e-marketplaces and any independent suppliers. The architectures offered by Ariba and Commerce One are an example of this kind as depicted in Fig. 3. In this architecture, the solution has a pair of servers in the buyer and seller sites; namely BuySite and MarketSite.

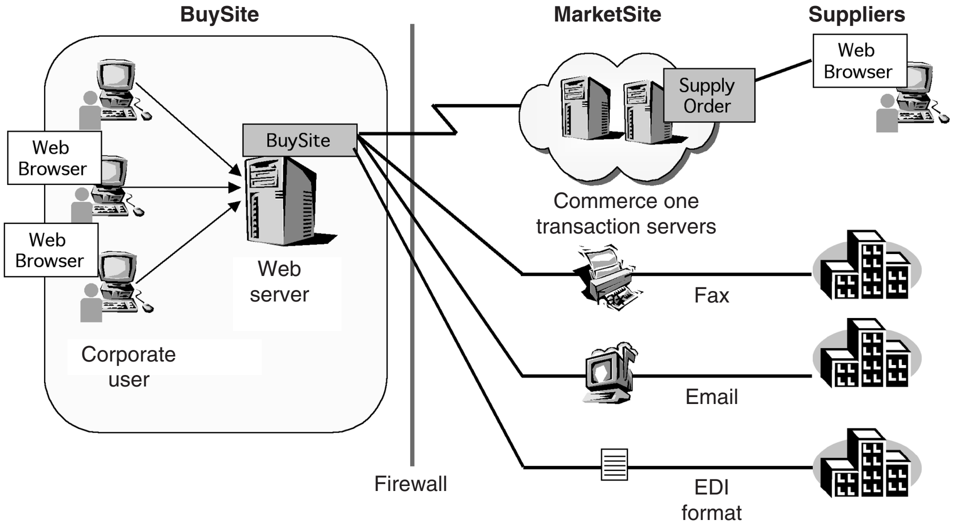


**Figure 3** An architecture of buyer-centric B2B e-marketplace. [Copyright © Commerce One Operations, Inc. 1999. All rights reserved.]

Suppliers can input their goods in the MarketSite server via the extranet. In the buyer site, the web server BuySite supports corporate users in the web-based search of e-catalog and ordering. The order will then be executed in the MarketSite. For the customers who have not joined the MarketSite, the orders can also be transmitted via fax, e-mail, and EDI message. To expand the capability of this architecture, many MarketSites should be able to support the BuySite. Therefore solution vendors try to make many vertical MarketSites and aggregate them.

## C. Integration of E-Marketplaces with ERP

Buyers tend to develop their back-end information systems as a combination of intranet, database, ERP, and legacy systems. ERP is enterprise-wide application software, which can provide a centralized repository of information for the massive amount of transactional detail generated daily. ERP integrates core business processes from planning to production, distribution, and sales. SAP's R/3 is one such software. Early versions of the ERP solution did not consider the integration with e-marketplaces, however, integration has become a critical issue in the B2B EC environment. Integration can be realized by adopting one of the following approaches: the inside-out, outside-in, and buyer's cart approaches.

### 1. The Inside-Out Approach: Extend ERP Outward

The leading ERP vendors offer a way to extend their solutions so that they are usable with the external e-marketplaces. This approach is called the inside-out approach as depicted in Fig. 4a. One scheme of this approach is that an ERP solution maker also provides an e-marketplace solution that is compatible with the ERP package. For instance, the solution called MySAP (www.mysap.com) is developed by SAP for this purpose. The other scheme is to build a strategic alignment with the e-marketplace solution providers to establish a mutually agreed upon interface. For instance, SAP has a strategic partnership with Commerce One.

When the e-marketplace solution requires a simple mapping of ERP functionality with a web interface, the inside-out architecture can be highly effective. It lets companies distribute ERP transaction capabilities to a wide audience of web users, without requiring that they load any specific client software on their PCs. However, from the e-marketplace's point of view, this approach is applicable only when the ERP system is installed. Many companies still use legacy systems.

### 2. The Outside-In Approach

In this approach, instead of extending the reach of ERP-based business processes through a web server, the software named application server integrates multiple back-end systems with an e-marketplace solution, as depicted in Fig. 4b. The outside-in architecture is better suited for complex e-business with multiple back- and front-end applications. In the outside-in approach, the e-business application resides within the application server, rather than within the individual back-end systems. Typical application servers are Application Server (Netscape), Enterprise Server (Microsoft), Domino (Lotus), Websphere (IBM), and Enterprise Server (Sun). However, the outside-in

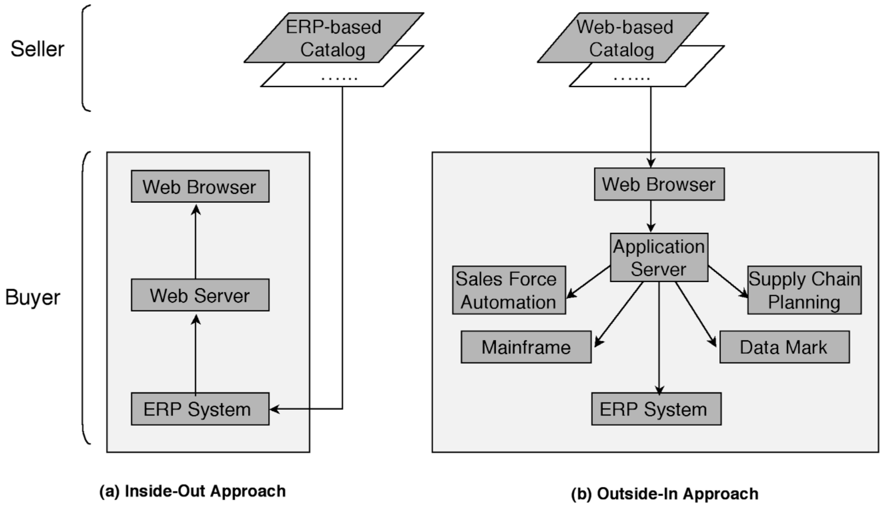


**Figure 4** Architectures of integrating EC with ERP. [From Sullivan, 1999a and b.]

approach is limited by the capabilities of the application server platforms upon which they are built.

### 3. Buyer's Cart Approach

In this approach, the buyer keeps a shopping cart in the buyer's PC or server instead of the seller's server. The items from multiple e-marketplaces can be tentatively selected and stored in the *buyer's electronic cart* (called b-cart). The order can also be made at the b-cart, and the result can be stored in the b-cart as well. With a standard file format in b-cart, the ERP or any other legacy system can be compatibly interfaced. This architecture is simple and economical, and it is well suited for the B2B EC environment.

## V. INTERNET-BASED EDI

The most basic instrument for B2B EC is efficient exchange of messages between companies. Thus Electronic Data Interchange (EDI) has been around for almost 30 years in the non-Internet environment. EDI is a system that standardizes the process of trading and tracking routine business documents, such as purchase orders, invoices, payments, shipping manifests, and delivery schedules. EDI translates these documents into a globally understood business language and transmits them between trading partners using secure telecommunications links. The most popular standard is the United Nations EDI for Administration, Commerce, and Trade (EDIFACT). In the United States the most popular standard is ANSI X.12. Traditional EDI users (most Fortune 1000 or global 2000 companies) use leased or dedicated telephone lines or a value-added network, such as those run by IBM and AT&T, to carry these data exchanges. Now the platform is moving to the Internet.

### A. Traditional EDI

Traditional EDI has changed the landscape of business, triggering new definitions of entire industries. Well-known retailers, such as Home Depot, Toys R Us and Wal-Mart would operate very differently today without EDI, since it is an integral and essential element of their business strategy. Thousands of global manufacturers, including Proctor & Gamble, Levi Strauss, Toyota, and Unilever have used EDI to redefine relationships with their customers through such practices as quick response retailing and JIT manufacturing. These highly visible, high-impact applications of EDI by large companies have been extremely successful.

However, despite the tremendous impact of traditional EDI among industry leaders, the current set of adopters represents only a small fraction of potential EDI users. In the United States, where several million businesses participate in commerce every day, fewer than 100,000 companies have adopted EDI (in 1998). Furthermore, most of the companies could maintain contact with only a small number of business partners on the EDI, mainly due to its high cost. Therefore, in reality, most businesses have not benefited from EDI. The major factors that limit businesses from benefiting from the traditional EDI are

- Significant initial investment is necessary
- Restructuring business processes is necessary to fit the EDI requirements
- Long start-up time is needed
- Use of expensive private value-added network (VAN) is necessary
- High EDI operating cost is needed
- There are multiple EDI standards
- The system is complex to use
- There is a need to use a converter to translate business transactions to EDI standards

These factors suggest that the traditional EDI—relying on formal transaction sets, translation software, and value-added networks—is not suitable as a long-term solution for most corporations, because it does not meet the following requirements:

- Enabling more firms to use EDI
- Encouraging full integration of EDI into trading partner's business processes
- Simplifying EDI implementation
- Expanding the capabilities of on-line information exchange

Therefore, a better infrastructure is needed; such infrastructure is the Internet-based EDI.

### B. Internet-based EDI

When considered as a channel for EDI, the Internet appears to be the most feasible alternative for putting on-line B2B trading within the reach of virtually any organization, large or small. There are several reasons for firms to create the EDI ability using the Internet:

- The Internet is a publicly accessible network with few geographical constraints. Its largest attribute, large-scale connectivity (without requiring any special company networking architecture), is a seedbed for growth of a vast range of business applications.

- The Internet global internetwork connections offer the potential to reach the widest possible number of trading partners of any viable alternative currently available.
- Using the Internet can cut communication cost by over 50%.
- Using the Internet to exchange EDI transactions is consistent with the growing interest of businesses in delivering an ever-increasing variety of products and services electronically, particularly through the web.
- Internet-based EDI can compliment or replace current EDI applications.
- Internet tools such as browsers and search engines are very user friendly and most people today know how to use them.

### 1. Types of the Internet EDI

The Internet can support the EDI in a variety of ways:

- Internet e-mail can be used as the EDI message transport in place of VAN. For this end, the Internet Engineering Task Force (IETF) considers standards for encapsulating the messages within the Secure Internet Mail Extension (S/MIME).
- A company can create an extranet that enables trading partners to enter information in web form whose fields correspond to the fields of an EDI message or document.
- Companies can utilize the services of a web-based EDI hosting service in much the same way that companies rely on third parties to host their commerce sites. Netscape Enterprise is illustrative of the type of web-based EDI software that enables a company to provide its own EDI services over the Internet, while Harbinger Express is illustrative of those companies that provide third-party hosting services.

### 2. Prospect of Internet EDI

Companies who currently possess traditional EDI respond positively to Internet EDI. A recent survey by Forester Research on 50 Fortune 1000 companies showed that nearly half of them plan to use EDI over the Internet. Frequently, companies combine the traditional EDI with the Internet by having their Internet-based orders transmitted to a VAN or a service provider that translates the data into an EDI format and sends it to their host computers. The Internet simply serves as an alternative transport mechanism to a more expensive lease line. The combination of the Web, XML (eXtensible Markup Language), and Java makes EDI worthwhile even for small, infrequent transactions. Whereas EDI is not interactive, the Web and Java were designed specifically for interactivity as well as ease of use.

## VI. SCM AND COLLABORATION AMONG ALIGNED PARTNERS

The major roles of e-marketplaces are the reducing search cost and competitive purchasing. However, when strategic partnership is essential, the advantage of the e-marketplace diminishes. In this setting, a more critical aspect is the elimination of uncertainty between companies along the supply chain, reducing the burden of inventory and buffer capacity. However, a lean supply chain is inherently vulnerable to the system collapse if one company in the chain cannot fulfill its mission.

In this section, we describe three types of B2B collaboration: the electronic SCM system, the shared data warehouse and data mining, and virtual corporations.

## A. Electronic Supply Chain Management (SCM) System

*Supply chain* is a business process that links material suppliers, manufacturers, retailers, and customers in the form of a chain to develop and deliver products as one virtual organization of pooled skills and resources. SCM is often considered an outgrowth of JIT manufacturing where companies operate with little or no inventory, relying instead on a network of suppliers and transportation partners to deliver the necessary parts and materials to the assembly line just as they are needed for the production.

Key functions in SCM are

- Managing information on demand to better understand the market and customer needs
- Managing the flow of physical goods from suppliers
- Managing the manufacturing process
- Managing the financial flows with suppliers and customers

In the early stage, SCM is attempted within a single factory, and then within an entire enterprise that may possess geographically dispersed workplaces. To link the supply chain between companies, the corresponding companies may be connected point-to-point. However, as the number of participating companies increases, the point-to-point connection becomes too expensive and technically too difficult for small companies. So the electronic hub-based collaboration become more feasible as depicted in Fig. 5, and the third party SCM service providers like i2 opened their e-hub service on the Internet.

**Convergence of e-hub**—The initial purpose of SCM was sharing information among aligned partners. But since the technical architecture of the e-hub is basically the same as that of the e-marketplace, the SCM service companies try to integrate the e-marketplace function with the hub. In the near future, we will observe the integration of the SCM hub and e-marketplaces. However, for the actual implementation, EC managers should judge which is more important: the transaction cost reduction among the aligned partners, or the flexible competitive selection of products and suppliers. Eventually by providing the combined service, participating companies will be able to enjoy the benefit of transaction cost reduction and competitive selection.

**JIT delivery**—Even though the SCM reduces the inventory by sharing the information, if there is no physical JIT delivery, implementation of SCM is nothing but a mirage. In the EC environment, the orders can arrive from any geographical locations, so the in-house delivery is not feasible in most cases. Therefore partnership with the third-party delivery companies like FedEx, United Parcel Service, and United States Postal Service becomes very important.

Key delivery companies provide on-line tracking service on their web sites. Moreover, the tracking information can be embedded in the manufacturer's information system so that the employees and its customers can utilize the information as if the manufacturer handles the delivery. The deliverers also provide the warehouse rental service for quick delivery and reduced carrying cost, and some value-added services like simple installation and repair.

For instance, National Semiconductor (NatSemi) dealt with a variety of different companies to get products from Asian factories to customers across the world, including freight forwarders, customs agents, handling companies, delivery companies, and airlines. They decided to outsource this entire process to FedEx. Today, virtually all of NatSemi's products, manufactured in Asia by three company factories and three subcontractors, are shipped directly to a FedEx distribution warehouse in Singapore. Each day, NatSemi sends its order electronically to FedEx. FedEx makes sure the order gets matched to a product and the product is delivered directly to the customer at the promised time. By going with FedEx as a one-stop shop for their logistics needs, NatSemi has seen a reduction of the average customer delivery cycle from four weeks to one week and their distribution costs drop from 2.9% of sales to 1.2%.

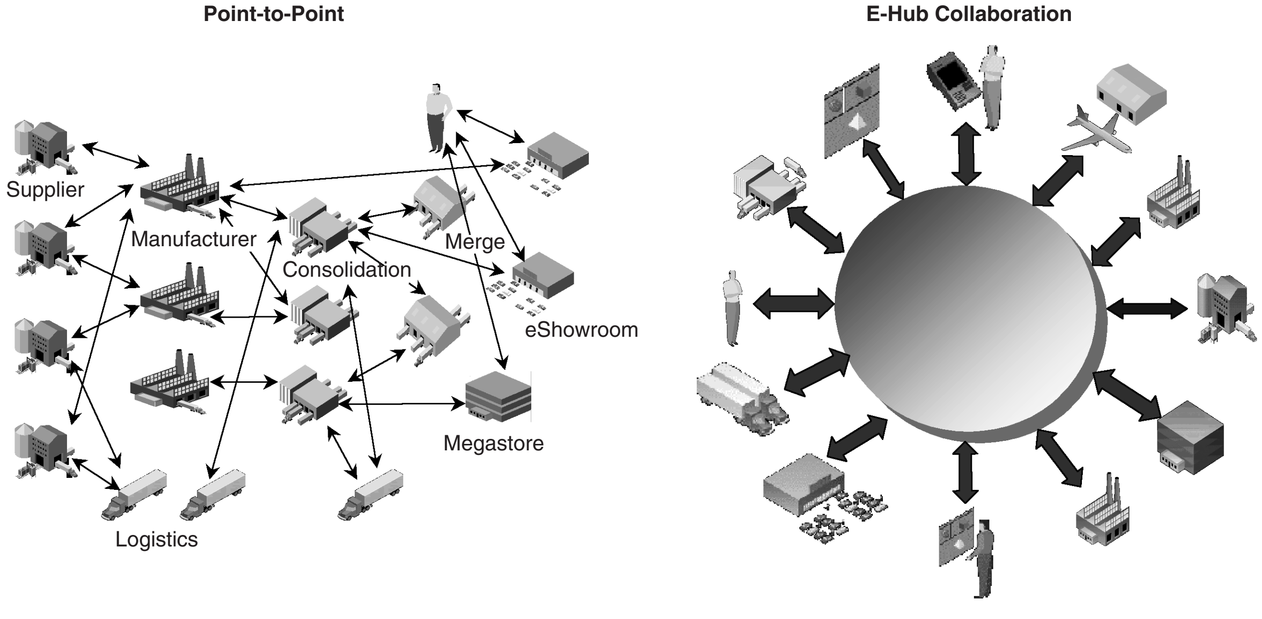


**Figure 5** Point-to-point and e-hub collaborated SCM.

## B. Shared Data Warehouse and Data Mining

Large retailers like Wal-Mart share their data warehouses with the suppliers to share the sales history, inventory, and demand forecast. This framework is illustrated in Fig. 6. This example is a case that a buyer owns its private SCM e-hub. For instance, Wal-Mart's 3570 stores share their data warehouse, RetailLink, with 7000 suppliers like Warner-Lambert. The world's largest RetailLink stores 2 years of sales history in 101 terabytes disks. There are about 120,000 data mining inquiries per week to RetailLink.

Traditionally, the retailers and suppliers forecasted separately, which resulted in excessive inventory, running out of stock, and lost opportunity for suppliers. However, with RetailLink, collaborative forecasting and replenishment become possible. Suppliers can use accurate sales and inventory data and take care of inventory management. By sharing RetailLink, Wal-Mart could display the right product in the right store at the right price. The retail industry is projected to save $150–250 billion per year.

## C. Virtual Corporation

One of the most interesting reengineered organization structures is the *virtual corporation* (VC). A virtual corporation is an organization composed of several business partners sharing costs and resources for the purpose of producing a product or service. According to Goldman et al. (1995), permanent virtual corporations are designed to create or assemble productive resources rapidly, frequently, concurrently, or to create or assemble a broad range of productive resources. The creation, operation, and management of a VC is heavily dependent on the B2B EC platform.

Sometimes a VC can be constructed with the partners in the supply chain. In this case, the SCM can be a vehicle of implementing the VC. However, VCs are not necessarily organized along the supply chain. For example, a business partnership may include several partners, each creating a portion of products or service, in an area in which they have special advantage such as expertise or low cost. So the modern virtual corporation can be viewed as a network of creative people, resources, and ideas connected via on-line services and/or the Internet.

The major goals that virtual corporations pursue are

- *Excellence:* Each partner brings its core competence, so an all-star winning team is created.
- *Utilization:* Resources of the business partners are frequently underutilized. A VC can utilize them more profitably.
- *Opportunism:* A VC can find and meet market opportunity better than an individual company.

The B2B EC platforms like the Internet and extranet will make the VC possible, because the communication and collaboration among the dispersed business part-

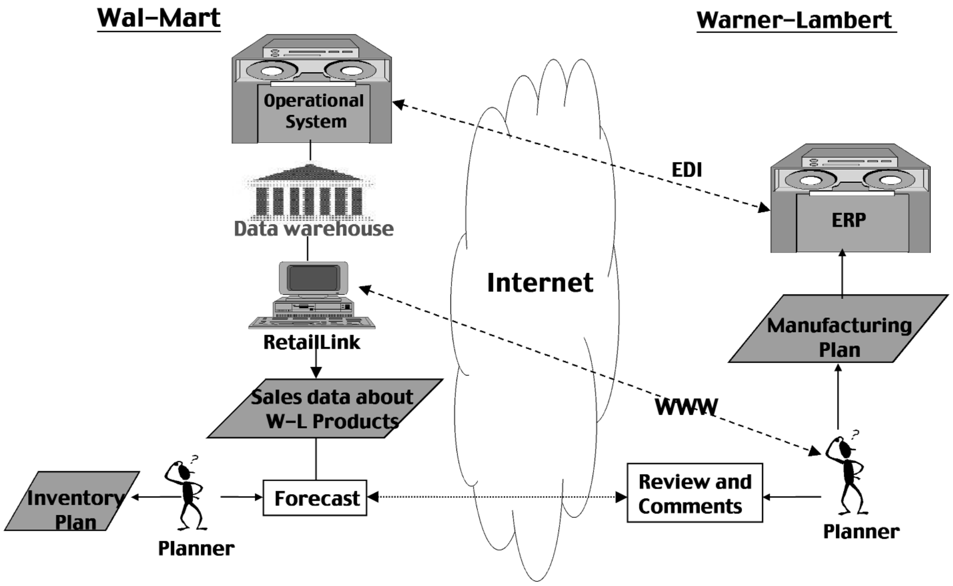


**Figure 6** Collaboration by shared data warehouse and data mining.

ners are the most critical essence to make it happen. *Extranet* is a network that links the intranets of business partners using the virtually private network on the Internet. On this platform, the business partners can use e-mail, desktop videoconferencing, knowledge sharing, groupware, EDI, and electronic fund transfer.

For instance, IBM Ambra formed a VC to take advantage of an opportunity to produce and market a PC clone. Each of five business partners played the following roles: engineering design and subsystem development, assembly on a build-to-order basis, telemarketing, order fulfillment and delivery, and field service and customer support. As the B2B EC platform propagates, more companies will be able to make VCs. More example cases, including Steelcase Inc. and The Agile Web, Inc., can be found in Turban et al. (1999) and the case AeroTech can be found in Upton and McAfee (1996).

## VII. AGENT-BASED COMMERCE

The necessity of agent-based commerce emerges, because B2B EC needs to utilize not only buyer agents but also seller agents, and also because these agents should work together.

**Role of buyer agents**—The purchase process consists of six steps: *need identification, product brokering, merchant brokering, negotiation, payment and delivery,* and *service and evaluation.* Among these steps, most agents are developed to assist the product brokering, merchant brokering, and negotiation process. For the buyers, the major roles of software agents are the collection of data from multiple e-marketplaces, filtering relevant items, scoring the products according to the customers' preference, and tabular comparison for side-by-side examination. Pricing Central.com lists the comparison search engines for each category of items. Buyer agents are mainly necessary in the seller-centric e-marketplaces. Buyer agents need to send out the requirement to the relevant seller agents, and interpret the received proposals.

**Role of seller agents**—On the other hand, in the buyer-centric e-marketplaces, sellers have to discover the relevant call for bids from multiple bidding sites. Proposal preparation for numerous buyer agents is another laborious and time-consuming task. So the seller agent must substitute for the transactional role of salesman as much as possible. To customize the proposal, seller agents should be able to understand the request for bid announced by buyer agents, identify its relevance to the items they handle, and generate the appropriate proposals.

In this manner, the interaction between buyer and seller agents becomes essential. A prototypical scenario of agent-based commerce is depicted in Fig. 7, and it works as follows:

1. A human buyer gives the requirement by his/her buyer agent.
2. The buyer agent generates a message (request for bids) to send to the relevant seller agents.
3. The seller agent interprets the message and identifies its relevance to the items they handle.
4. The seller agent generates an appropriate proposal and sends it back to the buyer agent.

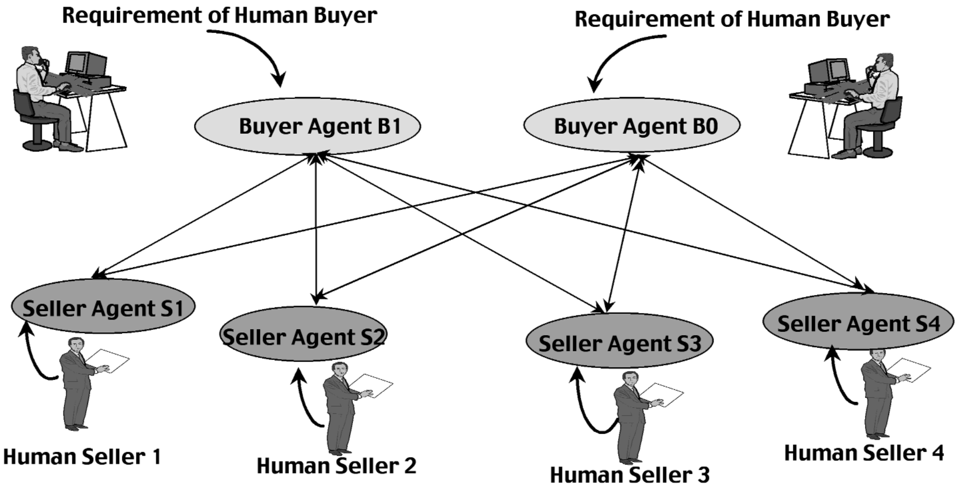


**Figure 7** A prototypical scenario of agent-based commerce. [From Lee, J. K., and Lee, W. (1997). Proceedings of the 13th Hawaii International Conference on System Sciences, pp. 230–241. With permission.]

5. The buyer agent compares the received proposals and reports the results to the human buyer for final decision-making. This step is the same as the comparison shopping aid in the current e-marketplaces.
6. The buyer agent receives the decision made by human buyer, and reports the selection to the bidders.

The procedure may vary depending upon the contract protocol adopted. So the protocol of contract, message format, representation of requirement, and specification of products are crucial elements of meaningful agent-based commerce. There is much research going on in various settings.

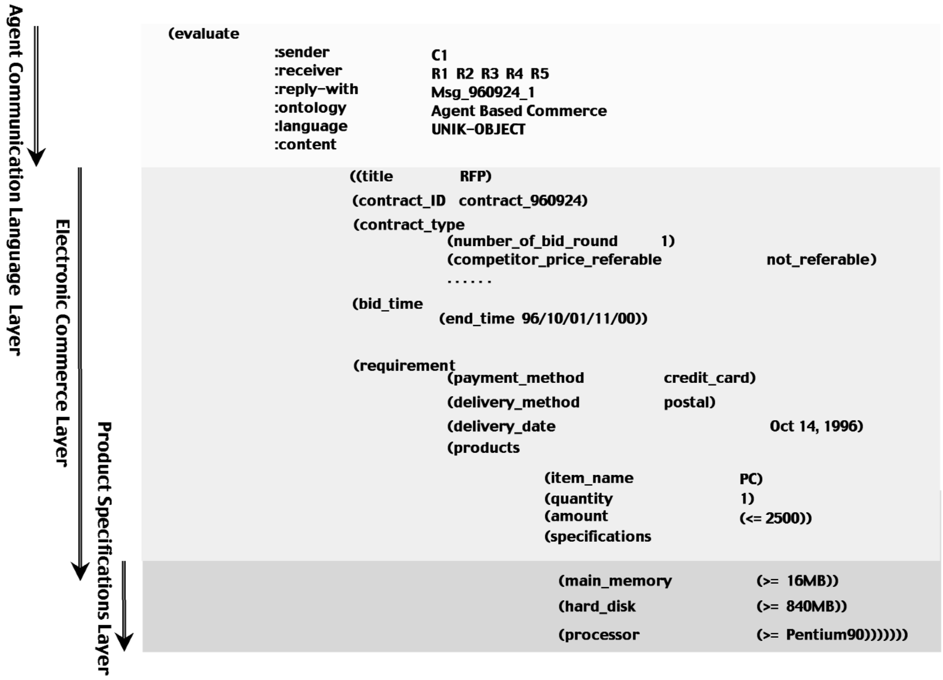
To exchange the messages in a compatible format, we need a common standard language called agent communication languages (ACL) like KQML (Knowledge Query and Manipulation Language). An illustrative message that requests a proposal from seller agents is demonstrated in Fig. 8. ACL consists of performatives and parameters. For instance, *evaluate* is a performative, and *sender* and *receiver* are parameters. To develop a dedicated ACL for EC, we need to augment the performatives and parameters to incorporate the generic terms necessary for EC, product specification, and buyers' requirement representation.

For instance in Fig. 8, the parameters such as *title, contract_ID, contract_type, bid_time, payment_method, delivery_method, delivery_date, item_name,* and *quantity* are the generic terms that are necessary in any agent-based commerce. So let us distinguish these terms as the *electronic commerce layer.* In the bottom of the message, the buyer's requirements are represented by the product specification. They are not generic, but depend upon the products to buy and sell. So this layer is distinguished as the *product specification layer.*

In many cases, the buyers may not be comfortable with expressing their requirements in the product specification level. It may be too technical and incomprehensible. Buyers are more familiar with the terms used in the buyer's environment. So mapping between the buyer's requirements expressions with seller's product specification is necessary. For this purpose, the salesman expert system can be used. For instance, in the process of computer purchase in the Personalogic site, customers may express the level of using word processing, network, graphics, etc. Then the system suggests the recommended memory requirement.

## VIII. XML IN B2B EC

Hypertext Markup Language (HTML) is developed to display text for human comprehension. For the agents to comprehend the global HTML files, the agents should be equipped with natural language processing capability. However, the natural language processing


Agent Communication Language Layer
Electronic Commerce Layer
Product Specifications Layer

```
(evaluate
            :sender          C1
            :receiver        R1  R2  R3  R4  R5
            :reply-with      Msg_960924_1
            :ontology        Agent Based Commerce
            :language        UNIK-OBJECT
            :content
                  ((title          RFP)
                  (contract_ID  contract_960924)
                  (contract_type
                            (number_of_bid_round        1)
                            (competitor_price_referable                not_referable)
                            ......
                  (bid_time
                           (end_time  96/10/01/11/00))

                  (requirement
                            (payment_method            credit_card)
                            (delivery_method           postal)
                            (delivery_date                             Oct 14, 1996)
                            (products
                                      (item_name               PC)
                                      (quantity                1)
                                      (amount                  (<= 2500))
                                      (specifications
                                        (main_memory            (>=  16MB))
                                        (hard_disk              (>=  840MB))
                                        (processor              (>=  Pentium90)))))))
```

**Figure 8** Three layers of message representation.

capability is limited only to the cases with a small vocabulary and structured grammar. Since the web sites cover all sorts of information, it is impossible to effectively extract the relevant data from the HTML files. So eXtensible Markup Language (XML) is developed to define the semantic data items in tags. The software agents are able to read out the values in the tags. XML files are also transformable to HTML files to display on the browser. To complement the XML files, the document type definition (DTD) defines the structure between data elements, and the eXtensible Stylesheet Language (XSL) defines the transformation of XML and DTD to HTML files for human comprehension.

XML has become popular as the second generation of web technology. For B2B EC, XML can be observed from the view of EDI and agents. From the EDI's point of view, XML is a tool to implement the EDI on the web. Users can just download the XML-based web page, and browse the transformed HTML files and retrieve the values of embedded data items. If the purpose of XML is just a message exchange, it can be regarded as the XML/EDI. However, the EDI community does not limit the function of XML just for the message exchange, as the traditional EDI did. When the XML is used in the context of agent-based commerce, our concern is not only message format but also the entire protocol. Therefore the EDI community and agent community are destined to converge to set up a commonly agreed upon B2B protocol implemented on XML.

So far, the major research focus in the agent community is not its implementing platform like XML because it is more practical than academic. On the other hand, the EDI task forces, mainly led by industrial experts, try to set up simple and useful business protocols implemented on XML. The main issue here is the establishment of standard protocols including business entities, procedure, message format, and product specification. For instance, the protocol OBI (open business interface) adopts the requisitioner, buying organization, supplier, and payment authority as the entities of B2B protocol. It is very difficult for a single protocol to meet the needs of all circumstances. So the standard establishment is booming to capture the strategically beneficial position in B2B EC.

Many task forces attempt to establish quite a number of XML-based business protocols. Some of them are

- OTP (open trading protocol). Proposed by a 30-company consortium for handling remote electronic purchase regardless of the payment mechanism (www.otp.org)
- OFX (open financial exchange). Proposed by Microsoft, Intuit, and Checkfree for exchanging financial transaction documents
- OSD (open software description). Proposed by Marimba and Microsoft for describing software package components to be used in automated software distribution environment
- OBI (open buying on the Internet). Proposed by American Express and major buying and selling organizations (www.openbuy.com)
- CBL (common business language). Supports the messages for the basic business forms used in ANSI X12 EDI (www.xmledi.com) transactions as well as those in OTP and OBI
- RossetaNet. Developed for PC industry (www.rosettanet.org)

Software vendors attempt to establish a standard, and develop solutions that can implement the standard protocol. Biz Talk developed by Microsoft is one of these. A concern at this point is that there are too many standards, so companies are confused which standard will really become the de facto standard in the market. To overcome this chaos, UN EDIFACT and OASIS have developed a standard ebXML. Many standard bodies, industry groups, vendors, users from around the world are integrating ebXML into their implementation. So the ebXML may become the de facto standard in the industry.

If the industry moves with the XML standard, agent research should take this movement into account to develop the practical agents that meet the standard. That is why the XML, EDI, and agent-based commerce will merge for effective B2B EC.

## IX. KNOWLEDGE MANAGEMENT IN THE B2B EC PLATFORM

Another important aspect of B2B EC is the management of knowledge in the web sites that are used by employees and partners. For instance, the e-catalog is an important source of knowledge about the products; the Q&A for the technical support is the source of technical knowledge; the regulation for budgetary control is important knowledge for budgeting.

The first users of visually displayed knowledge are human. So browsing HTML files is a way to gain it. Most of the current knowledge management systems belong to this category. It provides a search engine for retrieving the statements with the requested key words. The second users are software agents as mentioned in the previous section. For this purpose, we need to represent the data in XML files explicitly. However, XML files cannot process the implicitly embedded rules in the texts. So we need the third generation web technology, which can explicitly codify

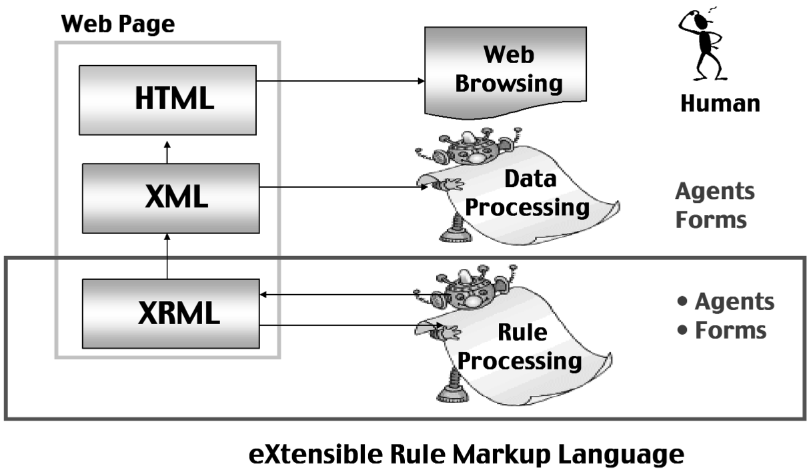


**Figure 9** Knowledge management with XRML.

the rules for inference engines to process. An issue here is the maintenance of consistency between the structured rules and natural language texts. To solve this issue, eXtensible Rule Markup Language XRML is under development, as depicted in Fig. 9.

To realize the rule processing with XRML files, we need three components:

1. *Rule structure markup language:* This language represents the rule in markup syntax so that the variables and values in the rule can be associated with the natural language texts.
2. *Rule identification markup language:* This language identifies the natural language sentences and its relationship with the rules, variables, and values used in the rules.
3. *Rule trigger markup language:* This meta-language defines the condition that the rules will be triggered, and is codified as embedded language in agents and forms in workflow management systems.

The knowledge editor should be able to support the consistent maintenance of natural language text and rules and any other programs codified for machine processing. By developing the XRML or a similar environment, the B2B EC can become more intelligent allowing visual display, a machine's data processing, and a machine's rule processing. Agents and workflow software will be more intelligent in this environment. More information about Rule Markup Language research initiatives can be found in cec.net.

## X. SUMMARY AND CONCLUSION

Key activities in B2B EC have been discussed in this paper. B2B EC started with the seller-centric e-marketplaces, and evolved to buyer-centric e-marketplaces. A large number of vertical marketplaces are being developed to help the exchange and supply chain of each industry. To integrate the e-procurement systems with the external e-marketplaces, the Inside-out and outside-in approaches are competitively being attempted. The EDI platform for B2B is moving onto the Internet. The electronic SCM system, shared data warehouse, and virtual corporations are getting provided on the e-hub exploring collaboration among the aligned partners. Agents that assist the buyers and sellers are establishing the agent-based commerce environment, and they will be implemented using XML. To build the standard B2B EC protocol, a large number of XML consortiums are conducting research.

Each of the above activities has started from the different angles for B2B interactions. However, eventually the B2B platform will converge to the e-hubs with a unified architecture, and will continuously seek the effective integration with the internal information systems. Initial development of B2B EC was pushed by information technology, but it will be concluded by the pull of meeting managerial goals.

## ACKNOWLEDGMENT

Materials in the author's book (Turban *et al.* 2000) is used with permission of the publisher.

## SEE ALSO THE FOLLOWING ARTICLES

Advertising and Marketing in Electronic Commerce • Electronic Commerce • Electronic Commerce, Infrastructure for • Enterprise Computing • Marketing • Sales • Service Industries, Electronic Commerce for

## BIBLIOGRAPHY

Blankenhorn, D. (May 1997). GE's e-commerce network opens up to other marketers. *NetMarketing*. www.netb2b.com.

Cunningham, M. J. (2000). *B2B: How to build a profitable E-commerce strategy*. Cambridge, MA: Perseus Publishing.

Davis, C. (September 1997). Most B-to-B sites don't meet customer needs: Gartner. *NetMarketing*. www.netb2b.com.

Finin, T., Wiederhold, J. W. G., Genesereth, M., Fritzson, R., McGuire, J., Shapiro, S., and Beck, C. (1993). Specification of the KQML Agent Communication Language, DARPA Knowledge Sharing Initiative, External Interface Working Group.

Freeman, L. (January 1998). Net drives B-to-B to new highs worldwide. *NetMarketing*. www.netb2b.com.

Frook, J. E. (July 1998). Web links with back-end systems pay off. *Internet Week*. www.internetwk.com.

Goldman *et al.* (1995). *Competitors and virtual organization*. New York: Van Nostrand Reinhold.

Handfield, R., and Nicols, E. (1999). *Supply chain management*. Upper Saddle River, NJ: Prentice Hall.

Joh, Y. H., and Lee, J. K. (2001). A framework of buyer's e-catalog directory management system. *Decision Support Systems*.

Kalakota, R., and Whinston, A. B. (1997). *Electronic commerce: A manager's guide*. Reading MA: Addison-Wesley.

Lawrence, E. *et al.* (1998). *Internet commerce: Digital models for business*. New York: John Wiley and Sons.

Lee, J. K. (1998). Next generation electronic marketing environment: ICEC perspective. *Proceedings of International Conference on Electronic Commerce '98*, p. 6. icec.net.

Lee, J. K. and Sohn, M. (2002). eXtensible Rule Markup Language—Toward the Intelligent Web Platform. Communications of the ACM. Forthcoming.

Lee, J. K., and Lee, W. (1997). An intelligent agent based contract process in electronic commerce: UNIK-AGENT approach. *Proceedings of 13th Hawaii International Conference on System Sciences,* pp. 230–241.

Lee, S. K., Lee, J. K., and Lee, K. J. (1997). Customized purchase supporting expert system: UNIK-SES. *Expert Systems with Applications,* Vol. 11, No. 4, pp. 431–441.

Lim, G., and Lee, J. K. (2002). Buyer-carts for B2B EC: The b-cart approach. Organizational Computing and Electronic Commerce, Forthcoming.

Maddox, K. (1998). Cisco wins big with net ordering. *NetMarketing*. www.netb2b.com/cgi-bin/cgi_article/monthly/97/05/01/article.html.

Maes, P., Guttman, R. H., and Moukas, A. G. (March 1999). Agents that buy and sell. *Communications of ACM,* Vol. 42, No. 3, 81–91.

Nelson, M. (July 1998). SAP adds module for I-commerce. *InfoWorld,* Vol. 21, No. 27.

Retter, T., and Calyniuk, M. (July 1998). *Technology Forecast: 1998*. Price Waterhouse.

Sculley, A. B., and Woods, W. W. (1999). *B2B Exchanges*. Philadelphia, PA: ISI Publications.

Silverstein, B. (1999). *Business-to-business internet marketing*. Gulf Breeze, FL: Maximum Press.

Silwa, C. (1998). Software improved net purchasing process. *ComputerWorld*. www.computerworld.com.

Sullivan, D. (January 1999a). Extending E-business to ERP. *e-business Advisor,* pp. 18–23.

Sullivan, D. (January 1999b). Take ERP on the road. *e-business Advisor,* pp. 24–27.

Trading Process Network. (1999). Extending the enterprise: TPN post case study—GE Lighting. tpn.geis.com/tpn/resouce_center/casestud.html.

Turban, E., Lee, J. K., King, D., and Chung, M. (2000). *Electronic Commerce: a managerial perspective*. Englewood Cliffs, NJ: Prentice Hall.

Turban, E., McLean, E., and Wetherbe, J. (1999). *Information technology for management,* 2nd Ed. New York: John Wiley & Sons.

Upton, D. M., and McAfee, A. (July 1996). The real virtual factory. *Harvard Business Review,* pp. 123–133.

Weston, W. (June 1998). Commerce one debuts SAP-oriented Tools. News.com. www.news.com/News/Item/0,4,23566,00.html.

# C and C++

**Jiang Guo**
*California State University, Bakersfield*

## GLOSSARY

**data encapsulation** The process of combining elements to create a new entity. For example, a complex data type, such as a class, is a type of data encapsulation because it combines built-in types or use-defined types and functions. Object-oriented programming languages rely heavily on data encapsulation to create high-level objects.

**dynamic binding** A method of attaching processor addresses to instructions and data during program execution. C++ implements dynamic binding through the use of virtual functions, which allows derived classes to override the base class functionality. Which function is invoked depends on the context in which the function is invoked at run time.

**inheritance** The concept that when a class is defined, then any subclass can inherit the definitions of one or more general classes. This means for the programmer that a subclass need not carry its own definition of data and methods that are generic to the class (or classes) of which it is a part. This not only speeds up program development; but it also ensures an inherent validity to the defined subclass object.

**memory leak** A bug in a program that prevents it from freeing up memory that it no longer needs. As a result, the program grabs more and more memory until it finally crashes because there is no more memory left.

**object-oriented programming** A type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects. To perform object-oriented programming, one needs an object-oriented programming language (OOPL). Java, C++, and Smalltalk are three popular OOPL languages, and there is also an object-oriented version of Pascal.

**programming language** A vocabulary and set of grammatical rules for instructing a computer to perform specific tasks. The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal. Each language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

**type checking** Ensures that all declarations and uses referring to the same object are consistent. It is also the key to determining when an undefined or unexpected value has been produced due to the type conversions that arise from certain operations in a language.

**C AND C++** are widely used for teaching, research, and developing large systems in industry. They are the most important computer languages. With a few modest exceptions, C++ can be considered a superset of the C programming language. While C++ is similar to C in syntax and structure, it is important to realize that the two languages are radically different. C++ and its support for object-oriented programming provide a new methodology for designing, implementing, and ease of maintaining software projects which C, a structured programming language, is unable to support. This article describes the histories of C and C++. It compares the advantages and disadvantages of the two languages and discusses the object-oriented features of C++. It also addresses some advanced topics of C++, such as storage and memory leaks, type checking, templates, and exceptions.

## I. HISTORIES OF C AND C++

The C programming language came into being in the years 1969–1973. It was developed at AT&T for the purpose of writing an operating system for PDP-11 computers. This operating system evolved into Unix. In 1978 Brian Kernighan and Dennis Ritchie published *The C Programming Language*. The C programming language was finally and officially standardized by the ANSI X3J11 committee in mid-1989. During the 1980s the use of the C language spread widely, and compilers became available on nearly every machine architecture and operating system; in particular it became popular as a programming tool for personal computers, both for manufacturers of commercial software for these machines, and for end users interested in programming. Today it is among the languages most commonly used throughout the computer industry.

C evolved from the B and BCPL programming languages. BCPL, B, and C all fit firmly into the traditional procedural family typified by FORTRAN and Algol 60. They are particularly oriented toward system programming, are small and compactly described, and are amenable to translation by simple compilers. They are low level programming languages. The abstractions that they introduce are readily grounded in the concrete data types and operations supplied by conventional computers, and they also rely on library routines for input-output and other interactions with an operating system. The programmers can use library procedures to specify interesting control constructs such as coroutines and procedure closures. At the same time, their abstractions lie at a sufficiently high level that, with care, portability between machines can be achieved.

BCPL, B, and C are syntactically different in many details. But broadly speaking, they are similar. Programs consist of a sequence of global declarations and function (procedure) declarations. Procedures can be nested in BCPL, but may not refer to nonstatic objects defined in containing procedures. B and C avoid this restriction by imposing a more severe one: no nested procedures at all. Each of the languages recognizes separate compilation, and provides a means for including text from named files which are called header files. BCPL, B, and C do not strongly support character data in the language; each treats strings much like vectors of integers and supplements general rules with a few syntactic conventions. In the C language, a string literal denotes the address of a static area initialized with the characters of the string, packed into cells. The strings are terminated by a special character "\0". It also should be pointed out that BCPL and B are the typeless languages, whereas C is a typed language (every variable and expression has a data type that is known as compile time).

The programming language C has several direct descendants, though they do not rival Pascal in generating progeny, such as Concurrent C, Objective C, C*, and especially C++. The C language is also widely used as an intermediate representation (essentially, as a portable assembly language) for a wide variety of compilers, both for direct descendents like C++, and independent languages like Modula 3 and Eiffel.

C++ is a programming language developed at AT&T Bell Laboratories by Bjarne Stroustrup in the early 1980s. In the first phase, Stroustrop merely viewed his work as an extension of C, calling it C with Classes. This was released around 1982. From 1983–1984 C with Classes was revised and renamed C++. Subsequent comments led to a revision and new release in 1985. The programming language C++ was designed with the intent of merging the efficiency and conciseness of C with the object-oriented programming (OOP) features of SIMULA-67. The language has evolved rapidly and several new features have been added since its initial release in 1985. The language also provides support for several other useful mechanisms such as parameterized types, templates, and exception handling. A formal ISO/ANSI C++ committee (X3J16) was established to help develop an accurate and reliable standard for the language in order to eliminate the ambiguities in the C++ compilers and translators of the time. The ISO/ANSI C++ language standard was officially approved in 1998 and adopted most of the rules present in the ANSI base document *The Annotated C++ Reference Manual* as written by Ellis and Stroustrup.

With a few modest exceptions, C++ can be considered a superset of the C programming language. While C++ is similar to C in syntax and structure, it is important to realize that the two languages are radically different. C++ and its support for OOP provide a new methodology for designing, implementing, and ease of maintaining software projects which C, a structured programming language, is unable to support.

Extensive libraries are available for the C programming language; consequently, a deliberate effort was made by the developers of C++ to maintain backward compatibility with C. Any major deviation from the C programming language would have meant that all the libraries available for C would have to be tediously rewritten for C++. This would have severely limited the usefulness of C++ in an environment where C libraries were used extensively.

C++ is largely an amalgamation of several other programming languages. Obviously, C++ inherits most of its characteristics, such as its basic syntax, looping mechanisms and the like, from C. Apart from C, C++ borrows most heavily from the aforementioned SIMULA-67 programming language. Nearly all the support that C++ provides for OOP comes from this language. The concept of a class and the so-called virtual function mechanism are a few of the features present in SIMULA-67 which have been integrated in C++.

To a limited extent, C++ also borrows some programming mechanisms from Algol 68. These include support for operator overloading and the declaration of variables almost anywhere in the code. As mentioned, the newer C++ compilers provide support for parameterized types and exception handling, concepts borrowed from Ada and Clu.

## II. C'S CHARACTERISTICS

C is a relatively small language, but one which wears well. C's small, unambitious feature set is a real advantage: there's less to learn and there isn't excess baggage in the way when programmers do not need it. It can also be a disadvantage: since it doesn't do everything for programmers, there's a lot they have to do themselves. (Actually, this is viewed by many as an additional advantage: anything the language doesn't do for programmers, it doesn't dictate to them either, so they are free to do that something however they want.)

### A. The Advantages of C

Despite some aspects mysterious to the beginner and occasionally even to the adept, C remains a simple and small language, translatable with simple and small compilers. The good news about C is that programmers can write code that runs quickly, and their program is very "close to the hardware." That means that they can access low-level facilities in computers quite easily without the compiler or run time system stopping them from doing something potentially dangerous. Its types and operations are well-grounded in those provided by real machines, and for people used to how computers work, learning the idioms for generating time- and space-efficient programs is not difficult. At the same time, the language is sufficiently abstracted from machine details that program portability can be achieved.

C is sometimes referred to as a "high-level assembly language." Some people think that is an insult, but it is actually a deliberate and significant aspect of the language. If a programmer has programmed in assembly language, he/she will probably find C very natural and comfortable (although if he/she continues to focus too heavily on machine-level details, he will probably end up with unnecessarily nonportable programs). If he/she has not programmed in assembly language, he/she may be frustrated by C's lack of certain higher level features. In either case, he/she should understand why C was designed this way: so that seemingly simple constructions expressed in C would not expand to arbitrarily expensive (in time or space) machine language constructions when compiled. If a programmer writes a C program simply and concisely, it is likely to result in a succinct, efficient machine language executable. If he/she finds that the executable program resulting from a C program is not efficient, it is probably because of something silly he/she did, not because of something the compiler did behind his back with which he has no control. In any case, there is no point in complaining about C's low-level flavor: C is what it is.

C imposes relatively few built-in ways of doing things on the programmer. Some common tasks, such as manipulating strings, allocating memory, and doing input/output (I/O), are performed by calling on library functions. Other tasks which a programmer might want to do, such as creating or listing directories, interacting with a mouse, displaying windows or other user-interface elements, or doing color graphics, are not defined by the C language at all. A programmer can do these things from a C program, of course, but he/she will be calling on services which are peculiar to his programming environment (compiler, processor, and operating system) and which are not defined by the C standard.

The use of compiler directives to the preprocessor makes it possible to produce a single version of a program which can be compiled on several different types of computers. In this sense C is said to be very portable.

The function libraries are standard for all versions of C so they can be used on all systems. C's central library support always remains in touch with a real environment. It was not designed in isolation to prove a point or to serve as an example, but as a tool to write programs that did useful things; it was always meant to interact with a larger operating system, and was regarded as a tool to build larger tools. A parsimonious, pragmatic approach influences the things that go into C: it covers the essential needs of many programmers, but does not try to supply too much.

C is quirky, flawed, and an enormous success. While accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments.

## B. The Disadvantages of C

The disadvantages of C fall neatly from the advantages. The biggest one is that a programmer can write C programs that can fail in very catastrophic ways. These programs will appear totally valid as far as the compiler is concerned, but will not work and may even cause computers to stop. A more picky language would probably notice that programmers were doing something stupid in their program and allow them to find the error before it crashed their computers. However, a more picky language would probably not allow them to write the program in the first place.

It is worth mentioning that C is a bit dangerous. C does not, in general, try hard to protect a programmer from mistakes. If a programmer writes a piece of code which will do something wildly different from what he intended it to do, up to and including deleting his data or trashing his disk, and if it is possible for the compiler to compile it, it generally will. C is often compared to a sharp knife: it can do a surgically precise job on some exacting task a programmer has in mind, but it can also do a surgically precise job of cutting off his finger. It is up to a programmer to use it carefully.

This aspect of C is very widely criticized; it is also used to argue that C is not a good teaching language. C aficionados love this aspect of C because it means that C does not try to protect them from themselves: when they know what they're doing, even if it's risky or obscure, they can do it. Students of C hate this aspect of C because it often seems as if the language is some kind of a conspiracy specifically designed to lead them into booby traps. This is another aspect of the language that is fairly pointless to complain about. If a programmer takes care and pays attention, he/she can avoid many of the pitfalls.

Another disadvantage of C is that it allows programmers to write very terse code. They can express exactly what they want to do in very few statements. They might think that this is nice, because it makes their programs even more efficient, but it has the side effect of making them much harder to understand. At the time a programmer writes the code he/she knows exactly what each part is supposed to do. If he/she comes back to the program in several months, he/she will need time to "get back inside it." If the code is written very tightly he/she will take much longer to do this, and other people may not be able to understand it at all. In contrast, many programmers strive to write code that is not necessarily the most efficient possible, but is easy to understand. Such programmers sacrifice a bit of program performance for ease of maintenance.

## C. A Critique of C

Two ideas are most characteristic of C among languages of its class: the relationship between arrays and pointers, and the way in which declaration syntax mimics expression syntax. They are also among its most frequently criticized features, and often serve as stumbling blocks to the beginner.

C treats strings as arrays of characters conventionally terminated by a marker (the character \0). Aside from one special rule about initialization by string literals, the semantics of strings are fully subsumed by more general rules governing all arrays, and as a result the language is simpler to describe and to translate than one incorporating the string as a unique data type. Some costs accrue from its approach: certain string operations are more expensive than in other designs because application code or a library routine must occasionally search for the end of a string, because few built-in operations are available, and because the burden of storage management for strings falls more heavily on the user. Nevertheless, C's approach to strings works well.

On the other hand, C's treatment of arrays in general (not just strings) has unfortunate implications both for optimization and for future extensions. The prevalence of pointers in C programs, whether those declared explicitly or arising from arrays, means that optimizers must be cautious, and must use careful dataflow techniques to achieve good results. Sophisticated compilers can understand what most pointers can possibly change, but some important usages remain difficult to analyze. For example, functions with pointer arguments derived from arrays are hard to compile into efficient code on vector machines because it is seldom possible to determine that one argument pointer does not overlap data also referred to by another argument, or accessible externally. More fundamentally, the definition of C so specifically describes the semantics of arrays that changes or extensions treating arrays as more primitive objects, and permitting operations on them as wholes, become hard to fit into the existing language. Even extensions to permit the declaration and use of multidimensional arrays whose size is determined dynamically are not entirely straightforward, although they would make it much easier to write numerical libraries in C. Thus, C covers the most important uses of strings and arrays arising in practice by a uniform and simple mechanism, but leaves problems for highly efficient implementations and extensions.

Many smaller infelicities exist in the language and its description besides those discussed above. There are also general criticisms to be lodged that transcend detailed points. Chief among these is that the language and its generally expected environment provide little help for writing very large systems. The naming structure provides only two main levels, "external" (visible everywhere) and "internal" (within a single procedure). An intermediate level of visibility (within a single file of data and procedures) is weakly tied to the language definition. Thus, there is little direct support for modularization, and project designers are forced to create their own conventions.

Similarly, C itself provides two durations of storage: "automatic" objects that exist while control resides in or below a procedure, and "static," existing throughout execution of a program. Off-stack, dynamically allocated storage is provided only by a library routine and the burden of managing it is placed on the programmer: C is hostile to automatic garbage collection.

## III. COMPARISON OF C AND C++

C++ is an extension of C developed at AT&T with the purpose of adding object-oriented features to C while preserving the efficiencies of C. For all practical purposes, the C language is a subset of C++ even though it is possible to write C programs that are not valid in C++. The main similarity in C and C++ lies in the syntax of the two languages. C and C++ both share many of the same fundamental programming constructs. This is the reason why it is easy for a proficient C programmer to learn C++ provided he/she understands the object-oriented paradigm. C++ supports every programming technique supported by C. Every C program can be written in essentially the same way in C++ with the same run time and space efficiency. It is not uncommon to be able to convert tens of thousands of lines of ANSI C to C-style C++ in a few hours. Thus, C++ is as much a superset of ANSI C as ANSI C is a superset of the original C and as much as ISO/ANSI C++ is a superset of C++ as it existed in 1985.

C++ maintains its C roots at various levels:

- *Source code level.* Most ANSI C programs are valid C++ programs.
- *Object code level.* C++ structures are "binary-compatible" with equivalent C structures.
- *Environment/tool level.* C++ works with standard tools like the *make* facility.

C++ can be viewed as a programming language derived from C with improved procedural syntax as compared to C and object-oriented features not present in C. Note that though C++ supports OOP, it does not enforce it. C++ is therefore a multiparadigm language. If what a programmer is looking for is something that forces him to do things in exactly one way, C++ isn't it. There is no one right way to write every program—and even if there were, there would be no way of forcing programmers to use it. Of course, writing C-style programs in C++ is not an optimal use of C++ for most applications. To be a truly effective C++ programmer, he/she must use the abstraction mechanisms and the type system in a way that fits reasonably with their intent. Trying to ignore or defeat the C++ type system is a most frustrating experience.

C is a procedural language. It is not designed to support OOP. In a procedural program, the problem is broken down into modules and submodules which implement the solution. C++ on the other hand can provide all the advantages inherent in the object-oriented paradigm. In an OOP the problem space is represented by objects that interact with each other, and these objects are implemented along with messaging mechanisms to provide a solution.

C++ supports data abstraction, OOP, and generic programming. OO programming languages have a number of inherent advantages. They lend themselves to better design because they allow problem spaces to be modeled like real-world objects. In an object oriented language, an object is called an instance of a class. A class packages all the attributes and methods of an object. Attributes are the data associated with the object and methods are the functions that operate on the data and express the behavior of the object. As an object-oriented language C++ supports inheritance, encapsulation, and polymorphism, which if properly used lead to better programs. These features are discussed in detail in the next section.

C++ provides stronger type checking than C. When a user defines a new type in C++, support is provided in the language to permit that type to behave in a manner similar to types already built into the language. The user may define how the standard operators act upon these user-defined types (operator overloading) and how these types can be converted to another type (user defined conversions). The user may also specify how memory is allocated or deallocated when an instance of that type is created or destroyed. This is done through the use of constructors and destructors, which are called implicitly at run time when an instance of that type is brought into and taken out of scope respectively.

C++ provides support for function prototypes (forward declarations of function signatures), hence enabling strong type checking of function parameters to take place during compilation. In addition, C++ provides support for the pass by reference mechanism and also supports default arguments to functions. The latter means that a function requires an argument that often has the same specific value, the user can default the argument to that value and not pass that parameter when the function is called. In the few cases where the function has to be called with a different value for the default argument, the user simply passes that argument into the function and the new value overrides the default value.

There is another aspect worth mentioning. Some people feel that C++ is a little overrated; in general this holds true for the entire OOP. Often it is said that programming in C++ leads to "better" programs. Some of the claimed advantages of C++ are

- New programs can be developed in less time because existing C++ code can be reused.
- Creating and using new data types is easier than in C.
- The memory management under C++ is easier and more transparent.
- Programs are less bug-prone, as C++ uses a stricter syntax and type checking.
- "Data hiding," the usage of data by one program part while other program parts cannot access the data, is easier to implement with C++.

Which of these allegations are true? Obviously, extreme promises about any programming language are overdone; in the end, a problem can be coded in any programming language (even BASIC or assembly language). The advantages or disadvantages of a given programming language aren't in "what a programmer can do with them," but rather in "which tools the language offers to make the job easier."

In fact, the development of new programs by reusing existing code can also be realized in C by, e.g., using function libraries: handy functions can be collected in a library and need not be reinvented with each new program. Still, C++ offers its specific syntax possibilities for code reuse in addition to function libraries.

Memory management is in principle in C++ as easy or as difficult as in C, especially when dedicated C functions such as `xmalloc()` and `xrealloc()` are used (these functions, often present in our C programs, allocate memory or abort the program when the memory pool is exhausted). In short, memory management in C or in C++ can be coded "elegantly," "ugly," or anything in between—this depends on the developer rather than on the language.

Concerning "bug proneness," C++ indeed uses stricter type checking than C. However, most modern C compilers implement "warning levels"; it is then the programmer's choice to disregard or heed a generated warning. In C++ many such warnings become fatal errors (the compilation stops).

As far as data hiding is concerned, C does offer some tools; e.g., where possible, local or static variables can be used and special data types such as structs can be manipulated by dedicated functions. Using such techniques, data hiding can be realized even in C; though it needs to be said that C++ offers special syntactical constructions. In contrast, programmers who prefer to use a global variable `int i` for each counter variable will quite likely not benefit from the concept of data hiding.

Concluding, C++ in particular and OOP in general are not solutions to all programming problems. C++, however, does offer some elegant syntactical possibilities, which are worth investigating. At the same time, the level of grammatical complexity of C++ has increased significantly compared to C. In time a programmer gets used to this increased level of complexity, but the transition doesn't take place quickly or painlessly.

In the strict mathematical sense, C isn't a subset of C++. There are programs that are valid C but not valid C++ and even a few ways of writing code that has a different meaning in C and C++. However, well-written C tends to be legal C++ also.

Here are some examples of C/C++ compatibility problems:

- Calling an undeclared function is poor style in C and illegal in C++, and so is passing arguments to a function using a declaration that doesn't list argument types.
- In C, a pointer of type `void*` can be implicitly converted to any pointer type, and free-store allocation is typically done using `malloc()`, which has no way of checking if "enough" memory is requested.
- C++ has more keywords than C.

## IV. ADVANTAGES OF C++

Most people think OOPs are easier to understand, correct, and modify. Besides C++, many other object-oriented languages have been developed, including most notably, Smalltalk. The best feature (some people feel this is the worst feature) of C++ is that C++ is a hybrid language—it is possible to program in either a C-like style, an object-oriented style, or both. Writing Smalltalk-style in C++ can be equally frustrating and as sub-optimal as writing C-style code in C++.

There is no formal definition of OOP. Hence there is some confusion surrounding what features a programming language must support in order to claim that it is object-oriented. Despite this, however, most agree that in order for a language to claim that it is object-oriented, it must provide support for three major concepts.

- Data encapsulation or data abstraction
- Inheritance or derivation
- Dynamic or run-time binding

The following subsections will explain these features and show how C++ provides support for them through its concept of a class; the underlying mechanism upon which all good C++ programs are based.

### A. Data Hiding and Encapsulation in C++

This is an important aspect of OOP languages and C++ offers several mechanisms to achieve data hiding. Encapsulation is the abstraction of information and is also called data hiding. It prevents users from seeing the internal workings of an object so as to protect data that should not be manipulated by the user. Encapsulation is supported in C++ through the class mechanism though the view of encapsulation differs from that in Eiffel. Data hiding and encapsulation form a crucial element in the protection mechanism within OOP languages. Encapsulation improves the modularity of the code and leads to more easy maintenance of programs by hiding the actual implementation details within an object and simply allowing access to an object's interface.

There are several ways in which data hiding can be achieved. It is true that C++ has the concept of an interface in which the "services" a class offers can be made accessible to other objects, however, the manner in which that interface can be described varies. Whatever the mechanism used, an object of any class A that wishes to send a message to an object of another class B needs to "include" the interface of class B in order to allow the compiler to perform its normal checks (method name, number of parameters, types of parameters, etc.).

C++ is backward compatible with C and the include mechanism derives from this. In effect, what happens is that a pre-compilation process occurs where any include directives are replaced with the contents of the file to be included. As this is a process that can be carried out separately, i.e., a programmer can carry out the precompilation without carrying on to the compilation stage it means that he can produce a text file that comprises his code together with the code from any include file. This does not seem to be a very good idea given the aims and objectives of data hiding and encapsulation.

The user can only perform a restricted set of operations on the hidden members of the class by executing special functions commonly called methods. The actions performed by the methods are determined by the designer of the class, who must be careful not to make the methods either overly flexible or too restrictive. This idea of hiding the details away from the user and providing a restricted and clearly defined interface is the underlying theme behind the concept of an abstract data type.

One advantage of using data encapsulation comes when the implementation of the class changes but the interface remains the same. For example, to create a stack class, which can contain integers, the designer may choose to implement it with an array, which is hidden from the user of the class. The designer then writes the `push()` and `pop()` methods which put integers into the array and remove them from the array, respectively. These methods are made accessible to the user. Should an attempt be made by the user to access the array directly, a compile time error will result. Now, should the designer decide to change the stack's implementation to a linked list, the array can simply be replaced with a linked list and the `push()` and `pop()` methods rewritten so that they manipulate the linked list instead of the array. The code that the user has written to manipulate the stack is still valid because it was not given direct access to the array to begin with.

The concept of data encapsulation is supported in C++ through the use of the public, protected, and private keywords, which are placed in the declaration of the class. Anything in the class placed after the public keyword is accessible to all the users of the class; elements placed after the protected keyword are accessible only to the methods of the class or classes derived from that class; elements placed after the private keyword are accessible only to the methods of the class.

As a convention, calling a method of an object instantiated from a class is commonly referred to as sending a message to that object.

## B. Inheritance in C++

There is a very important feature of C++, and object-oriented languages in general, called inheritance. Inheritance allows programmers to create a derived class from a previously defined base class. The derived class contains all the attributes and methods of the base class plus any additional specifications of the derived class. Any changes made to base classes are propagated to all derived classes unless explicitly overridden. Inheritance facilitates code reuse and thereby cutting development costs.

In the inheritance mechanism, the original class is called the "base" or "ancestor" class (also the "superclass"), and the derived class is the "derived" or "descendant" class (also the "subclass").

Inheritance is the mechanism whereby specific classes are made from more general ones. The child or derived class inherits all the features of its parent or base class, and is free to add features of its own. In addition, this derived class may be used as the base class of an even more specialized class.

Inheritance, or derivation, provides a clean mechanism whereby common classes can share their common features, rather than having to rewrite them. For example, consider a graph class, which is represented by edges and vertices and some (abstract) method of traversal. Next, consider a tree class, which is a special form of a graph. We can simply derive tree from graph and the tree class automatically inherits the concept of edges, vertices, and traversal from the graph class. We can then restrict how edges and vertices are connected within the tree class so that it represents the true nature of a tree.

Inheritance is supported in C++ by placing the name of the base class after the name of the derived class when the derived class is declared. It should be noted that a standard conversion occurs in C++ when a pointer or reference to a base class is assigned a pointer or reference to a derived class.

## C. Dynamic Binding of Function Calls in C++

Quite often when using inheritance, one will discover that a series of classes share a common behavior, but how that behavior is implemented is different from class to class. Such a situation is a prime candidate for the use of dynamic or run-time binding, which is also referred to as polymorphism.

Polymorphism allows different objects to respond differently to the same message. There are two types of polymorphism: (1) early binding, which allows overloading of functions; overloading means that different functions can have the same name but can be distinguished based on their signature (number, type and order of parameters) and (2) late binding, which allows derived classes to override the base class functionality. Which function is invoked depends on the context in which the function is invoked. Polymorphism improves the flexibility of programming by allowing developers better design options.

Going back to our previous example, we may decide to derive two tree classes from our graph class. The first class, `in_order_tree` would be traversed in an "in order" fashion when it received a `traverse()` message, whereas `post_order_tree` would be traversed in a "post order" manner. The different traversal algorithms could be incorporated into a dynamically bound `traverse()` method. Now, when one of these trees is passed to a function which accepts a reference to a graph class, the invocation of the `traverse()` method via the graph parameter would call the correct traversal algorithm at run time depending upon which tree was passed to the function. This reduces the burden on the programmer since a tag does not have to be associated with each class derived from a graph to distinguish it from other graphs. In addition, the programmer would not have to maintain an unwieldy switch statement to determine which traversal algorithm to invoke since this is all handled automatically by the compiler.

C++ implements dynamic binding through the use of virtual functions. While function calls resolved at run time are somewhat less efficient than function calls resolved statically, Stroustrup notes that a typical virtual function invocation requires just five more memory accesses than a static function invocation. This is a very small penalty to pay for a mechanism that provides significant flexibility for the programmer.

It is from inheritance and run time binding of function calls that OOP languages derive most of their power. Some problems lend themselves very well to these two concepts, while others do not. As Stroustrup notes: "How much types have in common so that the commonality can be exploited using inheritance and virtual functions is the litmus test of the applicability of object-oriented programming."

## V. ADVANCED TOPICS OF C++

### A. Storage and Memory Leaks

In C and C++, there are three fundamental ways of using memory:

1. *Static memory,* in which an object is allocated by the linker for the duration of the program. Global variables, static class members, and static variables in functions are allocated in static memory. An object allocated in static memory is constructed once and persists to the end of the program. Its address does not change while the program is running. Static objects can be a problem in programs using threads (shared-address-space concurrency) because they are shared and require locking for proper access.
2. *Automatic memory,* in which function arguments and local variables are allocated. Each entry into a function or a block gets its own copy. This kind of memory is automatically created and destroyed; hence the name automatic memory. Automatic memory is also said "to be on the stack."
3. *Free store,* from which memory for objects is explicitly requested by the program and where a program can free memory again once it is done with it (using the **new** and **delete** operators). When a program needs more free store, **new** requests it from the operating system. Typically, the free store (also called dynamic memory or the heap) grows throughout the lifetime of a program.

As far as the programmer is concerned, automatic and static storage are used in simple, obvious, and implicit ways. The interesting question is how to manage the free store. Allocation (using **new**) is simple, but unless we have a consistent policy for giving memory back to the free store manager, memory will fill up—especially for long-running programs.

The simplest strategy is to use automatic objects to manage corresponding objects in free store. Consequently, many containers are implemented as handles to elements stored in the free store.

When this simple, regular, and efficient approach isn't sufficient, the programmer might use a memory manager that finds unreferenced objects and reclaims their memory in which to store new objects. This is usually called automatic garbage collection, or simply garbage collection. Naturally, such a memory manager is called a garbage collector. Good commercial and free garbage collectors are available for C++, but a garbage collector is not a standard part of a typical C++ implementation.

Many problems encountered during C program development are caused by incorrect memory allocation or deallocation: memory is not allocated, not freed, not initialized, boundaries are overwritten, etc. C++ does not "magically" solve these problems, but it does provide a number of handy tools.

Memory leaks are perhaps the most famous dynamic memory problems, though they are by no means the only ones. A memory leak occurs when a program (either in application code or in library use) allocates dynamic memory, but does not free it when the memory is no longer useful. Memory leaks are especially problematic with X Window programs since they are often run for long periods of time (days or longer) and can execute the same event handling functions over and over. If there is a memory leak in X event handling code, an X application will continually grow in size. This will lead to decreased performance and possibly a system crash.

For example, each string must always be initialized before using it, since all strings must end with a null character (ASCII 0). The compiler interprets strings this way—it continues to process characters (in a cout or cin or string operation) until it hits a character with value 0—the null character. If this null character does not exist, then the computer will process the memory in the string and flow over, until it happens to hit a null character by chance or runs out of memory. This is also considered a memory leak and will probably crash the computer.

Here are some available tools to help debug the memory leak problem.

- Boehm-Weiser Conservative Garbage Collector: http://www.hpl.hp.com/personal/Hans_Boehm/gc/

- Centerline TestCenter: http://www.centerline.com/productline/test_center/test_center.html
- Debug Malloc Library, by Gray Watson: http://www.dmalloc.com
- MemCheck, by StratosWare: http://www.stratosware.com/products/MemCheck32/index.htm
- Memdebug, by Rene Schmit: http://www.bss.lu/Memdebug/Memdebug.html
- ParaSoft Insure++: http://www.parasoft.com/products/insure/index.htm
- Purify, by Rational Software, Inc.: http://www.rational.com/products/purify_nt/index.jsp

## B. Type Checking

Traditionally, a C or a C++ program consists of a number of source files that are individually compiled into object files. These object files are then linked together to produce the executable form of the program. Each separately compiled program fragment must contain enough information to allow it to be linked together with other program fragments. Most language rules are checked by the compiler as it compiles an individual source file (translation unit). The linker checks to ensure that names are used consistently in different compilation units and that every name used actually refers to something that has been properly defined. The typical C++ run-time environment performs few checks on the executing code. A programmer who wants run-time checking must provide the tests as part of the source code.

C++ interpreters and dynamic linkers modify this picture only slightly by postponing some checks until the first use of a code fragment.

- *Compile-Time Type Checking.* As with most other bug areas, the best debugging techniques are those that catch bugs at compile time rather than at run time. The compiler touches all of the code, so it can find errors that may only rarely occur at run time. At least occasionally, a programmer should set his compiler's warning output level to the most verbose setting and then track down and fix all the problems that it reports. Even if a report is not a critical problem, it may be worth fixing for portability reasons or to make real problems easier to find in the output. Compile-time error messages that are especially important with respect to pointer problems are those generated by function prototypes. Using incorrect pointer types in functions is a common and serious application programming problem. A programmer should enable this compiler feature all the time and immediately correct any problems it detects. Some problems almost always lead to program bugs, especially when the pointers are to data types of different sizes. Sometimes these problems are not immediately apparent. For example, the data types may be the same size on a particular machine, but the problems show up when a programmer tries to port the program to machines with other data type sizes.
- *Run-time Type Checking.* If a programmer cannot catch a bug at compile time, the next best thing for the system is to automatically halt his program with a core dump when the bug occurs. While a programmer never wants his end users to experience core dumps, they identify the program state at the time of the crash and can help him identify and debug many types of bugs.

The `assert()` macro, available with most C and C++ implementations, is a simple way to force a program to exit with an error message when unexpected results occur. Using assert() is an easy but powerful way to find pointer and memory problems.

A good programming technique is to initialize pointers to NULL and to reset them to NULL whenever the objects they point to are freed. If programmers do this, they can easily check for initialized pointers before using them.

## C. Operator Overloading

Operator overloading allows C++ operators to have user-defined meanings for user-defined types (classes). Overloaded operators are syntactic sugar for function calls. Overloading standard operators for a class can exploit the intuition of the users of that class. This lets users program in the language of the problem domain rather than in the language of the machine. The ultimate goal is to reduce both the learning curve and the defect (bug) rate. Some people don't like the keyword operator or the somewhat funny syntax that goes with it in the body of the class itself. But the operator overloading syntax isn't supposed to make life easier for the developer of a class. It's supposed to make life easier for the users of the class. Remember, in a reuse-oriented world, there will usually be many people who use class **R,** but there is only one person who builds it (**R**self); therefore he should do things that favor the many rather than the few.

Most operators can be overloaded. The only C operators that can't be are `.` and `?:` (and `sizeof`, which is technically an operator). C++ adds a few of its own operators, most of which can be overloaded except `::` and `.*`.

When compiling an expression of the form <var> <op> <value>, the compiler does the following:

1. If `<var>` is of a built-in type (`int, char*`, etc.), the standard (built-in) operator is used
2. If `<var>` is of a user-defined class type, the compiler checks if there is a suitable user-defined `operator<op>` function defined (that is, one whose parameter is of the same type as `<value>`, or of a type convertable to the type of `<value>`). If so, that function is used
3. Otherwise, a compiler error is flagged

Note that there are special problems if there are more than one "suitable" operator functions available. Such problems are resolved using the normal function overloading resolution rules

## D. Templates

The C++ language supports a mechanism that allows programmers to define completely general functions or classes, based on hypothetical arguments or other entities. These general functions or classes become concrete code once their definitions are applied to real entities. The general definitions of functions or classes are called templates; the concrete implementations, instantiations.

Templates automate the implementation of abstract algorithms that can operate on multiple data types. Considering Stack as an example, a different data class can be given for each instantiation of the Stack class without manually changing the Stack class definition. Instead, the compiler generates code specific for each data type listed as a specific class. For example, if there is a method that must be defined differently to print each different data type stored in the Stack class, this is done automatically by the template facility. Templates are valuable in promoting code reuse since different stack code can be generated for different data types from a single copy of the algorithm. Templates are satisfactory for constructing distinctly different stacks. But suppose a single stack that held data of different data type was required. Because C++ templates generate code specific for each data type statically (at compile time), the data type of the items stored in the stack must remain static during program execution. Different code is generated for different stacks, each able to hold only one data type.

The Standard Template Library (STL) represents a breakthrough in C++ programming methodology. Comprising a set of C++ generic data structures and algorithms, the STL provides reusable, interchangeable components adaptable to many different uses without sacrificing efficiency. Adopted by the ANSI/ISO C++ Standards Committee, the STL is an important addition to every C++ programmer's portfolio of skills. The STL is a general purpose library consisting of containers, generic algorithms, iterators, function objects, allocators, and adaptors. The data structures that are used in the algorithms are abstract in the sense that the algorithms can be used on (practically) any data type. The algorithms can work on these abstract data types due to the fact that they are template based algorithms.

## E. Exceptions

Improved error recovery is one of the most powerful ways a programmer can increase the robustness of his code. If a programmer can make several function calls with only one catch, he greatly reduces the amount of error-handling code he must write.

Unfortunately, it is almost accepted practice to ignore error conditions, as if programmers are in a state of denial about errors. Some of the reason is, no doubt, the tediousness and code bloat of checking for many errors. For example, `printf()` returns the number of characters that were successfully printed, but virtually no one checks this value. The proliferation of code alone would be disgusting, not to mention the difficulty it would add in reading the code.

The problem with C's approach to error handling could be thought of as one of coupling—the user of a function must tie the error-handling code so closely to that function that it becomes too ungainly and awkward to use.

One of the major features in C++ is exception handling, which is a better way of thinking about and handling errors. With exception-handling,

1. Error-handling code is not nearly so tedious to write, and it doesn't become mixed up with the "normal" code. A programmer can write the code he wants to happen; later in a separate section he writes the code to cope with the problems. If he makes multiple calls to a function, he handles the errors from that function once, in one place.
2. Errors will not be ignored. If a function needs to send an error message to the caller of that function, it "throws" an object representing that error out of the function. If the caller doesn't "catch" the error and handle it, it goes to the next enclosing scope, and so on until some code block catches the error.

There are two basic models in exception-handling theory. In termination (which is what C++ supports) programmers assume the error is so critical there is no way to get back to where the exception occurred. Whoever threw the exception decided there was no way to salvage the situation, and they don't want to come back.

The alternative is called resumption. It means the exception handler is expected to do something to rectify the situation, and then the faulting function is retried, presuming success the second time. If a programmer wants resumption, he still hopes to continue execution after the exception is handled, so his exception is more like a function call—which is how he should set up situations in C++ where he wants resumption-like behavior (that is, don't throw an exception; call a function that fixes the problem). Alternatively, the programmer can place his try block inside a while loop that keeps reentering the try block until the result is satisfactory. Historically, programmers using operating systems that supported resumptive exception handling eventually ended up using termination-like code and skipping resumption. So although resumption sounds attractive at first, it seems it isn't quite so useful in practice. One reason may be the distance that can occur between the exception and its handler; it's one thing to terminate to a handler that's far away, but to jump to that handler and then back again may be too conceptually difficult for large systems where the exception can be generated from many points.

## F. Use of C++

C++ is used by hundreds of thousands of programmers in essentially every application domain. This use is supported by about a dozen independent implementations, hundreds of libraries, hundreds of textbooks, several technical journals, many conferences, and innumerable consultants. Training and education at a variety of levels are widely available.

Early applications tended to have a strong systems programming flavor. For example, several major operating systems have been written in C++ and many more have key parts done in C++. C++ was designed so that every language feature is usable in code under severe time and space constraints. This allows C++ to be used for device drivers and other software that rely on direct manipulation of hardware under real-time constraints.

In such code, predictability of performance is at least as important as raw speed. Often, so is compactness of the resulting system. Most applications have sections of code that are critical for acceptable performance. However, the largest amount of code is not in such sections. For most code, maintainability, ease of extension, and ease of testing is key. C++'s support for these concerns has led to its widespread use where reliability is a must and in areas where requirements change significantly over time. Examples are banking, trading, insurance, telecommunications, and military applications. For years, the central control of the United States long-distance telephone system has relied on C++ and every 800 call (that is, a call paid for by the called party) has been routed by a C++ program. Many such applications are large and long-lived. As a result, stability, compatibility, and scalability have been constant concerns in the development of C++. Million-line C++ programs are not uncommon.

Like C, C++ wasn't specifically designed with numerical computation in mind. However, much numerical, scientific, and engineering computation is done in C++. A major reason for this is that traditional numerical work must often be combined with graphics and with computations relying on data structures that don't fit into the traditional FORTRAN mold. Graphics and user interfaces are areas in which C++ is heavily used.

All of this points to what may be C++'s greatest strength—its ability to be used effectively for applications that require work in a variety of application areas. It is quite common to find an application that involves local and wide-area networking, numerics, graphics, user interaction, and database access. Traditionally, such application areas have been considered distinct, and they have most often been served by distinct technical communities using a variety of programming languages. However, C++ has been widely used in all of those areas. Furthermore, it is able to coexist with code fragments and programs written in other languages.

C++ is widely used for teaching and research. This has surprised some who, correctly, point out that C++ isn't the smallest or cleanest language ever designed. However, C++ is clean enough for successful teaching of basic concepts,

- Realistic, efficient, and flexible enough for demanding projects
- Available enough for organizations and collaborations relying on diverse development and execution environments
- Comprehensive enough to be a vehicle for teaching advanced concepts and techniques
- Commercial enough to be a vehicle for putting what is learned into nonacademic use

There are many C++ compilers. Following is a list of popular C++ compilers.

- GNU C++ compiler
- Microsoft Visual C++
- Borland C++
- PARCompiler C++ and C
- The CC++ Programming Language—a parallel programming language based on C++
- Watcom C/C++ Compiler
- pC++/Sage++—a portable parallel C++ for high performance computers

## VI. CONCLUSION

The designers of C++ wanted to add object-oriented mechanisms without compromising the efficiency and simplicity that made C so popular. One of the driving principles for the language designers was to hide complexity from the programmer, allowing them to concentrate on the problem at hand.

Because C++ retains C as a subset, it gains many of the attractive features of the C language, such as efficiency, closeness to the machine, and a variety of built-in types. A number of new features were added to C++ to make the language even more robust, many of which are not used by novice programmers. Most of these features can be summarized by two important design goals: strong compiler type checking and a user-extensible language.

By enforcing stricter type-checking, the C++ compiler makes programmers acutely aware of data types in their expressions. Stronger type checking is provided through several mechanisms, including: function argument type checking, conversions, and a few other features. C++ also enables programmers to incorporate new types into the language through the use of classes. A class is a user-defined type. The compiler can treat new types as if they are one of the built-in types. This is a very powerful feature. In addition, the class provides the mechanism for data abstraction and encapsulation, which are a key to OOP.

C++ is a very useful and popular programming language. However, there are still some critiques. For example, the struct type constructor in C is a redundant feature when the class concept is introduced, and the worse, C++ sets up different accessibility rules for structures and classes. Struct is only in C++ as a compatibility mechanism to C. A struct is the same as a class that has by default public components. The struct keyword could not be eliminated without losing compatibility, but the class keyword is unnecessary.

As C is a subset of C++, C programmers can immediately use C++ to write and compile C programs, this does not take advantage of OOP. Many see this as a strength, but it is often stated that the C base is C++'s greatest weakness. However, C++ adds its own layers of complexity, like its handling of multiple inheritance, overloading, and others. Java has shown that in removing C constructs that do not fit with object-oriented concepts, that C can provide an acceptable, albeit not perfect base. One of the stated advantages of C++ is that programmers can get free and easy access to machine level details. This comes with a downside: if programmers make a great deal of use of low level coding their programs will not be economically portable. Java has removed all of this from C, and one of Java's great strengths is its portability between systems, even without recompilation.

## SEE ALSO THE FOLLOWING ARTICLES

COBOL • Fortran • Java • Pascal • Perl • Programming Languages Classification • Simulation Languages • Visual Basic • XML

## BIBLIOGRAPHY

Barton, J., and Nackman, L. (1994). *Scientific and Engineering C++*. Reading, MA: Addison-Wesley.

Brokken, F. (1994). C++ Annotations. University of Groningen.

Deitel & Deitel. (2001). *C How to Program.* Englewood Cliffs, NJ: Prentice Hall.

Deitel & Deitel. (2001). *C++ How to Program.* Englewood Cliffs, NJ: Prentice Hall.

Ellis, M., and Stroustrup, B. (1990). *The Annotated C++ Reference Manual.* Reading, MA: Addison-Wesley.

Joyner, I. (1999). *Objects Unencapsulated: Eiffel, Java and C++?* Englewood Cliffs, NJ: Prentice Hall.

Kernighan, B., and Ritchie, D. (1978). *The C Programming Language.* Englewood Cliffs, NJ: Prentice Hall.

Ritchie, D. (1993). *The Development of the C Language.* Second History of Programming Languages Conference, Cambridge, MA.

Stroustrup, B. (1994). *The Design and Evolution of C++*. Reading, MA: Addison-Wesley.

Stroustrup, B. (1997). *The C++ Programming Language.* Reading, MA: Addison-Wesley.

# COBOL

**Mohammed B. Khan**
*California State University, Long Beach*

## GLOSSARY

**alphabetic data field** A data field used to store alphabetic characters.

**CODASYL** The name of the organization that developed the first version of COBOL in 1960.

**data field** A single item contained within a record.

**documentation** Written statements that explain a program to humans.

**elementary data field** A data field that is not divided into subordinate components.

**file** An organized accumulation of related data.

**group data field** A data field that is divided into subordinate components.

**machine independent language** A computer programming language that can be executed on many different computers with little or no modifications.

**numeric data field** A data field used to store numbers.

**program maintenance** The activity of modifying preexisting programs.

**record** A group of data items pertaining to a single entity.

**COBOL (COMMON BUSINESS-ORIENTED LANGUAGE)** was once the dominant programming language for business applications. Many distinguishing features characterize this language among which are: structured modularity, efficient input-output capability, English-like sentences, and self-documenting nature. Though the language has undergone several revisions, its popularity has definitely reduced in recent years.

Many legacy systems written in COBOL are still in use. These systems are primarily run on mainframe computers. However, COBOL compilers have been written for smaller computers, including personal computers. COBOL programs interface gracefully with Structured Query Language (SQL) statements. Graphical user interfaces (GUI) have been implemented in COBOL, making the language more acceptable in today's visual environment.

## I. THE DEVELOPMENT OF COBOL

Cobol is one of the oldest high-level programming languages. Its development began in 1959 when a group of computer professionals from various organizations—government, education, business and others—agreed to hold a series of meetings to design a uniform, machine-independent computer programming language for business use. This group met under the name Conference on Data Systems Language or CODASYL. CODASYL developed the first version of COBOL, which was called COBOL-60 because it was released in 1960. Three revisions followed: COBOL-68, COBOL-74, and most recently, COBOL-85. Another revision of COBOL (COBOL-9X) is under development at this time.

Over the years, the American National Standards Institute (ANSI), an organization that adopts standards in a wide variety of fields, has adopted CODASYL standards for COBOL. Therefore, COBOL-68, COBOL-74, and COBOL-85 are often referred to as ANSI-68, ANSI-74, and ANSI-85, respectively. Each of these standards had new features that made it better and easier to use than the previous version.

Although more and more organizations are adopting COBOL-85, many organizations still use COBOL-74. Many existing COBOL programs that businesses use are COBOL-74. In fact, there are more COBOL-74 programs in the business world than there are COBOL-85 programs.

## II. COBOL'S DISTINGUISHING FEATURES

The original intent for the development of COBOL was to introduce a computer programming language that would have three major characteristics:

- It would be *machine independent,* meaning that COBOL programs could be executed on many different types of computers with little or no modification. For example, a COBOL program that was written to run on a VAX computer can be executed on an IBM computer with only minor modifications.
- It would be easy to maintain. *Program maintenance* is the process of making modifications to existing programs. In industry, the largest part of a programmer's time is spent modifying existing programs. Therefore, a program that is easy to maintain can save a company considerable time and money.
- It would be self-documenting. *Documentation* consists of explanations telling users how a computer program works or what the program is doing at a particular point. When a program is self-documenting, the programming language instructions, or code, contain English-like words and phrases.

COBOL has additional features that distinguish it from most other computer programming languages. Some of them are

- COBOL *programs are uniquely organized.* Each program is divided into four standard parts called *divisions.* These divisions will be discussed in detail later.
- COBOL *is well suited for commercial data processing.* Most of the data processing performed in business requires two major types of tasks: performing simple calculations, and inputting and outputting large quantities of data. For example, consider the tasks that a program performs when billing department store charge-account customers. The program reads the needed data, calculates the bill, and then outputs it in the desired form. Because there is likely to be a large number of customers, the program must input and output the data efficiently. In addition, it must calculate each bill using only addition and subtraction. COBOL easily and efficiently performs these operations.
- *The data in* COBOL *programs are organized into records.* Each record contains a collection of data. For example, each record used for the department store charge accounts program might contain one customer's account number, name, address, balance owed, and purchases made during the current month.
- COBOL programs can process several types of *files*—sequential, indexed sequential, and relative. A file consists of a group of records.
- A COBOL program can read and write massive amounts of data using only a few statements. These are referred to as input/output (I/0) operations. The COBOL language performs input/output operations efficiently.

## III. STRUCTURAL ORGANIZATION OF COBOL

The structure of a COBOL program is unique in that it consists of divisions and sections. Within a division and a section, a program contains statements that perform certain specific functions. COBOL programs are typed in an 80-column line format. The leftmost column is column 1, the rightmost column is column 80. Of the 80 columns, only 66 columns (columns 7–72) are actually used for the program itself. A unique feature of COBOL programs is that certain parts of the program must start in certain columns of the 80-column line. A sample COBOL program is presented in Table I.

### A. Column 1 through 6: Sequence Area

Columns 1 through 6 are known as the sequence area. These columns are used to provide program statement numbers. A computer program consists of statements that tell the computer the steps necessary to solve a specific problem. These statements must be arranged in a particular sequence. Normally, the computer follows these statements in the order they appear, one after another, unless specifically directed otherwise. For this reason, each statement in a program is given a sequence number. The first statement has the sequence number 1, the second statement has the sequence number 2, etc. Having six columns

**Table I** Sample COBOL Program

```
   ──────→  ***********************************************************************************
Column 7:
Indicator
Area        IDENTIFICATION DIVISION              This Division Identifies the Program and
                                                  Provides General Information.
            ***********************************************************************************

            PROGRAM-ID.SAMPLE.
            AUTHOR.M.B.KHAN.
            DATE-WRITTEN.JANUARY 5,1996.
            DATE-COMPILED.JANUARY 5,1996.
            ***********************************************************************************
            *
            *REMARKS. THIS PROGRAM CALCULATES GROSS PAY OF EMPLOYEES
            *BASED ON HOURLY RATES AND NUMBER OF HOURS WORKED.
            *OVERTIME PAY IS CALCULATED AT THE RATE OF TIME AND A
            *HALF FOR HOURS WORKED OVER 40. THE PROGRAM PRINTS
            *EMPLOYEE NAME, HOURLY RATE, HOURS WORKED, AND GROSS
            *WEEKLY PAY. SPECIFICALLY, THE PROGRAM
            *(1) READS NAME, HOURLY RATE, AND HOURS WORKED FOR AN
            *    EMPLOYEE
            *(2) CALCULATES REGULAR OR OVERTIME PAY, AS APPROPRIATE
            *(3) PRINTS NAME, HOURLY RATE, HOURS WORKED, AND GROSS
            *    PAY FOR THE EMPLOYEE
            ***********************************************************************************
            ***********************************************************************************
            ENVIRONMENT DIVISION
            ***********************************************************************************
                                                 This Division Describes the Computing
                                                 Environment-Specific Computers that
                                                 Will Be Used-for the Program, and the
                                                 Interface with Hardware Devices.
            CONFIGURATION SECTION.
            SOURCE-COMPUTER. IBM3090.
            OBJECT-COMPUTER. IBM3090.
            INPUT-OUTPUT SECTION
            FILE-CONTROL
            SELECT EMPLOYEE-FILE-IN
                 ASSIGN TO "PROG31I.DAT".
            SELECT EMPLOYEE-FILE-OUT
                 ASSIGN TO "PROG310.DAT".
      *****************************************************************************************
            DATA DIVISION ←──── This Division Describes All Data that
                                Are Needed by the Program
      *****************************************************************************************
            FILE SECTION.
      *****************************************************************************************
      *THE FOLLOWING STATEMENTS DEFINE THE INPUT FILE                                          *
      *****************************************************************************************
            FD EMPLOYEE-FILE-IN
                 LABEL RECORDS STANDARD.
            01   EMPLOYEE-RECORD-IN.
                 05 EMPLOYEE-NAME-IN PIC X(20).
                 05 EMPLOYEE-RATE-IN PIC 9(3)V99.
                 05 EMPLOYEE-HOURS-INPIC 9(3)V99.
      *****************************************************************************************
```

(continues)

**Table I** Sample COBOL Program *(continued)*

```
      *THE FOLLOWING STATEMENTS DEFINE THE OUTPUT FILE                                          *
*********************************************************************************************
            FD EMPLOYEE-FILE-OUT
                 LABEL RECORDS OMITTED.
      EMPLOYEE-RECORD-OUT.
                 05 EMPLOYEE-NAME-OUT     PIC X(20).
                 05                       PIC X(5).
                 05 EMPLOYEE-RATE-OUT     PIC 9(3).99.
                 05                       PIC X(5).
                 05 EMPLOYEE-HOURS-OUT PIC 9(3).99.
                 05                       PIC X(5).
                 05 EMPLOYEE-PAY-OUT PIC 9(5).99.
            WORKING-STORAGE SECTION.
            01   W1-CONTROL-FIELDS.
                 05 W1-END-OF-FILE  PIC X           VALUE “N”.
            01   W2-WORKING-FIELDS.
                 05 W2-REGULAR-HOURSPIC 9(2)     VALUE 40.
                 05 W2-REGULAR-PAY  PIC   9(5)V99   VALUE 0.
                 05 W2-OVERTIME-FACTOR  PIC 9V9     VALUE 1.5.
                 05 W2-OVERTIME-HOURS  PIC 9(3)V99  VALUE 0.
                 05 W2-OVERTIME-RATE PIC 9(3)V99 VALUE 0.
                 05 W2-OVERTIME-PAY PIC 9(5)V99  VALUE 0.
                 05 W2-TOTAL-PAY    PIC 9(5)V99  VALUE 0.
*********************************************************************************************
*           PROCEDURE DIVISION.◄— This Division Describes the
                                    Step-By-Step Instructions
                                    for Solving the problem
*********************************************************************************************
*********************************************************************************************
* THE FOLLOWING IS THE MAIN PARAGRAPH. FROM THIS PART, THE   *
* PROGRAM TRANSFERS TO OTHER PARTS FOR SPECIFIC FUNCTIONS    *
*********************************************************************************************
      AOOO-CONTROL-LOGIC.
                 PERFORM B100-INITIALIZE-PROCESSING.
                 PERFORM B200-PROCESS-DATA
                      UNTIL W1-END-OF-FILE EQUAL TO “Y”.
                 PERFORM B300-TERMINATE-PROCESSING.
                 STOP RUN.
*********************************************************************************************
* THE FOLLOWING PARAGRAPH OPENS THE INPUT AND THE            *
* OUTPUT FILE AND READS THE FIRST INPUT RECORD               *
*********************************************************************************************
                 B100-INITIALIZE-PROCESSING
                 OPEN INPUT EMPLOYEE-FILE-IN
                            OUTPUT EMPLOYEE-FILE-OUT.
                 PERFORM X100-READ-DATA.
                 B100-EXIT.
                 EXIT.
*********************************************************************************************
* THE MAIN PROCESSING IS CONTROLLED IN THE NEXT PARAGRAPH    *
* THE PROGRAM IS TRANSFERRED FROM HERE TO OTHER PARTS FOR    *
*                APPROPRIATE PROCESSING                      *
*********************************************************************************************
                 B200-PROCESS-DATA.
                   PERFORM C100-CALC-REGULAR-PAY.
                   IF EMPLOYEE-HOURS-IN IS GREATER THAN W2-REGULAR-HOURS
                         PERFORM C200-CALC-OVERTIME-PAY
```

*(continues)*

**Table I** Sample COBOL Program *(continued)*

```
                   ELSE
                            MOVE W2-REGULAR-PAY TO W2-TOTAL-PAY
                   END-IF
                   PERFORM X200-WRITE-DATA.
                   PERFORM X100-REAO-DATA.
                 B200-EXIT.
                   EXIT.
*********************************************************************************************
* INPUT AND OUTPUT FILES ARE CLOSED IN THE NEXT PARAGRAPH      *
*********************************************************************************************
                 B300-TERMINATE-PROCESSING.
                   CLOSE EMPLOYEE-FILE-IN
                   EMPLOYEE-FILE-OUT.
                 B300-EXIT.
                   EXIT.
*********************************************************************************************
* THE FOLLOWING PARAGRAPH CALCULATES REGULAR PAY               *
*********************************************************************************************
                 C100-CALC-REGULAR-PAY.
                   IF EMPLOYEE-HOURS-IN IS GREATER THAN W2-REGULAR-HOURS
                       MULTIPLY EMPLOYEE-RATE-IN BY W2-REGULAR-HOURS
                          GIVING W2-REGULAR-PAY
                 ELSE
                       MULTIPLY EMPLOYEE-RATE-IN BY EMPLOYEE-HOURS-IN
                          GIVING W2-REGULAR-PAY
                   END-IF
                 C100-EXIT.
                   EXIT.
*********************************************************************************************
* THE FOLLOWING PARAGRAPH CALCULATES OVERTIME PAY              *
*********************************************************************************************
                 C200-CALC-OVERTIME-PAY.
                   SUBTRACT W2-REGULAR-HOURS FROM EMPLOYEE-HOURS-IN
                      GIVING W2-OVERTIME-HOURS.
                   MULTIPLY W2-OVERTIME-FACTOR BY EMPLOYEE-RATE-IN
                      GIVING W2-OVERTIME-RATE.
                   MULTIPLY W2-OVERTIME-HOURS BY W2-OVERTIME-RATE
                      GIVING W2-OVERTIME-PAY.
                   ADD W2-OVERTIME-PAY W2-REGULAR-PAY
                      GIVING W2-TOTAL-PAY.
                 C200-EXIT.
                   EXIT.
*********************************************************************************************
* THE INPUT FILE IS READ IN THE FOLLOWING PARAGRAPH            *
*********************************************************************************************
               X100-READ-DATA.
                 READ EMPLOYEE-FILE-IN
                   AT END MOVE “Y” TO W1-END-OF-FILE
                 END READ
               X100-EXIT.
                 EXIT.
*********************************************************************************************
* THE NEXT PARAGRAPH MOVES DATA VALUES TO OUTPUT DATA          *
*     FIELDS AND WRITES THE OUTPUT DATA                        *
*********************************************************************************************
```

*(continues)*

**Table I** **Sample COBOL Program** *(continued)*

```
X200-WRITE-DATA.
   MOVE SPACES TO EMPLOYEE-RECORD-OUT.
   MOVE EMPLOYEE-NAME-IN TO EMPLOYEE-NAME-OUT.
   MOVE EMPLOYEE-RATE-IN TO EMPLOYEE-RATE-OUT.
   MOVE EMPLOYEE-HOURS-IN TO EMPLOYEE-HOURS-OUT.
   MOVE W2-TOTAL-PAY TO EMPLOYEE-PAY-OUT.
   WRITE EMPLOYEE-RECORD-OUT.
X200-EXIT.
 EXIT.
```

means that one could have hundreds of thousands of statements in a program if necessary.

The sequence numbers in columns 1 through 6 are a holdover from the days when programs were punched on cards. If the cards were dropped, they could easily be reordered by using these numbers. Today, programmers type their programs directly into the computer and sequence numbers are generated automatically by compilers. Therefore, sequence numbers have little use, and the programmer generally leaves these columns blank. A few sequence numbers have been provided in Table I to show where they would appear.

## B. Column 7: Indicator Area

Column 7 is called the *indicator* area. This column is used for special purposes. A specific entry in this column has a special meaning. For example, an asterisk (*) in this column indicates a comment statement. A slash (/) makes the computer go to the top of a new page before printing that line. A hyphen (-) indicates that the line is a continuation of the preceding line.

## C. Column 8 through 72: Area A and Area B

Columns 8 through 72 are known as *program areas.* Columns 8 through 11 are known as Area A, and columns 12 through 72 are known as Area B. These two areas represent the columns in which the COBOL program appears. The first column of Area A (column 8) is called the "A margin" and the first column of Area B (column 12) is called the "B margin." Some parts of a COBOL program must start in Area A, while others must start in Area B.

## D. Columns 73 through 80: Identification Area

Columns 73 through 80 are designated as the *identification area.* Program identification codes were useful when programs were punched on cards in preventing the cards of different programs from getting mixed together. The use of this area is optional, and the COBOL compiler ignores whatever is typed in it. For this reason, care should be taken not to type any significant part of a program statement in this area. The columns have not been filled in the sample program. It is important to note that every line of the sample program ends with a period; this is considered an important part of COBOL programs.

## E. Hierarchical Organization and Divisions in COBOL

The structural organization of a COBOL program is perhaps its most striking feature. The program follows a hierarchical organization. The basic structural unit of a program is the division. As previously mentioned, every COBOL program is divided into four divisions: the `IDENTIFICATION DIVISION`, the `ENVIRONMENT DIVISION`, the `DATA DIVISION`, and the `PROCEDURE DIVISION`. These divisions must be present in a specific sequence.

The `IDENTIFICATION DIVISION` is the first division in a COBOL program. Its purpose is to identify the program and its author and to provide general information about the program, such as the dates the program is written and compiled, any program security, etc. A narrative description of the program is usually included in this division.

The `ENVIRONMENT DIVISION` is the next division that appears in a COBOL program. The `ENVIRONMENT DIVISION` describes the "computing environment" of the program. The "computing environment" refers to the type of computer hardware on which the program is written and run. This division also briefly describes the data files required by the program.

The `DATA DIVISION` is the third division of a program. All data that are part of input and output files as well as other data are described in this division. Consequently, the `DATA DIVISION` is one of the most complicated and lengthy parts of a COBOL program. This division bears some relationship to the `ENVIRONMENT DIVISION` in that the files specified earlier are further described in the `DATA DIVISION`. Data are defined in COBOL in a specific way.

The `PROCEDURE DIVISION` contains the step-by-step instructions that are necessary to convert the input data to the desired output data. The instructions that perform a specific function are grouped together under a paragraph. Most COBOL statements start with regular English verbs. The divisions are divided into sections, which are again divided into paragraphs. Each paragraph is composed of several entries/clauses or sentences/statements.

The `IDENTIFICATION DIVISION` is divided into several paragraphs. There are four paragraphs in this division in Table I. However, only the `PROGRAM-ID` paragraph is required.

The `ENVIRONMENT DIVISION` is divided into sections. There are two sections in the sample program of Table I. These sections are the `CONFIGURATION SECTION` and the `INPUT-OUTPUT SECTION`, which are again divided into paragraphs. Paragraphs have either entries or clauses associated with them. Both entries and clauses perform the same function; they describe the paragraphs with which they are associated. Neither the division name nor the name of the first section is required.

The `DATA DIVISION` is divided into two sections. These sections have several entries that are used to describe the data used by the program. The first section is the `FILE SECTION`, the other is the `WORKING-STORAGE SECTION`.

The `PROCEDURE DIVISION` can be divided into sections or paragraphs. Each section or paragraph consists of one or more sentences or statements. A sentence is a collection of one or more statements. A statement is a valid combination of words and characters according to the rules of COBOL syntax. Most COBOL statements start with regular English verbs. There are eight paragraphs in the `PROCEDURE DIVISION` in Table I.

## IV. DATA ORGANIZATION FOR COBOL PROGRAMS

The input and the output data for a COBOL program are organized in the form of a file. A file is an organized accumulation of related data. Input data are contained in an input file; output data are contained in an output file. Any number of input and output files can be used in a COBOL program. A file is divided into records, and records are, in turn, subdivided into data fields.

### A. Files, Records, and Data Fields

A computer does not read all the data from a file at one time. Data are read in segments. A file is divided into many segments of data; each segment is called a *record*. Each record pertains to a single entity such as an individual person or item. For example, a school may have a computer file that has the data pertaining to all students. In this file, each record contains all the data of a single student. If there are 5000 students in the school, this file will have 5000 records. When instructed to read data from an input file, the computer reads one record at a time. Similarly, when instructed to write data to a file, the computer writes one record at a time. The record of an input file is called an input record; the record of an output file is called an output record.

Just as a file is divided into records, a record is divided into smaller segments, called *data fields,* or simply fields. An input or output record may be divided into any number of data fields; the number of data fields depends on the specific problem being solved by the computer. In COBOL, data fields can be of three different types: *numeric, alphabetic,* and *alphanumeric*. The data fields that represent only the digits 0 through 9 are called numeric. Examples of data that might appear in numeric data fields are age, social security number (not including the hyphens), price, sales, quantity, and number of employees. The data fields that represent alphabetic characters and spaces only are called alphabetic. Examples are names of individuals, and names of cities, states, or countries. Alphanumeric data fields can represent both numeric and nonnumeric data. Alphanumeric data include letters of the alphabet and special characters such as hyphens, parentheses, blanks, etc. Examples are telephone numbers (including the parentheses and hyphens), social security numbers (including the hyphens), dates of birth (including the hyphens), and addresses. Although most names contain letters of the alphabet only, there are names that consist of alphabets and special characters such as the apostrophe in the name LINDA O'DONNELL. For this reason, most COBOL programs store names as alphanumeric data.

A data field that is subdivided into subordinate components is called a *group data field;* a data field not further divided into components is called an elementary data field. For example, the data field "date of birth" can be subdivided into three subordinate components: "day of birth," "month of birth," and "year of birth." In this case, date of birth is a group data field; its components are elementary data fields. In a similar manner, the data field "mailing address" is a group data field. Its components, "residence number," "street name," "city name," "state name," and "zip code," are elementary data fields. A group data field may consist of a combination of numeric and nonnumeric elementary data fields as in the case of the "mailing address" data field.

It is important to be able to distinguish between a data field name and the data value it contains. A data field may be thought of as the name of a container, and the data value as the actual contents of the container.

COBOL offers considerable latitude in the naming of files, records, and data fields. However, certain rules must be observed.

**Rules for Choosing Names of Files, Records, and Data Fields**

1. A name must be contructed from the letters of the alphabet, the digits 0 through 9, and the hyphen. No other characters are allowed.
2. A name must not be longer than 30 characters.
3. A name must not begin or end with a hyphen.
4. A name must contain at least one letter of the alphabet. Digits only and digits in combination with hyphens are not allowed.
5. A name must not be a reserved COBOL word. Reserved COBOL words have preassigned meanings.

The names of files, records, and data fields should be selected carefully so that these names are self-explanatory. Usually, programmers select names that consist of English words joined by hyphens. Thus, a file for students may be named STUDENT-FILE; a file for the master inventory of a company may be named INVENTORY-MASTERFILE. A record in the STUDENT-FILE may be named STUDENT-RECORD; the data fields of this record may be named STUDENT-NAME, STUDENTADDRESS, STUDENT-PHONE-NUMBER, etc. Based on the five rules for naming files, records, and data fields, the following are all valid names:

```
INPUT-EMPLOYEE-FILE
EMPLOYEE-RECORD
EMPLOYEE-ADDRESS-1
CUSTOMER-FILE-IN
CUSTOMER-RECORD-I
CUSTOMER-BALANCE
FILE- 123
```

However, the following names are invalid:

```
EMPLOYEE FILE ACCOUNTS-RECEIVABLES-
                          MASTER-FILE
INVENTORY:RECORD
123-FILE
```

## B. Structural Description of a Record

In COBOL, records and data fields are described in a special way. A clear understanding of the structural description of a record is essential for writing COBOL programs.

### 1. Levels

Records and data fields are described in COBOL using *levels.* Levels are designated by numbers 01 through 49. All records must have level number 01 and their data fields must have level numbers 02 through 49. A common practice is to use level numbers in increments of five—5, 10, 15, and so on—instead of increments of one. Though both practices are allowed in COBOL, levels are usually numbered 05, 10, 15, and so on. The 05 level number indicates the highest level data field. This data field may be subdivided into smaller data fields whose level numbers are 10. These data fields with level number 10 may be further divided into smaller data fields whose levels are 15, and so on. In this way, level numbers 01 through 49 can be used in a COBOL program for describing records and data fields.

An example will make this clearer. Suppose STUDENT-RECORD is the name of a record. This record consists of five data fields: STUDENT-NAME, STUDENT-ADDRESS, STUDENT-DOB, STUDENT-CLASS (for student classification), and STUDENT-MAJOR. This record and its five data fields can be described using the structure given below.

The level number 01 for records may start anywhere in Area A but usually starts in column 8. Similarly, the level number for data fields may start anywhere in Area B, though it usually starts in column 12 and every fourth column thereafter. The record name starts in column 12 (B margin); the data field name starts in column 16. It is customary to leave two blank spaces between the level number and the name of the record or data field.

The data field STUDENT-DOB can be divided into three smaller data fields: STUDENT-BIRTH-DAY, STUDENT-BIRTH-MONTH, and STUDENT-BIRTH-YEAR. If this is done, the structure of the record and the data fields will appear as shown below.

```
01  STUDENT-RECORD
   05  STUDENT-NAME
      05  STUDENT-ADDRESS
             05  STUDENT-DOB
                    10  STUDENT-BIRTH-DAY
                           10  STUDENT-BIRTH-
                                         MONTH
                                  10  STUDENT-
                                     BIRTH-YEAR
      05  STUDENT-CLASS
             05  STUDENT-MAJOR
```

The only restriction is that each subordinate data field must have a higher level number than the level number of the data field to which it is subordinate. As each data field subdivides, each new level of data fields is given a higher number. Level 77 and level 88 are two other levels of data fields used in COBOL programs, though level 88 is much more common than level 77.

## 2. PICTURE Clause

The description of a record and its data fields is not complete without understanding and using the PIC (short for *PICTURE*) *clause*. Each elementary data field must be defined by a PIC clause that provides information as to its type (numeric, alphabetic, or alphanumeric) and its size. A group data field's length attribute is defined by the PIC clauses of its subordinate data fields, but it is always considered to be alphanumeric. A specific picture character designates a specific type of data.

The picture character 9 designates numeric data; the picture character A designates alphabetic data; and the picture character X designates alphanumeric or nonnumeric data. The picture character A is rarely used in COBOL programs; in practice, the picture character 9 is used for numeric data and the picture character X is used for non-numeric data. The PIC clause is written at the end of the data field (leaving at least one blank space after the data field name) with the word PIC followed by at least one blank space and the appropriate picture character (X or 9 or A). If a data field requires more than one picture character, the number of characters to which the picture character applies is enclosed in parentheses following the picture character.

The PIC clause of a student record can be written as follows:

```
01  STUDENT-RECORD.
    05  STUDENT-NAME  PIC  X(20).
    05  STUDENT-ADDRESS  PIC  X(30).
    05  STUDENT-DOB.
        10  STUDENT-BIRTH-DAY  PIC  99.
        10  STUDENT-BIRTH-MONTH  PIC  99.
        10  STUDENT-BIRTH-YEAR  PIC  9999.
                                     ↑
Four-character  numeric  field───────┘
    05  STUDENT-CLASS  PIC  X(10).
    05  STUDENT-MAJOR  PIC  X(15).
```

According to these PIC clauses, the data field STUDENT-NAME can contain up to 20 characters of non-numeric data indicated by the number 20 in parentheses following the X. This field could also be written using 20 Xs:

```
05  STUDENT-NAME  PIC
                        XXXXXXXXXXXXXXXXXXXX.
```

However, the first method requires fewer keystrokes and offers fewer opportunities for error. Similarly, STUDENT-ADDRESS may contain up to 30 characters of non-nummic data (each blank in the address counts as one space). The other data fields contain data of the following types and sizes (Table II):

Each PIC character occupies one byte of computer memory. For example, STUDENT-NAME occupies 20 bytes of computer memory, and STUDENT-ADDRESS occupies 30 bytes. The entire STUDENT-RECORD occupies 83 bytes of computer memory.

A record name can have a PIC clause provided it is not divided into component data fields. If the record is subdivided, then each component data field must carry a PIC clause. PIC clauses can only be used with elementary data fields. The following illustrations make this clear:

```
01  STUDENT-RECORD   PIC  X(83).
01  STUDENT-RECORD   PIC  X(83).
    05  STUDENT-NAME   PIC  X(20).
    05  STUDENT-ADDRESS   PIC  X(30).
    05  STUDENT-DOB.
```

**Table II** Data Fields, Types and Sizes

| Data field | Type | Maximum characters/digits |
|---|---|---|
| STUDENT-BIRTH-DAY | Numeric[a] | 2 |
| STUDENT-BIRTH-MONTH | Numeric[a] | 2 |
| STUDENT-BIRTH-YEAR | Numeric[a] | 4 |
| STUDENT-CLASS | Nonnumeric[b] | 10 |
| STUDENT-MAJOR | Nonnumeric[b] | 15 |

[a] Assuming these data are to be described as numbers such as 16 (for STUDENT-BIRTH-DAY), 02, (for STUDENT-BIRTH-MONTH) and 1976 (for STUDENT-BIRTH-YEAR).

[b] Assuming these data are to be described with words such as "freshman" (for STUDENT-CLASS) and "management" (for STUDENT-MAJOR).

```
         10 STUDENT-BIRTH-DAY PIC 99.
         10 STUDENT-BIRTH-MONTH PIC 99.
         10 STUDENT-BIRTH-YEAR PIC 9(4).
      05 STUDENT-CLASS PIC X(10).
      05 STUDENT-MAJOR PIC X(15).
```

In the first example STUDENT-RECORD has a PIC clause because it is not divided into subordinate data fields. This record is defined correctly. However, in the second example, the same record is divided into subordinate data fields, and thus, it should not have a PIC clause. A PIC clause must be provided for all input and output data fields as well as for all other data fields that may be required in a program.

Similarly, data fields that are divided into subordinate data fields must not have the PIC clause. The following examples are given to illustrate this point.

```
01 STUDENT-RECORD.
   05 STUDENT-NAME  PIC X(20).
   05 STUDENT-ADDRESS.
      10 STREET-ADDRESS.
         15 RESIDENCE-NUMBER PIC X(5).
         15 STREET-NAME PIC X(10).
      10 CITY-NAME PIC X(8).
      10 STATE-NAME PIC X(2).
      10 ZIP-CODE PIC 9(5).
```

These data fields do not have PIC clauses because they have subordinate data fields.

```
01 INVENTORY-RECORD.
   05 PART-NO PIC X(5).
┌─►05 PART-MANUFACTURER.
│     10 COUNTRY-CODE PIC X(3).
│     10 MANUFACTURER-CODE PIC X(5).
│  05 UNIT-PRICE PIC 9(5)V99.
│  05 QUANTITY-IN-STOCK PIC 9(5).
```

This data field does not have the PIC clause because it has subordinate data fields.

It is important to know the characters that are allowed in each type of PIC clause. The following table lists these valid characters. Spaces are valid characters in COBOL; they occupy computer memory just like other characters. In this respect, spaces are treated by COBOL differently than we would view them in the English language. Valid Characters for each type of `picture` Clause are shown in Table III.

A numeric data field normally contains only digits 0 through 9. If the numeric field is input data, it cannot contain anything else, not even a decimal point. In some systems, leading spaces are allowed. Thus, a data value of 10.50 cannot be directly stored

**Rules for Choosing PIC Clauses:**

1. A group data field must not have a PIC clause.
2. The word PIC can start in any column after the data field. However, for easy readability, the word PIC for all data fields should start in the same column.
3. There must be at least one blank space between the word PIC and the picture characters.
4. If a data field requires more than one picture character, the number of times the picture character appears can be enclosed in parentheses. Thus, PIC 9999 is the same as PIC 9(4).
5. Each group data field must end with a period. Each elementary data field must also end with a period after the picture characters.
6. It is important to understand the distinction between a numeric data value that can be contained in a data field with 9s for picture characters and in a data field with Ks as picture characters. Only a data field that specifies "9" for picture characters can contain numeric data that is to be used in a calculation.

**Table III** Valid Characters for Each Type of `PICTURE` Clause

| Data field type | Valid characters |
|---|---|
| Alphabetic | A through Z, a through z, spaces |
| Numeric | 0 through 9 |
| Alphanumeric | A though Z, a through z, 0 through 9, all special characters, spaces |

in an input numeric data field. How does one represent data values with decimal points in input data fields? The letter V is used to represent an implied decimal point in the PIC clause. For example, the `PIC` clause `PIC 99V99` represents a four-digit numeric data value with two digits to the left of the decimal point and two digits to the right of the decimal point. If the data value 1050 is stored in a data field having this PIC clause, the contents of the data field would represent the value 10.50. If the same data value is stored in a data field with the PIC clause `PIC 999V9,` it would represent *105.0*. The following examples illustrate how the letter V works in PIC clauses (see Table IV).

Though numeric data values with decimal points are stored in data fields without their decimal points, the computer keeps track of the decimal points during arithmetic calculations. The presence of implied decimal points does not require any computer memory space. Thus, a data field defined with the PIC `9(3)V99` requires five, not six, bytes of computer memory.

Review the following data fields.

```
PIC99V9    PIC999V99    PIC9V99    PICV99
  105        23450        105        10
   ↑           ↑           ↑          ↑
---------------------------------------------
   The Computer Assumes Decimals in
            these Positions
```

**Table IV** The Letter V in PIC Clauses

| Data value stored | Data field's PIC clause | Value contained |
|---|---|---|
| 10005 | PIC 999V99 | 100.05 |
| 9725 | PIC 9V999 | 9.725 |
| 124 | PIC 99V9 | 12.4 |
| 567 | PIC V999 | .567 |

## V. EDITING FEATURES OF COBOL

COBOL provides several features by which numeric output data are edited to make them more understandable and readable to humans. Among these features are leading zero suppression, comma insertion, dollar sign insertion, sign (+ or −) insertion, character insertion, and check protection.

### A. Leading Zero Suppression

Through the use of a special PIC clause, all leading zeros in output data can be suppressed. This PIC clause is represented by the letter `Z` (instead of 9). Table V explains the difference between `PIC 9` and `PIC Z` clauses.

### B. Comma/Dollar Sign Insertion

In output data, comma and dollar sign `($)` can be inserted so that 1000 looks like $1,000. This is accomplished by placing `$` and comma in the PIC clause of an output data field. This is illustrated in Table VI.

### C. Check Protection

COBOL supports the appearance of asterisks (*) in numeric output data—a feature that is common in checks. This feature prevents tampering with dollar amounts. A special PIC clause `(PIC *)` accomplishes this feature. This PIC clause can be used in combination with dollar sign and comma. Review Table VII.

## VI. COMPUTING IN COBOL

Calculations are performed in COBOL using several statements. Among them are `ADD, SUBTRACT, MULTIPLY, DIVIDE`, and `COMPUTE` statements. While the `ADD, SUBTRACT, MULTIPLY`, and `DIVIDE` statements perform primarily one arithmetic operation in one statement, a single `COMPUTE` statement performs several arithmetic operations.

Examples of these statements follow in Table VIII.

## VII. COBOL'S INTERFACE WITH SQL

COBOL can execute SQL statements to access relational databases. A COBOL compiler does not recognize SQL

**Table V** Differences in Output Data Using PIC 9 and PIC Z Clauses

| Data field | PIC clause | Value in data field | Output PIC clause | Output value |
|---|---|---|---|---|
| QUANTITY-IN-STOCK | PIC 9(4) | 0900 | PIC Z(4) | b900 |
| DEDUCT-AMOUNT | PIC 9(2)V99 | 5075 | PIC Z(3).99 | b50.75 |
| ACCT-BALANCE | PIC 9(3)V99 | 10000 | PIC ZZ9.99 | 100.00 |

statements. Therefore, a program that contains SQL statements must first pass through a precompiler, which translates those statements into standard COBOL code which a COBOL compiler can the process. The precompiler also verifies the validity and syntax of the SQL statements.

An assortment of coding requirements and options govern coding techniques for SQL in a COBOL program:

- Delimiting SQL statements
- Declaration of communications area for SQL, known as SQLCA
- Declaration of tables (optional)
- Declaration and usage of host variables and structures to pass data between the database management system (DBMS) and COBOL
- Retrieving multiple rows of data using a cursor

SQL statements must be contained within EXEC SQL and END-EXE delimiters, code as follows:

```
EXEC SQL
     SQL statement(s)
END-EXEC.
```

These keywords, which must be coded between Columns 12 and 72 (both inclusive), alert the precompiler to the presence of one or more SQL statements. A comment may be coded within the statements by placing an asterisk in column 7. The period on the END-EXEC clause is optional, but a COBOL warning may be used if it is missing. However, the END-EXEC clause should carry no period when SQL statements are embedded within an IF . . . THEN . . . ELSE set of statements to avoid ending the IF statement inadvertently.

All programs with embedded SQL statements need to communicate with the DBMS through the SQL communications area (SQLCA). The results of the DBMA operation are placed in the SQLCA variables. The most important of theses, the return code, is placed in the SQLCODE field. It should be tested to examine the success, failure, or exception condition on the operation, as shown in the following code:

```
EXEC SQL
     SELECT LSTMNS, DEPT
            FROM TEMPL
            WHERE EMPID = ‘123456’
END-EXEC
IF SQL CODE NOT EQUAL TO 0
       process error
```

An abbreviated list of common SQL return codes appears in Table IX.

The SQLCA is a data structure that should be copied into the program's WORKING-STORAGE SECTION through the SQL INCLUDE statement, as shown below:

```
          EXEC SQL
               INCLUDE SQLCA
          END-EXEC
Structure of SQLCA
01        SQLCA
          05 SQLCAID      PIC X(08)
          05 SQLCABC      PIC S9(09) USAGE
                                           COMP
          05 SQLCODE      PIC S9(09) USAGE
                                           COMP
```

**Table VI** Dollar Sign and Comma Insertion in Output Data Fields

| Data field | PIC clause | Value in data field | Output PIC clause | Output value |
|---|---|---|---|---|
| NET-SALARY | PIC 9(4)V99 | 123456 | PIC $Z,ZZZ.99 | $1,234.56 |
| UNIT-PRICE | PIC 9V99 | 345 | PIC $Z,ZZZ.99 | $bbbb3.45 |
| INSURANCE-FEE | PIC 9(2)V99 | 1234 | PIC $Z,ZZZ.99 | $bbb12.34 |

**Table VII** Check Protection Feature of COBOL

| Sending data field | | Output data field | |
|---|---|---|---|
| PIC clause | Data value | PIC clause | Output data |
| `9(4)V99` | 123456 | `$*,***.99` | `$1,234.56` |
| `9(4)V99` | 12345 | `$*,***.99` | `$**123.45` |
| `9(4)V99` | 1234 | `$*,***.99` | `$***12.34` |
| `9(6)V99` | 123456 | `$**,***.99` | `$*1,234.56` |

```
05 SQLERRM
   10 SQLERRML   PIC S9(04)
                     USAGE COMP
   10 SQLERRMC   PIC X(70)
05 SQLERRP       PIC X(08)
05 SQLERRD       OCCURS 6 TIMES
                 PIC S(09)
                     USAGE COMP
   10 SQLWARN0   PIC X(01)
   10 SQLWARN1   PIC X(01)
   10 SQLWARN2   PIC X(01)
   10 SQLWARN3   PIC X(01)
   10 SQLWARN4   PIC X(01)
   10 SQLWARN5   PIC X(01)
   10 SQLWARN6   PIC X(01)
   10 SQLWARN7   PIC X(08)
05 SQLEXT
```

This statement resembles the COPY statement in that the SQL INCLUDE copies portions of code stored in a disk library into the program. The INCLUDE statement, however, copies at precompile time rather than at compile time.

## VII. FUTURE OF COBOL

At one time, COBOL was the most widely used programming language for business applications. Introduction of newer languages has somewhat adversely affected the popularity of COBOL. Although new business applications are still being written in COBOL, the introduction of newer languages such as Visual Basic and Visual C++ has reduced COBOL's use. It remains to be seen how object-oriented features introduced in COBOL will impact COBOL.

**Table VIII** COBOL Calculation Statements

| | |
|---|---|
| `ADD data-field-a TO data-field-B` | Adds contents of data-field-a and data-field-b and stores the results in data-field-b |
| `ADD data-field-a, data-field-b GIVING data-field-c` | Adds contents of data-field-a and data-field-b and stores the result in data-field-c |
| `SUBTRACT data-field-a FROM data-field-b` | Subtracts contents of data-field-a from data-field-b and stores the result in data-field-b |
| `SUBTRACT data-field-a FROM data-field-b GIVING data-field-c` | Subtracts contents of data-field-a from data-field-b and stores the result in data-field-c |
| `MULTIPLY data-field-a BY data-field-b` | Multiplies contents of data-field-a and data-field-b and stores the result in data-field-b |
| `DIVIDE data-field-a BY data-field-b` | Divides contents of data-field-a by contents of data-field-b and stores the result in data-field-a |
| `DIVIDE data-field-a INTO data-field-b` | Divides contents of data-field-b by contents of data-field-a and stores the result in data-field-b |
| `COMPUTE data-field-a = data-field-b + data-field-c * data-field-d / data-field-e` | In a COMPUTE statement, arithmetic operations are performed from left to right with the following priority order:<br>1. Exponentiation<br>2. Multiplication/division<br>3. Addition/subtraction |

**Table IX** Common SQL Return Codes

| SQL code | Explanation |
|---|---|
| <0 | Warning |
| 0 | Successful execution |
| +100 | Row not found for `FETCH, UPDATE,` or `DELETE,` or the result of a query is an empty table |
| +802 | Data exception |
| −007 | Statement contains illegal character |
| −101 | Statement too long |
| −103 | Invalid numeric literal |
| −105 | Invalid string |
| −117 | Number of insert values not the same as number of object columns |
| −803 | Inserted or updated value invalid due to duplicate key |
| >0 | Error |

## SEE ALSO THE FOLLOWING ARTICLES

C and C++ • Fortran • Java • Pascal • Perl • Programming Languages Classification • Simulation Languages • Visual Basic • XML

## BIBLIOGRAPHY

Arnett, K. P., and Jones, M. C. (Summer 1993). Programming languages today and tomorrow. *The Journal of Computer Information Systems,* 77–81.

Bauman, B. M., Pierson, J. K., and Forchit, K. A. (Fall 1991). Business programming language preferences in the 1990's. *The Journal of Computer Information Systems,* 13–16.

Borck, J. R. (August 2000). Cobol programmers face it: Your favorite code is outdated and losing support. *Infoworld,* Vol. 23, Issue 2, 56.

Borck, J. R. (August 2000). Application transformation. *Infoworld,* Vol. 22, Issue 33, 53.

Borck, J. R. (August 2000). PERCobol whips Cobol into shape for budget-minded enterprises. *Infoworld,* Vol. 22, Issue 33, 54.

Buckler, G. (1990). Languages, methods all part of a COBOL evolution. *Computing Canada,* Vol. 16, Issue 13, 31–34.

Coffee, P. (August 2000). Cobol comes to the party. *eweek,* Vol.17, Issue 33, 53.

Garfunkel, J. (March 1992). Trends in COBOL. *Enterprise Systems Journal,* 120–122.

Garfunkel, J. (July 1990). COBOL—The next stage. *Computerworld,* Vol. 24, Issue 30, 31–34.

Khan, M. B. (1996). Structured COBOL: First course. Danvers, MA: Boyd & Fraser Publishing Company.

Lauer, J., and Graf, D. (Spring 1994). COBOL: Icon of the past or the symbol of the future? *The Journal of Computer Information Systems,* 67–70.

McMullen, J. (May 1991). Why PC COBOL is gaining ground. *Datamation,* Vol. 37, Issue 10, 70–72.

# Cohesion, Coupling, and Abstraction

**Dale Shaffer**
*Lander University*

## GLOSSARY

**abstraction** A technique that allows one to focus on a portion of an information system while ignoring other features of the system.

**abstraction barrier** A barrier around the implementation of a portion of an information system that is designed so that all access to the implementation is through well-defined avenues of access known as the interface.

**class** A set of objects with similar characteristics and behavior. It identifies the structure of the characteristics and behaviors that are exhibited by all of its instances.

**encapsulation** Binding a collection of data elements and modules that operate on those data elements into one package which can be invoked using one name.

**functional independence** The result from making modules that were of a single purpose and that avoided interaction with other modules.

**implementation** The portion of an information system located below the abstraction barrier that contains the programming code that implements specified behaviors.

**information hiding** The process of hiding detail from the user of a module or object; the information that is hidden is located below the abstraction barrier.

**interface** The bridge across an abstraction barrier that carefully controls access to the implementation portion.

**module** Any collection of programming statements that are combined together to perform a specified task. A module can also contain data, but the data is generally not maintained when program control passes from the module. Examples include a function, procedure, subroutine, subprogram, and method.

**object** An encapsulation of data and the methods that operate on the data. Access to the data and methods is strictly controlled through message-passing and inheritance. An object is an instance of a class.

**structured design** A set of techniques and strategies that are used to design modules that will best solve a problem.

**COMPUTER PROGRAMS** are usually constructed from building blocks, most of which can be categorized as modules and objects. Each module should perform a specific process in a program, and each object should perform one or more specific processes on a specific set of data. Cohesion is a measure of the functional strength, and coupling is a measure of the independence, of a module or object. A computer program with modules and objects that exhibits a high degree of cohesion and a low degree of coupling is considered to be well designed.

While cohesion and coupling are metrics used in software design, abstraction is a technique for building software systems. By employing abstraction, a software developer can focus on one portion of an information system at a time.

## I. HISTORICAL PERSPECTIVE

Early computer programs followed computer architecture, with data in one block of memory and program statements in another. With larger systems and recognition of the major role of maintenance, the block of program statements was further broken down into modules, which could be developed independently. Research in cohesion and coupling has its roots in the early 1970s as part of the development of modules. Structured design formalized the process of creating modules, recognizing that better written modules were self-contained and independent of each other. This functional independence was achieved by making modules that were of a single purpose, avoided interaction with other modules, and hides implementation details.

Consider the modules in Fig. 1. The combinations module calculates the number of n things taken r at a time. For example, consider a visit to a restaurant that had a salad bar where you were allowed to choose any three of the six meat and vegetable selections to include on your lettuce salad. The combinations module would determine that there were twenty different ways you could select three of the six items.

Functional independence is shown by the factorial module. The factorial module does only one task; it returns the factorial of the value given. It also has minimal interaction with other modules. The calling module, combinations, sends the minimum information that factorial needs—one number. Functional independence is measured using two criteria, cohesion and coupling.

## II. COHESION

First consider cohesion from outside of information systems. Two teams, each aligned along opposite ends of a rope, are pulling against each other. Each member of a team shares a common goal—to pull the knot at the midpoint of the rope across a specified finish line. Some members of the team have dug in their heels for better traction, and the one on the end has the rope tied around himself. The team has a high degree of cohesion. However, if one team member quits, or even begins to pull in the opposite direction, the high degree of cohesion is lost.

Much like the cohesive nature of the team, cohesion in information systems is the cement that holds a module together. Cohesion of a module is a measure of the relationship between its parts. If the cohesion is high, then the module is more likely to be designed well.

Cohesion in a module is a measure of the closeness of the data and programming statements, or more specifically the tasks performed, within the module. The higher the degree of cohesion that exists within a module, the less likely it is that the programmer using the module will have to become knowledgeable about what the module does. The programmer using a module with high cohesion can generally focus on what a module does rather than how it does it.

The encapsulation that cohesion promotes also aids in maintenance of the entire software system. For example, if a program contains modules strictly related to accessing and updating an inventory, and one module within it performs a particular search function, it would be quite easy to implement a new and improved search algorithm for the module. After thoroughly testing the new module, it would simply replace the old module in the software system. If done correctly, the replacement would be virtually unnoticeable by the user (except, for example, improved search time).

A module with high cohesion is generally better designed and represents a more general approach than one with low cohesion.

```
int factorial (int num) {
    int
        result = 1,
        counter;
    for (counter = 1; counter <= num; counter++)
        result= result * counter;
    return result;
}

int combinations (int n, int r) {
    return (factorial (n) / (factorial (r) * factorial (n-r)));
}
```

**Figure 1** Example of functional independence.

## A. Cohesion in Objects

Cohesion in objects is a measure of how well methods within an object work together. Objects whose methods work together to achieve one well-defined purpose are highly cohesive. Each method provides functionality that allows the data in the object to be either modified, used, or inspected.

If an object contains a series of unrelated methods, input and several trigonometric methods, for example, then the object has low cohesion since input and the trigonometric sine have very little in common. If an object contains a series of methods that implement operations on a table, then the object is said to have high cohesion.

## B. Levels of Cohesion

High cohesion is desirable in that it encapsulates a series of tasks into one cohesive unit. This encapsulation allows for easy application of the module when building software systems. For example, if one were building a software system to maintain an inventory, one could select only the inventory object and ignore a trigonometric object. However, if a system is being built to support the activities of a surveyor, then the trigonometric object could quickly be selected, and one could count on the object to have only trigonometric capabilities.

Cohesion can be examined at more levels than simply the arbitrary high and low. In 1979 Yourdon and Constantine specified seven levels of cohesion. The initial levels generally result in a poor module or object. Subsequent levels result in an increasingly acceptable module. Table I summarizes these levels of cohesion. Functional cohesion is the strongest, and coincidental is the weakest.

**Table I** Levels of Cohesion

| Level | Relationship between tasks within a module |
|---|---|
| Coincidental | None |
| Logical | Related |
| Temporal | Processed at the same time |
| Procedural | Established order of execution |
| Communicational | Operates on the same set of data |
| Sequential | Output from one serves as input to another |
| Functional | A module does one task |

The levels of cohesion will be discussed below in order based on increasing strength. Much of the discussion will be from the module standpoint. Some discussion of object cohesion will be made as well.

### 1. Coincidental Cohesion

Coincidental cohesion is the lowest level of cohesion. It occurs when there is little or no reasonable relationship among the tasks within an object or module. Although it occurs infrequently, it does serve as a zero point on the cohesion scale.

Coincidental cohesion can occur in objects when the object consists of unrelated methods. This can happen when a programmer compiles a set of favorite methods in an object. Other than being the programmer's favorites, the collection has nothing in common with each other. For example, the object could contain a method for rounding numbers, identifying a pattern in a character string, and inputting integers.

### 2. Logical Cohesion

When a collection of related methods are bound together into a single object, logical cohesion is present. When the object is called, one or more of the tasks are selected.

Consider, for example, an error handling object that, depending on the error, does one or more of the following: terminate execution, issues a warning, display a box describing the error and ask whether to proceed, adds the error to the error log, and offers a menu of selections for further processing. When an error is first determined, an object is instantiated and the error is processed by `handleError` (Fig. 2). If the error is not severe, a warning is issued and normal processing continues. If it is more severe, up to three actions could occur: an error message is displayed, the error log is updated, and/or the program is terminated. Severity levels are utilized to determine when these two actions occur.

Each of the methods within the class in Fig. 2 is important to handling errors, but the relationship between them ends there. They are simply related in that, depending on the error, various combinations of methods are executed.

In a similar way, a module that controls a robot arm could demonstrate logical cohesion if it contained a set of tasks that support each movement available to the arm. The movements could include rotation of the wrist and movement at the elbow and shoulder joints. A call to the module would invoke

```
public class ErrorHandler {

  public void displayWarning ();

  public void terminateExecution ();

  public void addToErrorLog ();

  public void whatNext ();
    // user choice: terminate or continue processing

  public void handleError ();
    // error handling routine

private
  file errorLog;
  int
      errorCode,
      logThreshold,  // level at which an error is logged
      endThreshold;
             // level at which an error ends processing

}
```

**Figure 2** Example of logical cohesion.

one or more of the individual tasks to move the arm to a specified location and position.

## 3. Temporal Cohesion

Temporal cohesion between modules transpires when the modules are placed together so that they can be processed within a limited time period. The order of the tasks is not important.

Figure 3 is an example of a set of startup routines for a personal computer. Since they all occur within a specified time period, they are temporally cohesive.

Temporal cohesion is slightly better than logical cohesion. The modules in Fig. 3 are logically cohesive in that they are performed in association with the startup process. What makes them temporally cohesive is that they are related by time.

## 4. Procedural Cohesion

Procedural cohesion occurs when the module contains tasks that are placed together on the basis of the algorithmic or procedural relationships. The relationship exists because the tasks must occur in a specific order, within a repetitive control structure (a loop), or, as in Fig. 4, as part of a selective control structure.

Procedural cohesion is one step above temporal cohesion. Not only are the tasks within the module connected by time, but they must also be executed in a specified order. Note that in Fig. 4, it is important to activate the virus protection software prior to connecting to the network and that network services can only be established if the network connection was successfully made.

## 5. Communicational Cohesion

The previous levels of cohesion were not necessarily tied to the problem structure. The individual tasks were somewhat related, but the relationship was lim-

```
void userStartup () {
    startEmail ();
    startWordProcessor ();
    startSpreadsheet ();
}
```

**Figure 3** Example of temporal cohesion.

```
void doAtStartup () {
    startVirusChecker();
    connectToNetwork ();
    if (successfulConnection)
        establishNetworkServices ();
else {
    startup.errorHandler (networkFailure);
    startupStatus.network (localUseOnly);
    }
    userStartup ();
}
```

**Figure 4** Example of procedural cohesion.

```
public class Inventory {

   public void addNewItem (Item element);

   public void removeItem(int stockNumber);

   public void updateItem (Item element);

   public Item findItem(int stockNumber);

   public void sellItem (int stockNumber, int quantity);

   public void addToStock (int stockNumber, int quantity);

}
```

**Figure 5** Example of communicational cohesion.

ited to, for example, the sequencing or the timing of the tasks.

Communicational cohesion occurs within a module or class whose tasks are centered on the data. All of the tasks either create, amend, report, or return portions of the data. It typically occurs when software is developed from the standpoint of the data.

An example of communicational cohesion is shown in Fig. 5. Note that every method in the Inventory class performs some operation on the inventory. `addNewItem` creates data, `removeItem` destroys data, `findItem` returns data, and all of the remaining methods modify the data.

## 6. Sequential Cohesion

Within a module, sequential cohesion occurs when the output from one task serves as input to another task. In other words, the operations performed on a set of data are executed in a specific order.

Prior to beginning a game of poker, Fig. 6 shows a function that could be called to set up the game. Note that all of the tasks that are within `setUpCardGame` not only are performed in the specified sequence, but the resultant data from each task is used in the next task.

## 7. Functional Cohesion

When each task in a module is needed for the execution of a single function, the module is said to be functionally cohesive. Moreover, the module in question is not cohesive at any of the previous levels. In other words, a module that is functionally cohesive is not comprised of tasks that are related only by sequential or weaker levels of cohesion.

This description of functional cohesion is less than ideal. Some examples, however, should help to delineate the difference between functional and lower levels of cohesion.

The Inventory class in Fig. 5 is an example of communicational cohesion. However, the `updateItem` module within Inventory is functionally cohesive because it does precisely one task—updating an inventory item.

Another example to consider is an automated bit changer for an industrial drill (Fig. 7). Note that all tasks contribute directly to the overall goal of changing the drill bit. In that sense, one might consider that it is an example of functional cohesion. However, the definition specifies that the module is not cohesive at another level. The module in Fig. 7 is an example of procedural cohesion.

```
void setUpCardGame ( int totalCards, int dealEachPlayer, int
numberPlayers) {
  Cards deck, player1, player2;
  deck.createDeck (totalCards);
  deck.shuffle ();
  deck.cut ();
  player1.deal (deck, dealEachPlayer);
  player2.deal (deck, dealEachPlayer);
}
```

**Figure 6** Example of sequential cohesion.

```
boolean changeBit (int newSize) {
 int oldSize= getCurrentSize();
 rotateHolder (oldSize);
 removeBit ();
 storeBit ();
 rotateHolder (newSize);
 retrieveBit ();
 mountBit ();
}
```

**Figure 7** Example of cohesion that is not functional.

A third example of functional cohesion was shown in Fig. 1. The factorial module is functionally cohesive in that it does only one task—it determines the factorial of the number provided. No superfluous actions occur; only the processing absolutely necessary for determining the factorial takes place.

However, if the factorial module were amended so that it can find the sum as well as the product from 1 to the provided number, then the module would no longer be functionally cohesive. Figure 8 shows such a module. Note that when it is called by `sumOrProduct` (4, false) it returns the value 24, which is equivalent to the factorial module. However, when it is called by `sumOrProduct` (4, true), it returns the sum of the integers from 1 to the value provided, returning the value 10. The `sumOrProduct` module in Fig. 7 is clearly not functionally cohesive.

## C. Levels of Cohesion in Objects

Many of the levels of cohesion can be viewed from the perspective of objects. For example, an object that contains methods that perform trigonometric operations can be considered to be logically cohesive. The class shown in Fig. 5 is an example of communicational cohesion.

The most desirable form of cohesion is functional cohesion, where all methods within an object work together to provide some well-bounded behavior. However, often the cohesion at the object level is at the communicational level. But functional cohesion should always be the goal within a method.

## D. Cohesion Levels in Perspective

Within a software system, cohesion can be examined from different perspectives. Cohesion can be measured in a module and in an object.

Within a module or method, the level of cohesion should be at the functional level. If possible, a module should accomplish exactly one task. If a module is found to be at a lower level, rewriting the module to raise it to a higher level should be considered.

The potential weakness of a module at a level below functional cohesion is increased difficulty in modifying the program. For example, if Fig. 8 were intended to find the factorial or the sum of the integers from 0 to the provided value, a programmer attempting to determine why the sum from 0 to 0 is 1 might amend the module as shown in Figure 9. Note that by initializing result to 0 does indeed correct the problem. However, the module now produces 0 for any factorial application.

Modules that are not written at a functionally cohesive level are not necessarily poor modules. Figure 7 shows a module that is an example of procedural cohesion. Nonetheless, it is a good module in that it does only one task—changing the bit on an industrial drill. It did not meet the strict definition of a functionally cohesive module, but is still a well-written module.

Objects are better written if they are at least communicationally cohesive. Communicational cohesion is often found in objects because all of the methods in the object operate on the common data.

However, objects could be logically cohesive and represent good design. For example, objects that contain only methods that enable trigonometric functions are logically cohesive.

```
int sumOrProduct (int num, boolean sum) {
  int
    result = 1,
    counter;
  for (counter = 1; counter <= num; counter++)
    if (sum)
      result= result + counter;
    else
      result= result * counter;
  return result;
}
```

**Figure 8** A module with two tasks.

```
int sumOrProduct (int num, boolean sum) {
 int
   result = 0,
   counter;
 for (counter = 1; counter <= num; counter++)
   if (sum)
     result= result + counter;
 else
   result= result * counter;
 return result;
}
```

**Figure 9** Modification difficulty at lower levels of cohesion.

When one designs modules and objects, the goal is to be aware of issues relating to cohesion and to strive for as high a level as reasonable. The goal of functional cohesion may not always be realized, yet a well-written module or object could still be written.

## III. COUPLING

The second of the two metrics for modules is coupling. Coupling is the measure of the strength of the interconnections between modules. It is a measure of the degree to which individual modules are tied together. Coupling is generally dependent on the complexity of the interface of a module. In other words, coupling depends on the data passed to the module.

Modules that are highly coupled are joined by many interconnections, loosely coupled modules are joined by weak interconnections, and uncoupled modules are not connected in any manner. Loosely coupled or uncoupled modules are considered best.

High coupling complicates a software system by making it harder to understand, more difficult to change, and more challenging to find errors. The more highly coupled a module is to other modules, the more likely it will be that a programmer who is revising the module will have to examine and possibly modify additional modules. Therefore, software that is written with the goal of avoiding high coupling will generally be less expensive to maintain.

### A. Examples of Coupling

Figure 10 shows an example of a highly coupled module. Note that when designing the combinations module, in order to use the factorial module one must provide a second parameter that is set to 1. Contrast this requirement to Fig. 1 where the result was a local variable initialized to 1. In both cases, the result is initialized to 1 and eventually holds the final answer.

This additional parameter unnecessarily increases the coupling between the combinations module and the factorial module. Notice that in Fig. 1 the additional parameter is omitted and the coupling between the modules is reduced. This reduced coupling makes it easier to use the factorial module.

Consider another example of coupling. A programming team is designing a large program to maintain airline reservations for customers in America. One of the modules manages the customer information, including the customer's address and postal code. The postal code is stored as a character string. Without consulting the programming team, a programmer decides to verify the city and state fields within the customer's record by comparing it against the five digit postal code. The module looks up the postal code in a table, determines the corresponding city, then updates the city field if it is incorrect. Knowing that not all postal codes had been assigned at the time he developed the table, the programmer decided that if the postal code was not found, no changes would be made.

The change in specifications seemed to be a good idea to the programmer. Unfortunately, the change was never reviewed by the programming team, and was never included in the documentation accompanying the program.

A few years later the airline reservations program is expanded to include a web interface for customers to make reservations. After the interface is implemented successfully in America, other countries are brought on-line. The United Kingdom is added first, and with postal codes like PL20 7SN, none of the postal codes are found by the module written several years ago, so no changes to the city and state were made. The successful implementation of the additions to the original program gave the programming team a stronger trust in the original program.

Eventually Australia was brought on-line. Postal codes in Australia are four digits, and when they were compared to postal codes in the table, almost every

```
int factorial (int num, int result) {
    int
        counter;
    for (counter = 1; counter <= num; counter++)
        result= result * counter;
    return result;
}
int combinations (int n, int r) {
    return (factorial (n, 1) / (factorial (r, 1) * factorial (n-r, 1)));
}
```

**Figure 10** Example of a highly coupled module.

address for customers in Australia were converted to cities and states in America. When complaints about lost tickets began flowing into the operations center, the staff tried to regroup and manage the mailing of tickets manually while programmers went on a search for the problem. The difficulty in locating the problem was compounded by the additional code that was recently added to the system.

The problem began when the programmer connected the module to a table. This additional connection increased the degree of coupling. If the programming team were aware of the additional coupling and agreed to it, the existence of the table would have been documented and, when other countries were included, been given consideration during the development of those additions.

This fictitious situation demonstrates the need to carefully control coupling. It shows that the goal is not simply to reduce coupling, but to carefully consider if the additional coupling is needed.

## B. Kinds of Coupling

Coupling can be considered at five levels, ordered from least interconnections to the most connections: uncoupled, parameter, control, common, and content. The forms of coupling are covered in turn.

### 1. Uncoupled

Uncoupled is the state where two modules have absolutely no interconnections between them. In other words, none of the forms of coupling below are present.

### 2. Parameter Coupling

The most typical and benign coupling that occurs is parameter coupling. Values are passed between modules via the parameter list. Short of uncoupled modules, parameter coupling is the lowest level of coupling. It is also known as data coupling.

### 3. Control Coupling

If several modules exchange control information, then the modules are connected via control coupling. For example, Fig. 11 accepts an integer representing an error level and takes predetermined actions based on the value. The parameter `errorLevel` is the avenue for control coupling.

```
void errorResponse (int errorLevel) {
    if (errorLevel > 100)
        terminateExecution();
        else if (errorLevel > 80) {
            addToErrorLog (severeError);
            issueUserWarning(severeError);
    } else if (errorLevel > 50)
        issueUserWarning (mildError);
}
```

**Figure 11** Example of control coupling.

### 4. Common Coupling

Common coupling occurs when a group of modules all access the same global data. It is also known as common-environment and global coupling.

Consider a group of modules that all perform actions associated with a file. One module might sort the file, another search, a third update. Since they all perform operations on the same set of data, this forms an example of common coupling.

The down side of common coupling is that diagnosing problems can be time consuming. This does not mean that common coupling is wrong. It simply means that it should be avoided. If, as in the above example, it cannot be avoided, then the software developer must recognize that common coupling exists and allow time to identify techniques to independently test the modules. The developer could, for example, test each of the modules above separately. It should also be recognized that there could be connections between the modules. For example, the search module might only be effective on sorted data.

### 5. Content Coupling

Content coupling has the highest degree of coupling. It comes about when one module makes use of data or program statements that are strictly inside another module. It is also known as pathological coupling and internal data coupling.

Consider again the sort and search modules from the previous example, and assume that the search module only operates on sorted data. If the search module uses a goto statement to jump to the code in the sorting module to sort the data prior to searching it, it would be an example of content coupling.

Contrast this with the approach of calling the sort module from within the search module. This approach would lower coupling to the common level, i.e., sharing common data.

### C. Other Types of Coupling

There are other types of coupling evident in a software system. They are unranked, and are unrelated to the previously mentioned kinds of coupling.

External coupling occurs when a module communicates directly with a specific hardware device. For example, a module that accepts input from a mouse is very closely coupled with the mouse. The same module could not, for example, be used for input from a keyboard. This type of coupling, while entirely necessary, should be limited to specific modules that are dedicated to communicating to the external device.

Binding coupling occurs when names are prematurely bound to values in a program. For example, if the programmer places the value 0.05 throughout the program to represent the sales tax rate of 5%, the program is tightly coupled to the sales tax rate. Introducing a constant in the program and setting it to 0.05 can decrease binding coupling. The use of the constant allows future modification of the program to be simplified when it becomes necessary to change the sales tax to 6%.

### D. Coupling in Objects

Coupling between classes occurs in two ways—message passing and inheritance. Message passing serves as the primary means of communication between classes, and should be carefully minimized to achieve low coupling.

Coupling can also occur through inheritance. Inheritance has good qualities in that it exploits the commonality between classes. However, inheritance can increase coupling by closely connecting a superclass to its subclass. Common coupling, via the data, can be the result of inheritance.

Classes typically have a mechanism for controlling coupling. In some languages, data and methods are identified as belonging to one of three classifications—public, protected, and private. Private data and methods are not available for use outside the class where they are located.

Methods that are designated as public are available globally. Any programming component, be it an object of a different class or a module that is not part of a class, can access a method that is designated as public by way of message passing. Data can be designated as public, but is usually not done as it can defeat the purpose of objects.

Protected data and methods are not available globally. They are, however, available within the class and to any classes that inherit from the class where they are located.

## IV. COHESION AND COUPLING IN PERSPECTIVE

Cohesion and coupling form the central tenets of structured design. Together, these principles contribute to designing software components that will help solve a problem in a robust, reliable, and efficient manner.

When considering cohesion and coupling, the goal is not necessarily to use only functional cohesion and parameter coupling. However, other things being equal, the higher the level of cohesion and the lower the level of coupling, the more likely is it that software will be clear to understand and thus easily modified and maintained.

## V. ABSTRACTION

Abstraction is a technique that helps in understanding complex systems. It allows one to focus on a portion of the system while ignoring other aspects.

George Miller brought the term abstraction to the forefront in 1956. He proposed that humans were only able to handle between 5 and 9 pieces of information at one time, and that larger pieces of information were managed by forming abstractions, or breaking information into manageable pieces.

An early example of abstraction outside of information systems occurred during the industrial revolution. Early efforts at manufacturing required a craftsman to completely build, for example, one dresser by hand. One of the results of the industrial revolution was to break up the manufacturing process into distinct stages. Workers could specialize so that a drawer for a dresser was built in stages. The wood for the drawer was selected by one individual, cut and planed by another, built by a third, and finished by a fourth. If, for example, we focused on the individual applying the finish, we would be thinking abstractly of the other stages in the process.

Consider another example from industry. A company decides to manufacture a lawn mower. The decision was made to purchase engines rather than build them. A set of specifications was drawn up, including the following: 3 horsepower, vertical shaft, and 2400 revolutions per minute. Note that these specifications indicate what is needed, not how the engine operates. One of several items not included in these specifications was the diameter of the engine's piston. Abstraction allowed the company to focus on what was needed and allowed some items to be ignored.

Abstraction is a fundamental technique in the design of programs. It allows for the selective hiding of information to allow refocusing on a different aspect of the information system. The aspects of the information system that are important to the new perspective are brought to the forefront while other aspects are not directly considered.

Abstraction allows the designer to separate the goals of the software system from the actual details of implementing it. This separation of the "what" from the "how" is a key to abstraction. It allows the software designer to focus on the essential details of a programming component without concern for lower level details. The separation of what needs done from how it is actually implemented helps to break a complex software system into manageable parts.

The use of abstraction to break down complex problems into manageable pieces allows for a more easily managed software development process. Individual programmers can be assigned specific modules with less concern for side effects with the work of other programmers.

Abstraction is the basic tool of software libraries. A class written with abstraction in mind is more likely to be suitable for inclusion in a software library. The resultant library will tend to be more general and, therefore, applicable to other situations.

Abstraction can be approached from two perspectives when building a software system, from the bottom up or from the top down. First consider from the bottom up, where the initial focus is on the details and the goal is to form the details into a cohesive process. In other words, a complex set of details can be organized into a module, then dealt with as a single unit. For example, a series of steps are needed to produce the gelatin-like coatings on capsules. Factors like temperature and viscosity of the vat of gelatin are monitored while raw ingredients are added to bring the liquid to the proper consistency prior to pouring the molds for the capsules. The focus was initially on the individual steps, with careful attention given to ensuring each step was correct before combining it with other steps. Eventually the collection of steps involved in the preparation of the gelatin can be abstractly thought of as gelatin preparation.

The second perspective on abstraction is from the top down by dividing a large project into a series of components. For example, when building a program to maintain a database of paintings and related information for an art gallery, the task can be divided into routines to manage the paintings, maintain information on shows, and maintaining information on ownership of collections lent to the gallery. We can abstractly consider the information maintained on paintings until development of it is necessary.

## A. Levels of Abstraction

The idea of levels of abstraction is prevalent in the arts as well as the sciences. Henri Matisse produced a series of four bronze sculptures of a woman's back; each subsequent sculpture was more abstract, containing less detail. It is difficult to tell where the legs end and the back begins in the fourth rendering.

Levels of abstraction can also be illustrated with the automobile. We can think of the car simply as a means of transportation. One can turn on the key, put the car in gear, and go. At the next level of abstraction we can consider the car as a series of subsystems—the engine, the drive train, the electrical system, and so on. Next we can look closer at the engine, and consider the carburetor, the ignition system, and the combustion chamber. At an even lower level of abstraction, we can explore the operation of the combustion chamber, considering the four cycles of intake, compression, power, and exhaust. At an even lower level we can consider how the fuel is ignited by the spark plug to begin the power cycle. And at a lower level yet, we can discuss the qualities of petrol that allow it to ignite at low temperatures.

Abstraction is relative. At higher levels, one view is more general and more details are ignored. At lower levels the focus is more on details while less attention is given to the general view. When thinking of a car as a mode of transportation, we are ignoring many details. When considering the combustion process taking place in a car's cylinder, the center of attention is on the details of combustion and very little is given to the engine or the vehicle itself.

Abstraction in computer systems occurs at multiple levels as well. A user supplies data to and receives results from an application. A programmer wrote the information system using a high-level programming language. The compiler translates the high-level language into assembly language or to machine language that the computer can execute. At the microprogramming level, the control unit within the computer interprets the machine language instructions and activates gates to move information to the necessary locations. If more than one program is executing at one time, the operating system might temporarily suspend the execution of the machine language statements in one program while another program takes precedence. These levels of abstraction are summarized in Fig. 12.

Identification of levels of abstraction in the development of an information system is also evident. A solution is first stated in broad terms using the language of the problem environment. Next a modeling technique like Unified Modeling Language is used to express the problem and move a step closer to the solution.

As the developer moves closer to creating a solution in a high-level language, the specifications look more like classes and modules. It is in this final level of the development of software, the creation of the high-level language solution, that there are two kinds of abstraction—procedural and data.

## B. Kinds of Abstraction

There are two major kinds of abstraction in high-level languages. Both give the ability to group items together under one name.

Procedural abstraction provides the methodology that allows a sequence of actions that has a specific and limited function to be named. The actions are encapsulated under a single name that can be invoked to perform the sequence of actions. The name is descriptive of what the sequence of actions perform.

The resultant sequence of statements that is formed can be called a module, or a procedure, function, subroutine, subprogram, or method. A module, then, is an abstraction that describes the compound actions and is independent of other modules. This independence can be measured by the coupling metric. The statements within the module should be cohesive.

A data abstraction is a named collection of data that describes a data object. It isolates how a compound data object is used from the details of how it is constructed. For example, the data abstraction for dog would include data such as gender, height, breed, and age. Whenever one considers the information about the dog in aggregate, data abstraction is being used. Examples of constructs that support data abstraction are arrays, lists, records, and files.

| Level | Description |
|---|---|
| 1 | logic gates |
| 2 | microprogramming |
| 3 | machine language |
| 4 | operating system |
| 5 | assembly language |
| 6 | high-level language |
| 7 | application |

**Figure 12** Levels of abstraction in a computer system.

Procedural and data abstraction has been identified as being important to software development for years. The emergence of the object-oriented paradigm has given a new focus on applying them, however. Procedural abstraction is employed in the development of methods, and data abstraction is utilized in determining the data elements in an object. A higher level of abstraction is accomplished by specifying an interface to the object's data by describing a set of methods that build the data structure and provide access to it. By careful specification of the interface, an abstraction barrier is formed.

## C. Abstraction Barriers

An abstraction barrier can be compared to a fence that encloses a large flock of sheep. From outside the fence one can determine that sheep are indeed inside the fence. But it is less easy to determine specific information like quantity, gender, and the types of sheep inside the fence. If one saw the shepherd inside the fence and asked him questions about the sheep, specific attributes about the herd could be determined. If the fence is well built, it not only keeps out wolves. It prevents someone outside the fence from climbing over the fence and counting the sheep.

An abstraction barrier performs a similar function in an information system. A well-constructed abstraction barrier hides information about the internal structure, yet allows pre-determined and carefully controlled avenues of access to the data. It also hides from view modules that are not needed outside the abstraction barrier.

A well-written abstraction barrier helps in the development of large systems. For example, assume we have been assigned the task of developing software to maintain stock ownership for a stockbroker. The number of shares is maintained using mixed numbers. The data that represents mixed numbers and the operations that are needed for mixed numbers is shown in Fig. 13. Once the abstraction barrier is established, a programming team can better divide responsibilities. Several programmers can work below the abstraction barrier and implement the operations. Other programmers can work above the abstraction barrier to build the stock ownership system that uses mixed

```
Abstraction barrier:
    operations
        +, -, *, /, reduce
    data
        whole number
        numerator
        denominator
```

**Figure 13** Abstraction barrier definition in a stock ownership system.

numbers like 4 1/3. The programmers working below the abstraction barrier would be practicing information hiding by hiding selected portions of the information system from access above the abstraction barrier.

The object-oriented paradigm has a strong basis in abstraction barriers. The class specified in Fig. 14 shows the interface for a mixed number. The only information provided is what is needed for an individual to establish the value for a mixed number, return the mixed number, and perform specified operations on the mixed number. Hidden from the interface is the exact representation of the mixed number and how the operations are performed.

## D. Abstraction in Information Systems

Abstraction is important to the development of an information system. One of the early approaches to software development, stepwise refinement, has strong roots in abstraction. At the highest levels of stepwise refinement, the solution to an information system is stated in broad terms. As lower levels are defined, abstraction barriers are specified, more details are considered, and a more procedural orientation is taken. Stepwise refinement, then, moves from higher levels of abstractions to lower levels.

```
public class mixed {

    public mixed (int whole, numer, denom);

    public void add (mixed number);

    public void subtract (mixed number);

    public void multiply (mixed number);

    public void divide (mixed number);

    public void reduce ();

    public int returnWhole ();

    public int returnNumerator ();

    public int returnDenominator ();

}
```

**Figure 14** Abstraction barrier used with objects.

Like abstraction, abstraction barriers exist in many areas of information systems. For example, an abstraction barrier exists between the high-level code for a software system and the compiled code. The programmer building the software system generally does not think about the compilation process beyond the fact that it hopefully returns the compiled code. The developer of the compiler allowed for carefully controlled access to the compilation process.

# VI. CONCLUSION

In the early years of programming, abstraction was not central to the development process. Modules were not utilized, and data structures were rare. The result was generally a high rate of errors per line of code written, and therefore high software development costs.

Abstraction is so pervasive in information systems today that it is seldom considered in its own right. It is so fundamental that programmers are faced with abstraction on a daily basis, yet seldom consider it directly.

The change to the object-oriented paradigm has placed abstraction, if not in name then at least in concept, to the forefront. Well-designed classes have well-designed abstraction barriers.

Abstraction allows a proposed information system to be broken down into manageable parts to facilitate development. These parts can then be glued back together into a complete system. Cohesion measures how well the individual portions have been divided, while coupling measures the relationships between the parts.

# SEE ALSO THE FOLLOWING ARTICLES

Data Flow Diagrams • Documentation for Software and IS Development • Object-Oriented Programming

# BIBLIOGRAPHY

Abelson, H., Sussman, G. J., and Sussman, J. (1985). *Structure and interpretation of computer programs.* New York: McGraw-Hill.

Bieman, J. M., and Ott, L. M. (1994). Measuring functional cohesion. *IEEE Transactions on Software Engineering,* 20(8), 644–657.

Booch, G. (1994). *Object-oriented Analysis and Design,* second edition. Redwood City, CA: Benjamin/Cummings.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review,* 63, 81–97.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM,* 15(12), 1053–1058.

Pressman, R. (2001). *Software engineering,* 5th ed. New York: McGraw-Hill.

Stevens, W., Myers, G., and Constantine, L. (1974). Structured design. *IBM Systems Journal,* 13(2), 115–139.

Wirth, N. (1971). Program development by stepwise refinement. *Communications of the ACM,* 14(4), 221–227.

Yourdon, E., and Constantine, L. (1979). *Structured design.* Englewood Cliffs, NJ: Prentice-Hall.

# Compilers

**Seth D. Bergmann**
*Rowan University*

## GLOSSARY

**code generation** The phase of the compiler which produces machine language object code from syntax trees or atoms.

**compiler** A software translator which accepts, as input, a program written in a particular high-level language and produces, as output, an equivalent program in machine language for a particular machine.

**compiler-compiler** A program which accepts, as input, the specifications of a programming language and the specifications of a target machine, and produces, as output, a compiler for the specified language and machine.

**derivation** A sequence of applications of rewriting rules of a grammar, beginning with the starting nonterminal and ending with a string of terminal symbols.

**derivation tree** A tree showing a derivation for a context-free grammar, in which the interior nodes represent nonterminal symbols and the leaves represent terminal symbols.

**formal language** A language which can be defined by a precise specification.

**grammar** A language specification system consisting of a finite set of rewriting rules involving terminal and nonterminal symbols.

**high-level language** A programming language which permits operations, control structures, and data structures more complex than those available on a typical computer architecture.

**lex** A lexical analyzer generator utility in the Unix programming environment which uses regular expressions to define patterns.

**lexical analysis** The first phase of the compiler, in which words in the source program are converted to a sequence of tokens representing entities such as keywords, numeric constants, identifiers, operators, etc.

**object program** A program produced as the output of the compiler.

**optimization** The process of improving generated code in run time and/or space.

**parse** A description of the structure of a valid string in a formal language, or to find such a description.

**parsing algorithm** An algorithm which solves the parsing problem for a particular class of grammars.

**parsing problem** Given a grammar and an input string, determine whether the string is in the language of the grammar and, if so, find its structure (as in a derivation tree, for example).

**programming language** A language used to specify a sequence of operations to be performed by a computer.

**recursive descent** A top down parsing algorithm in which there is a procedure for each nonterminal symbol in the grammar.

**register allocation** The process of assigning a purpose to a particular register, or binding a register to a source program variable or compiler variable, so that for a certain range or scope of instructions that register can be used to store no other data.

**semantic analysis** That portion of the compiler which generates intermediate code and which attempts to find non-syntactic errors by checking types and declarations of identifiers.

**semantics** The intent, or meaning, of an input string.

**source language** The language in which programs may be written and used as input to a compiler.

**source program** A program in the source language, intended as input to a compiler.

**symbol table** A data structure used to store identifiers and possibly other lexical entities during compilation.

**syntax** The specification of correctly formed strings in a language, or the correctly formed programs of a programming language.

**syntax analysis** The phase of the compiler which checks for syntax errors in the source program, using, as input, tokens put out by the lexical phase and producing, as output, a stream of atoms or syntax trees.

**syntax directed translation** A translation in which a parser or syntax specification is used to specify output as well as syntax.

**syntax tree** A tree data structure showing the structure of a source program or statement, in which the leaves represent operands, and the internal nodes represent operations or control structures.

**target machine** The machine for which the output of a compiler is intended.

**token** The output of the lexical analyzer representing a single word in the source program.

**translation grammar** A grammar which specifies output for some or all input strings.

**yacc (yet another compiler-compiler)** A parser generator utility in the Unix programming environment which uses a grammar to specify syntax.

# I. INTRODUCTION

A *user interface* is the mechanism through which the user of a device communicates with the device. Since digital computers are programmed using a complex system of binary codes and memory addresses, computer scientists have developed sophisticated user interfaces, called programming languages, which enable people to specify computations in ways that seem more natural. This article describes the compiler, which is the software translator normally used to implement this kind of interface. The concepts presented here are valuable not only to those responsible for implementing programming languages, but also to all programmers, who will become better programmers as a result of understanding how programming languages are implemented, and will have a greater appreciation for programming languages. In addition, the techniques which are presented here can be used in the construction of other user interfaces, such as the query language for a database management system.

## A. What Is a Compiler?

The articles on computer hardware and software describe the kinds of instructions that the computer's CPU is capable of executing. In general, they are very simple, primitive operations, constituting a language which we call *machine language*. For example, there are often instructions which do the following kinds of operations: (1) add two numbers stored in memory; (2) move numbers from one location in memory to another; and (3) move information between the CPU and memory. But there is certainly no single instruction capable of computing an arbitrary expression such as $((x-x_0)^2 + (x-x_1)^2)^{1/2}$, and there is no way to do the following with a single instruction:

```
if (array6[loc]<MAX) sum
            = 0; else array6[loc] = 0;
```

These are some of the capabilities which are implemented with a software translator, known as a *compiler*. The function of the compiler is to accept statements such as those above and translate them into sequences of machine language operations which, if loaded into memory and executed, would carry out the intended computation. It is important to bear in mind that when processing a statement such as `x = x * 9;` the compiler does not perform the multiplication. The compiler generates, as output, a sequence of instructions, including a "multiply" instruction.

Languages which permit complex operations, such as the ones above, are called *high-level languages,* or *programming languages.* A compiler accepts as input a program written in a particular high-level language and produces as output an equivalent program in machine language for a particular machine called the *target* machine. We say that two programs are *equivalent* if they always produce the same output when given the same input. The input program is known as the *source program,* and its language is the *source language.* The output program is known as the *object program,* and its language is the *object language.* A compiler translates source language programs into equivalent object language programs. Some examples of compilers are

- A Pascal compiler for the Apple Macintosh
- A COBOL compiler for the VAX
- A C++ compiler for the Apple Macintosh

If a portion of the input to a C++ compiler looked like this:

```
A = B + C * D
```

the output corresponding to this input might look something like this:

```
LOD  R1,C      // Load the value of C
               // into reg 1
MUL  R1,D      // Multiply the value
               // of D by reg 1
STO  R1,TEMP1  // Store the result in
               // TEMP1
LOD  R1,B      // Load the value of B
               // into reg 1
ADD  R1,TEMP1  // Add value of Temp1
               // to register 1
STO  R1,TEMP2  // Store the result in
               // TEMP2
MOV  A,TEMP2   // Move TEMP2 to A,
               // the final result
```

The compiler must be smart enough to know that the multiplication should be done before the addition even though the addition is read first when scanning the input. The compiler must also be smart enough to know whether the input is a correctly formed program (this is called checking for proper *syntax*), and to issue helpful error messages if there are syntax errors.

Many compilers will not generate optimal, or efficient, code. In designing a compiler, the primary concern is that the object program be semantically equivalent to the source program (i.e., that they mean the same thing, or produce the same output for a given input). Object program efficiency is important, but not as important as correct code generation, which is of paramount importance.

What are the advantages of a high-level language over machine or assembly language?

1. Machine language (and even assembly language) is difficult to work with and difficult to maintain.
2. With a high-level language you have a much greater degree of machine independence and portability from one computing platform to another (as long as the other machine has a compiler for that language).
3. Application programmers need not be retrained every time a new machine (with a new instruction set) is introduced.
4. High-level languages may support data abstraction (through data structures and objects) and program abstraction (procedures and functions).

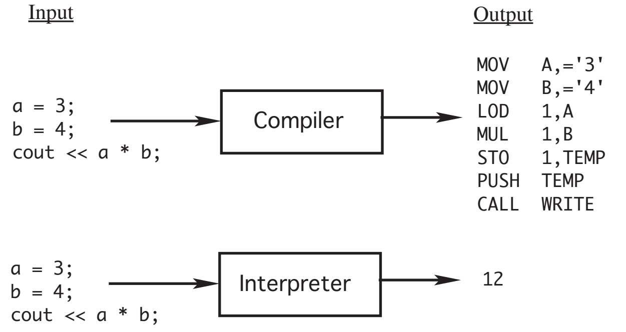
What are the disadvantages of high-level languages?

1. The programmer doesn't have complete control of the machine's resources (registers, interrupts, I/O buffers, etc.).
2. The compiler may generate inefficient machine language programs.
3. Additional software—the compiler—is needed in order to use a high-level language.

As compiler development and hardware have improved over the years, these disadvantages have become less problematic. Consequently, most programming today is done with high-level languages.

An *interpreter* is software which serves a purpose very similar to that of a compiler. The input to an interpreter is a program written in a high-level language, but rather than generating a machine language program, the interpreter actually carries out the computations specified in the source program. In other words, the output of a compiler is a program, whereas the output of an interpreter is the source program's intended output. Figure 1 shows that although the input may be identical, compilers and interpreters produce very different output. Nevertheless, many of the techniques used in designing compilers are also applicable to interpreters.

There is often confusion about the difference between a compiler and an interpreter. Many commercial compilers come packaged with a built-in edit-compile-run front end. In effect, one is not aware that after compilation is finished, the object program must be loaded into memory and executed, because this all happens automatically. As larger programs are needed to solve more complex problems, programs are divided into manageable source modules, each of which is compiled separately to an object module. The object modules can then be linked to form a single, complete, machine language program. In this mode, it is



```
a = 3;
b = 4;
cout << a * b;
```

```
MOV   A,='3'
MOV   B,='4'
LOD   1,A
MUL   1,B
STO   1,TEMP
PUSH  TEMP
CALL  WRITE
```

```
a = 3;
b = 4;
cout << a * b;
```

```
12
```

**Figure 1** A compiler and interpreter producing very different output for the same input.

more clear that there is a distinction between *compile time,* the time at which a source program is compiled, and *run time,* the time at which the resulting object program is loaded and executed. Syntax errors are reported by the compiler at compile time and are shown at the left, below, as compile-time errors. Other kinds of errors not generally detected by the compiler are called *run-time errors* and are shown at the right below:

**Compile-time errors Run-time errors**

```
a = ((b+c)-d;  x = a-a;
               y = 100/x;
               // division by 0
if x<b fn1();  ptr = NULL;
  else fn2();  data = ptr->info;
               // use of null pointer
```

It is important to remember that a compiler is a program, and it must be written in some language (machine, assembly, high-level). In describing this program, we are dealing with three languages: (1) the *source* language, i.e., the input to the compiler; (2) the *object* language, i.e., the output of the compiler; and (3) the language in which the compiler is written, or the language in which it exists, since it might have been translated into a language foreign to the one in which it was originally written. For example, it is possible to have a compiler that translates Pascal programs into Macintosh machine language. That compiler could have been written in the C language, and translated into Macintosh (or some other) machine language. Note that if the language in which the compiler exists is a machine language, it need not be the same as the object language. For example, a compiler that produces Macintosh machine language could run on a VAX. Also, the object language need not be a machine or assembly language, but could be a high-level language. A concise notation describing compilers is shown in Fig. 2. In these diagrams, the large C stands for compiler (not the C programming language), the superscript describes the intended translation of the compiler, and the subscript shows the language in which the compiler exists. Figure 2a shows a Pascal compiler for the Macintosh. Figure 2b shows a compiler which translates Pascal programs into equivalent Macintosh machine language, but it exists in VAX machine language, and consequently it will run only on a VAX. Figure 2c shows a compiler which translates IBM PC machine language programs into equivalent Pascal programs. It is written in Ada and will not run in that form on any machine.

In this notation the name of a machine represents the machine language for that machine; i.e., VAX represents VAX machine language, and PC represents PC machine language (i.e., Intel 80 × 86 or Pentium).

a $$C_{\text{Mac}}^{\text{Pas} \longrightarrow \text{Mac}}$$

b $$C_{\text{Vax}}^{\text{Pas} \longrightarrow \text{Mac}}$$

c $$C_{\text{Ada}}^{\text{PC} \longrightarrow \text{Pas}}$$

**Figure 2** Big C notation for compilers: (a) A Pascal compiler that runs on the Mac. (b) A Pascal compiler that generates Mac programs and runs on a VAX. (c) A compiler that translates PC programs into Pascal and is written in Ada.

Two opposing approaches have been used in the implementation of programming languages. The first approach was to view the compiler (or interpreter) simply as a means of accessing the capabilities of a particular CPU architecture., e.g., the addition of two integer quantities in a program implies that the CPU's integer addition instruction should be utilized (whether it be a 16-, 32-, or 36-bit, or other precision). This can result in software which is not always portable from one platform to another. The second approach, used in the specification of more recent languages such as Java, is to specify the exact semantics of all operations as part of the language. This results in more portable code.

It is safe to say that no other field of computer science makes as extensive use of theoretical results for use in an applied manner. The lexical and syntax phases of a compiler draw extensively from automata and formal language theory. The code generation and optimization phases make extensive use of graph theory.

## B. A Brief History of Compilers

The history of compilers is tied to the history of programming languages. The first compilers were primarily methods for including calls to previously written subprograms. At this time coding was done in machine language; these methods allowed the programmer to "compile" a collection of previously written code modules into one program. A few of the early such systems were A-0, A-2, and FLOW-MATIC, developed by Grace Murray Hopper at Harvard University in the early 1950s. Around the same time IT (Internal Translator) was developed at Carnegie Institute of Technology. These early systems were soon followed by compilers which were capable of translating algebraic expressions and which were closer to the current notion of a compiler as a language translator. The first such system was Formula Translator, or FORTRAN, developed by John Backus of IBM Corporation.

In the 1960s much progress was made in the areas of lexical and syntax analysis. It was at this time that compiler-generators (or compiler-compilers) were developed. These systems were capable of producing translators automatically from a formal specification of a language's syntax and semantics.

In the 1970s the focus shifted to the generation of efficient object code, as research produced better register allocation algorithms and optimization techniques. In the years since that time, new computer architectures and new programming language paradigms have made the art of compiler design a challenge to this day, with many open problems remaining to be solved.

## C. The Phases of a Compiler

It is important to remember that the input to a compiler is simply a string of characters. Some people assume that a particular interpretation is automatically "understood" by the computer (`count = count + 1;` is obviously an assignment operation, but the computer must be programmed to determine that this is the case).

In order to simplify the compiler design and construction process, the compiler is implemented in phases. In general, a compiler consists of at least three phases: (1) lexical analysis, (2) syntax analysis, and (3) code generation. In addition, there could be other optimization phases employed to produce efficient object programs.

### 1. The Lexical Phase

The first phase of a compiler is called *lexical analysis* (and is also known as a *lexical scanner*). As implied by its name, lexical analysis attempts to isolate the "words" in an input string. We use the word "word" in a technical sense. A word, also known as a *lexeme,* a lexical *item,* or a lexical *token,* is a string of input characters which is taken as a unit and passed on to the next phase of compilation. Examples of words are

1. *key words*—`while, void, if, for, ...`
2. *identifiers*—declared by the programmer
3. *operators*—+, −, *, /, =, ==, . . .
4. *numeric constants*—numbers such as `124, 12.35, 0.09E-23,` etc.
5. *character constants*—single characters or strings of characters enclosed in quotes.
6. *special characters*—characters used as delimiters such as `. ( ) , ; :`
7. *comments*—ignored by subsequent phases. These must be identified by the scanner, but are not included in the output.

The output of the lexical phase is a stream of tokens corresponding to the words described above. In addition, this phase builds tables which are used by subsequent phases of the compiler. One such table, called the *symbol table,* stores all identifiers used in the source program, including relevant information and attributes of the identifiers. In block-structured languages it may be preferable to construct the symbol table during the syntax analysis phase because program blocks (and identifier scopes) may be nested.

### 2. The Syntax Phase

The *syntax analysis phase* is often called the *parser.* This term is critical to understanding both this phase and the study of languages in general. The parser will check for proper syntax, issue appropriate error messages, and determine the underlying structure of the source program. The output of this phase may be a stream of *atoms* or a collection of *syntax trees.* An atom is an atomic operation, or one that is generally available with one (or just a few) machine language instruction(s) on most target machines. For example, `MULT, ADD,` and `MOVE` could represent atomic operations for multiplication, addition, and moving data in memory, respectively. Each operation could have 0 or more operands also listed in the atom (operation, operand1, operand2, operand3). The meaning of the following atom would be to add `A` and `B`, and store the result into `C`:

```
(ADD, A, B, C)
```

Some parsers put out syntax trees as an intermediate data structure, rather than atom strings. A syntax tree indicates the structure of the source statement, and object code can be generated directly from the syntax tree. In syntax trees, each interior node represents an operation or control structure and each leaf node represents an operand. A syntax tree for the statement

```
A = B + C * D;
```

is shown in Fig. 3.

Once a syntax tree has been created, it is not difficult to generate code from the syntax tree; a *postfix* traversal of the tree is all that is needed. In a postfix traversal, for each node, N, the algorithm visits all the subtrees of N, and visits the node N last, at which point the instruction(s) corresponding to node N can be generated.

Many compilers also include a phase for *semantic analysis.* In this phase the data types are checked, and

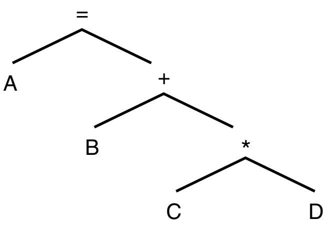


**Figure 3** A syntax tree for `A = B + C * D;`.

type conversions are performed when necessary. The compiler may also be able to detect some semantic errors, such as division by zero, or the use of a null pointer.

## 3. Global Optimization

The *global optimization* phase is optional. Its purpose is simply to make the object program more efficient in space and/or time. It involves examining the sequence of atoms put out by the parser to find redundant or unnecessary instructions or inefficient code. Since it is invoked before the code generator, this phase is often called machine-independent optimization. For example, in the following program segment:

```
for (i=1; i<=100000; i++)
   {  x = sqrt (y);   // square root
                      // function
      cout < x+i < endl;
   }
```

In this case, the assignment to `x` need not be inside the loop since `y` doesn't change as the loop repeats (it is a *loop invariant*). In the global optimization phase, the compiler would move the assignment to `x` out of the loop in the object program:

```
x = sqrt (y);       // loop invariant
for (i=1; i<=100000; i++)
      cout << x+i << endl;
```

This would eliminate 99,999 unnecessary run-time calls to the `sqrt` function.

The reader is cautioned that global optimization can have a serious impact on run-time debugging. For example, if the value of `y` in the above example was negative, causing a run-time error in the `sqrt` function, the user would be unaware of the actual location of that portion of code which called the `sqrt` function, because the compiler would have moved the offending statement (usually without informing the programmer). Most compilers that perform global optimization also have a switch with which the user can turn optimization on or off. When debugging the program, the switch would be off. When the program is correct, the switch would be turned on to generate an optimized version for the user. One of the most difficult problems for the compiler writer is making sure that the compiler generates optimized and unoptimized object modules, from the same source module, which are equivalent.

## 4. Code Generation

As discussed earlier, the computer is capable of executing only a limited number of primitive operations on operands with numeric memory addresses, all encoded as binary values, constituting the machine language. In the *code generation* phase, atoms or syntax trees are translated to machine language (binary) instructions, or to assembly language, in which case the assembler is invoked to produce the object program. Symbolic addresses (statement labels and data identifiers) are translated to relocatable memory addresses at this time.

For target machines with several CPU registers, the code generator is responsible for *register allocation*. This means that the compiler must be aware of which registers are being used for particular purposes in the generated program, and which become available as code is generated.

For example, an ADD atom might be translated to three machine language instructions: (1) load the first operand into a register; (2) add the second operand to that register; and (3) store the result, as shown for the atom (ADD, A, B, Temp):

```
LOD  R1,A     // Load A into reg. 1
ADD  R1,B     // Add B to reg. 1
STO  R1,Temp  // Store reg. 1 in Temp
```

It is not uncommon for the object language to be another high-level language. This is done in order to improve portablility of the language being implemented.

## 5. Local Optimization

The *local optimization* phase is also optional and is needed only to make the object program more efficient. It involves examining sequences of instructions put out by the code generator to find unnecessary or redundant instructions. For this reason, local optimization is often called machine-dependent optimization. An *addition operation* in the source program might result in three instructions in the object program: (1) load one operand into a register; (2) add the other operand to the register; and (3) store the result. Consequently, the expression `A + B + C` in the source program might result in the following instructions as code generator output:

```
LOD  R1,A      // Load A into
               // register 1
ADD  R1,B      // Add B to register 1
STO  R1,TEMP1  // Store the result in
               // TEMP1*
LOD  R1,TEMP1  // Load result into
               // reg 1*
ADD  R1,C      // Add C to register 1
STO  R1,TEMP2  // Store the result in
               // TEMP2
```

Note that some of these instructions (those marked with * in the comment) can be eliminated without changing the effect of the program, making the object program both smaller and faster:

```
LOD  R1,A     // Load A into
              // register 1
ADD  R1,B     // Add B to register 1
ADD  R1,C     // Add C to register 1
STO  R1,TEMP  // Store the result in
              // TEMP
```

A diagram showing the phases of compilation and the output of each phase is shown in Fig. 4. Note that the optimization phases may be omitted (i.e., the atoms may be passed directly from the Syntax phase to the Code Generator, and the instructions may be passed directly from the Code Generator to the compiler output file.)

A word needs to be said about the flow of control between phases. One way to handle this is for each phase to run from start to finish separately, writing output to a disk file. For example, lexical analysis is started and creates a file of tokens. Then, after the entire source program has been scanned, the syntax analysis phase is started, reads the entire file of tokens, and creates a file of atoms. The other phases continue in this manner; this would be a *multiple pass compiler* since the input is scanned several times.

Another way for flow of control to proceed would be to start up the syntax analysis phase first. Each time it needs a token it calls the lexical analysis phase as a subroutine, which reads enough source characters to produce one token, and returns it to the parser. Whenever the parser has scanned enough source code to produce an atom, the atom is converted to object code by calling the code generator as a subroutine; this would be a *single pass compiler*.

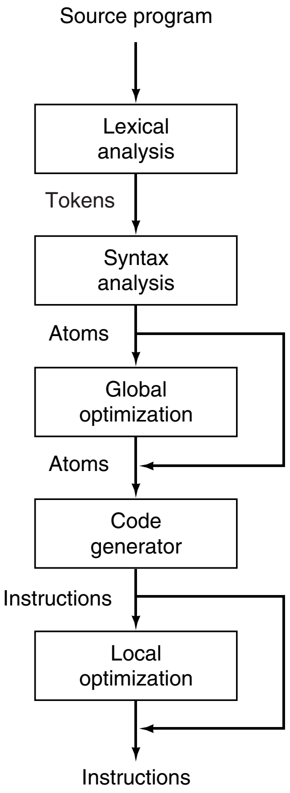


**Figure 4** The phases of a compiler.

## D. Some Implementation Techniques

By this point it should be clear that a compiler is not a trivial program. A new compiler, with all optimizations, could take over a person-year to implement. For this reason, we are always looking for techniques or shortcuts which will speed up the development process. This often involves making use of compilers, or portions of compilers, which have been developed previously. It also may involve special compiler generating tools, such as *lex* and *yacc,* which are part of the Unix environment.

In order to describe these implementation techniques graphically, we use the method shown in Fig. 5, in which the computer is designated with a rectangle, and its name is in a smaller rectangle sitting on top of the computer. In all of the examples shown here the program loaded into the computer's memory will be a compiler. It is important to remember that a computer is capable of running only programs written in the machine language of that computer. The input and output (also compilers in our examples) to the program in the computer are shown to the left and right, respectively.

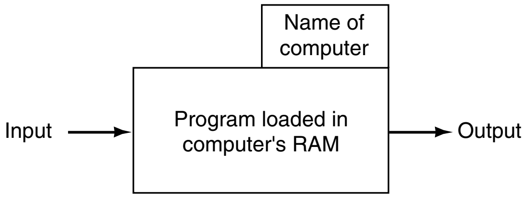


**Figure 5** Notation for a program running on a computer.

Since a compiler does not change the purpose of the source program, the superscript on the output is the same as the superscript on the input (X → Y), as shown in Fig. 6. The subscript language (the language in which it exists) of the executing compiler (the one inside the computer), M, must be the machine language of the computer on which it is running. The subscript language of the input, S, must be the same as the source language of the executing compiler. The subscript language of the output, O, must be the same as the object language of the executing compiler.

In the following sections it is important to remember that a compiler does not change the purpose of the source program; a compiler *translates* the source program into an equivalent program in another language (the object program). The source program could, itself, be a compiler. If the source program is a compiler which translates language A into language B, then the object program will also be a compiler which translates language A into language B.

## 1. Bootstrapping

The term *bootstrapping* is derived from the phrase "pull yourself up by your bootstraps" and generally involves the use of a program as input to itself (a *bootstrapping loader* is used to initialize a computer by loading a more sophisticated version of itself just after it has been switched on, hence the expression "to boot" a computer).

In this article, we are talking about bootstrapping a compiler, as shown in Fig. 7. We wish to implement a Java compiler for the VAX computer. Rather than writing the whole thing in machine (or assembly) language, we instead choose to write two easier programs. The first is a compiler for a subset of Java, written in machine (assembly) language. The second is a compiler for the full Java language written in the Java subset language. In Fig. 7 the subset language is designated "Sub," and it is simply Java, without several of the superfluous features, such as enumerated types, interfaces, case statements, etc. The first compiler is loaded into the computer's memory and the second is used as input. The output is the compiler we want—i.e., a compiler for the full Java language, which runs on a VAX and produces object code in VAX machine language.

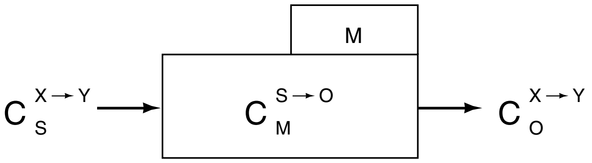


**Figure 6** Notation for a compiler being translated to a different language.

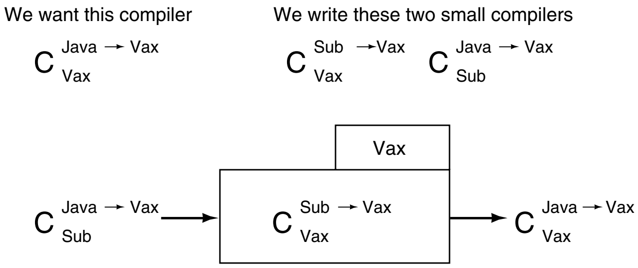


**Figure 7** Bootstrapping Java onto a VAX computer.

In actual practice this is an iterative process, beginning with a small subset of Java, and producing, as output, a slightly larger subset. This is repeated, using larger and larger subsets, until we eventually have a compiler for the complete Java language.

## 2. Cross Compiling

New computers with enhanced (and sometimes reduced) instruction sets are constantly being produced in the computer industry. The developers face the problem of producing a new compiler for each existing programming language each time a new computer is designed. This problem is simplified by a process called *cross compiling*.

Cross compiling is a two-step process and is shown in Fig. 8. Suppose that we have a Java compiler for the VAX, and we develop a new machine called a Mac. We now wish to produce a Java compiler for the Mac without writing it entirely in machine (assembly) language; instead, we write the compiler in Java. Step one is to use this compiler as input to the Java compiler on the VAX. The output is a compiler that translates Java into Mac machine language, and which runs on a VAX. Step two is to load this compiler into the VAX and use the compiler we wrote in Java as input once again. This time the output is a Java compiler for the Mac which runs on the Mac, i.e., the compiler we wanted to produce.

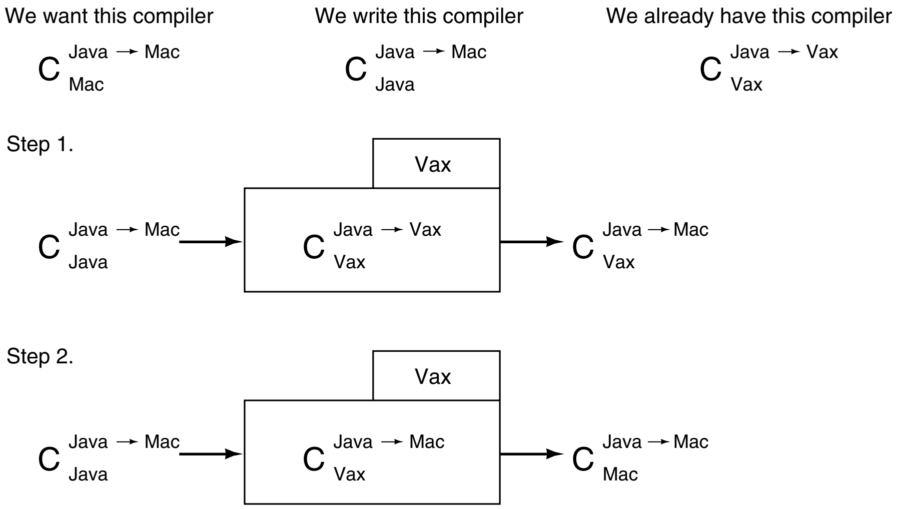


**Figure 8** Cross compiling Java from a VAX computer to a Mac computer.

Note that this entire process can be completed before a single Mac has been built. All we need to know is the architecture (the instruction set, instruction formats, addressing modes, etc.) of the Mac.

## 3. Compiling to Intermediate Form

As we mentioned in our discussion of interpreters above, it is possible to compile to an *intermediate form,* which is a language somewhere between the source high-level language and machine language. The stream of atoms put out by the parser is a possible example of an intermediate form. The primary advantage of this method is that one needs only one translator for each high-level language to the intermediate form (each of these is called a *front end*) and only one translator (or interpreter) for the intermediate form on each computer (each of these is called a *back end*). As depicted in Fig. 9, for three high-level languages and two computers we would need three translators to intermediate form and two code generators (or interpreters)—one for each computer. Had we not used the intermediate form, we would have needed a total of six different compilers. In general, given n high-level languages and m computers, we would need n x m compilers. Assuming that each front end and each back end is half of a compiler, we would need (n+m)/2 compilers using intermediate form.

A very popular intermediate form for the PDP-8 and Apple II series of computers, among others, called *p-code,* was developed several years ago at the University of California at San Diego. More recently, high-level languages such as C have been commonly used as an intermediate form. Today the Java byte code language developed by Sun Microsystems is an intermediate form which has been widely used on the internet.

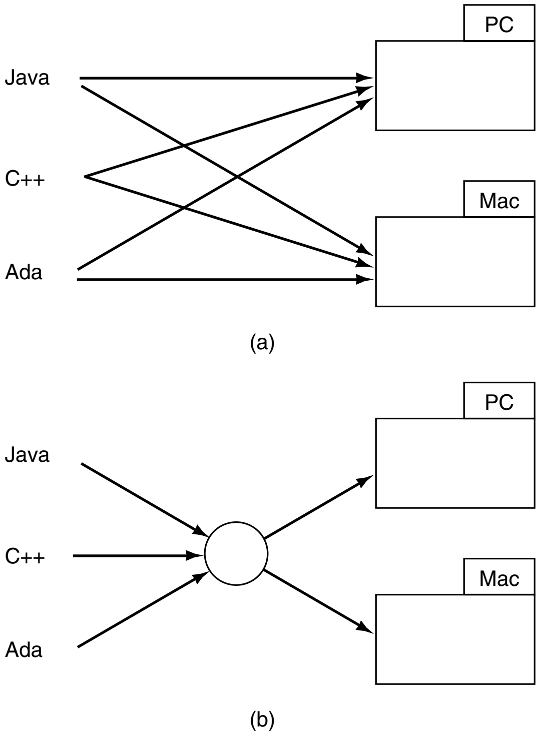


**Figure 9** (a) Six compilers needed to implement three languages on two computers without an intermediate form. (b) Fewer than three compilers needed for the same problem with an intermediate form.

## 4. Compiler-Compilers

Much of compiler design is understood so well at this time that the process can be automated. It is possible for the compiler writer to write specifications of the source language and of the target machine so that the compiler can be generated automatically. This is done by a *compiler-compiler,* which is introduced in Sections II and V with the study of the *lex* and *yacc* utilities of the Unix programming environment.

## II. THE LEXICAL PHASE

The first phase of a compiler is called *lexical analysis.* Because this phase scans the input string without backtracking (i.e., by reading each symbol once, and processing it correctly), it is often called a *lexical scanner.* As implied by its name, lexical analysis attempts to isolate the "words" in an input string. We use the word "word" in a technical sense. A *word,* also known as a *lexeme,* a *lexical item,* or a *lexical token,* is a string of input characters which is taken as a unit and passed on to the next phase of compilation. Examples of words are

1. *keywords*—`while, if, else, for,` These are words which may have a particular predefined meaning to the compiler, as opposed to identifiers which have no particular meaning. Reserved words are keywords which are not available to the programmer for use as identifiers. In most programming languages, such as Java and C, all keywords are reserved. PL/1 is an example of a language which has no reserved words.
2. *identifiers*—Words that the programmer constructs to attach a name to a construct, usually having some indication as to the purpose or intent of the construct. Identifiers may be used to identify variables, classes, constants, functions, etc.
3. *operators*—Symbols used for arithmetic, character, or logical operations, such as `+,-,=,!=,` etc. Notice that operators may consist of more than one character.
4. *numeric constants*—Numbers such as `124, 12.35, 0.09E-23,` etc. These must be converted to a numeric format so that they can be used in arithmetic operations, because the compiler initially sees all input as a string of characters. Numeric constants may be stored in a table.
5. *character constants*—Single characters or strings of characters enclosed in quotes.
6. *special characters*—Characters used as delimiters such as `.,(,),{,},;.` These are generally single-character words.
7. *comments*—Though comments must be detected in the lexical analysis phase, they are not put out as tokens to the next phase of compilation.
8. *white space*—Spaces and tabs are generally ignored by the compiler, except to serve as delimiters in most languages, and are not put out as tokens.
9. *newline*—In languages with free format, newline characters should also be ignored, otherwise a newline token should be put out by the lexical scanner.

An example of C++ source input, showing the word boundaries and token types put out is given below:

```
while ( x33 <= 2.5 e+33 = total )
  1   6  2  3      4    3   2   6
calc ( x33 ) ; //!
  2  6  2  6 6
```

During lexical analysis, a *symbol table* is constructed as identifiers are encountered. This is a data structure which stores each identifier once, regardless of the number of times it occurs in the source program. It also stores information about the identifier, such as the kind of identifier and where associated run-time information (such as the value assigned to a variable) is stored. This data structure is often organized as a binary search tree, or hash table, for efficiency in searching.

When compiling block-structured languages such as Java, C, or Algol, the symbol table processing is more involved. Since the same identifier can have different declarations in different blocks or procedures, both instances of the identifier must be recorded. This can be done by setting up a separate symbol table for each block, or by specifying block scopes in a single symbol table. This would be done during the parse or syntax analysis phase of the compiler; the scanner could simply store the identifier in a *string space* array and return a pointer to its first character.

Numeric constants must be converted to an appropriate internal form. For example, the constant "`3.4e+6`" should be thought of as a string of six characters which needs to be translated to floating point (or fixed point integer) format so that the computer can perform appropriate arithmetic operations with it. This is not a trivial problem, and most compiler writers make use of library routines to handle this.

The output of this phase is a stream of tokens, one token for each word encountered in the input program. Each token consists of two parts: (1) a class indicating which kind of token and (2) a value indicating which member of the class. The above example might produce the following stream of tokens:

| Token Class | Token Value |
|---|---|
| 1 | [code for while] |
| 6 | [code for (] |
| 2 | [ptr to symbol table entry for x33] |
| 3 | [code for <=] |
| 4 | [ptr to constant table entry for 2.5e+33] |
| 3 | [code for -] |
| 2 | [ptr to symbol table entry for total] |
| 6 | [code for )] |
| 2 | [ptr to symbol table entry for calc] |
| 6 | [code for (] |
| 2 | [ptr to symbol table entry for x33] |
| 6 | [code for )] |
| 6 | [code for ;] |

Note that the comment is not put out. Also, some token classes might not have a value part. For example, a left parenthesis might be a token class, with no need to specify a value.

Some variations on this scheme are certainly possible, allowing greater efficiency. For example, when an identifier is followed by an assignment operator, a single assignment token could be put out. The value part of the token would be a symbol table pointer for the identifier. Thus the input string "x =", would be put out as a single token, rather than two tokens. Also, each keyword could be a distinct token class, which would increase the number of classes significantly, but might simplify the syntax analysis phase.

Note that the lexical analysis phase does not check for proper syntax. The input could be } while if ( { and the lexical phase would put out five tokens corresponding to the five words in the input. (Presumably the errors will be detected in the syntax analysis phase.)

If the source language is not case sensitive, the scanner must accommodate this feature. For example, the following would all represent the same keyword: then, tHeN, Then, THEN. A preprocessor could be used to translate all alphabetic characters to upper (or lower) case.

Implementation techniques for the lexical phase often involve the use of finite automata theory and/or regular expressions. Lexical tokens in programming languages are almost always regular languages (see the article on Automata Theory) and do not require a stack for processing.

There is a software tool called lex which is available with the Unix programming environment which can generate a lexical analyzer. The input to lex is a series of patterns, or regular expressions, and the output is a C program which is the lexical phase of a compiler.

## III. THE SYNTAX AND SEMANTIC PHASES

The second phase of a compiler is called *syntax analysis.* The input to this phase consists of a stream of tokens put out by the lexical analysis phase. They are then checked for proper syntax, i.e., the compiler checks to make sure the statements and expressions are correctly formed. Some examples of syntax errors in C++ are

```
x = (2+3) * 9);          // mismatched
                         // parentheses
if x>y x = 2;            // missing
                         // parentheses
while (x==3) do fl();    // invalid key
                         // word do
```

When the compiler encounters such an error, it should put out an informative message for the user. At this point, it is not necessary for the compiler to generate an object program. A compiler is not expected to guess the intended purpose of a program with syntax errors. A good compiler, however, will continue scanning the input for additional syntax errors.

The output of the syntax analysis phase (if there are no syntax errors) could be a stream of atoms or syntax trees. An *atom* is a primitive operation which is found in most computer architectures, or which can be implemented using only a few machine language instructions. Each atom also includes operands, which are ultimately converted to memory addresses on the target machine. A *syntax tree* is a data structure in which the interior nodes represent operations, and the leaves represent operands. We will see that the parser can be used not only to check for proper syntax, but to produce output as well. This process is called *syntax directed translation.* It is also possible to do some semantic analysis during this phase of compilation. Checking of data types can be accomplished at this time, and, if necessary, calls to type conversion routines can be inserted.

Formal methods are almost always used to construct the syntax phase of the compiler. Most of the early work in the theory of compiler design focused on syntax analysis, resulting in a wide range of methods that can be used to construct the syntax phase (see the article on Automata Theory). Formal grammars are used not only to specify the programming language, but also as a means of implementing the syntax analysis phase of the compiler.

A tool called yacc (yet another compiler-compiler), which is available in the Unix programming environment, is often used to generate parsers automatically. The input to yacc is a grammar (see the section on Automata Theory) for the language to be implemented, and the output is a parser for that language. The grammar may be supplemented with code to define the semantics of the language and to produce output.

## A. Ambiguities in Programming Languages

Ambiguities in grammars for programming languages should be avoided. One way to resolve an ambiguity is to rewrite the grammar of the language so as to be unambiguous. For example, the grammar shown below is an ambiguous grammar for simple arithmetic expressions involving only addition and multiplication. The ambiguity can be demonstrated by observing that there are two different derivation trees for `var + var * var`.

1. `Expr` → `Expr + Expr`
2. `Expr` → `Expr * Expr`
3. `Expr` → `( Expr )`
4. `Expr` → `var`
5. `Expr` → `const`

This ambiguity can be eliminated by writing an equivalent grammar which is not ambiguous, as shown below:

1. `Expr` → `Expr + Term`
2. `Expr` → `Term`
3. `Term` → `Term * Factor`
4. `Term` → `Factor`
5. `Factor` → `( Expr )`
6. `Factor` → `var`
7. `Factor` → `const`

A derivation tree for the input string `var + var * var` is shown in Fig. 10. There is no other derivation tree for this input string, because the grammar is not ambiguous. Also note that in any derivation tree using this grammar, subtrees correspond to subexpressions, according to the usual precedence rules. The derivation tree in Fig. 10 indicates that the multiplication takes precedence over the addition. The left associativity rule would also be observed in a derivation tree for `var + var + var`.

Another example of ambiguity in programming languages is the conditional statement as defined below:

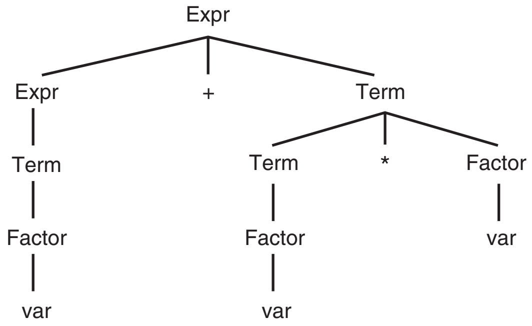


**Figure 10** The only derivation tree for `var + var * var` using the nonambiguous grammar.

1. `Stmt` → `IfStmt`
2. `IfStmt` → `if ( Expr ) Stmt`
3. `IfStmt` → `if ( Expr ) Stmt else Stmt`

Think of this grammar as part of a larger grammar in which the nonterminal `Stmt` is completely defined. For the present example we will show derivation trees in which some of the leaves are left as nonterminals. Two different derivation trees for the input string `if (Expr) then if (Expr) then Stmt else Stmt` are shown in Fig. 11. In this grammar, an `Expr` is interpreted as False (0) or True (nonzero), and a `Stmt` is any statement, including `if` statements. This ambiguity is normally resolved by informing the programmer that *elses* always are associated with the closest previous unmatched *ifs*. Thus, the second derivation tree in Fig. 11 corresponds to the correct interpretation. This grammar can be rewritten as an equivalent grammar which is not ambiguous:

1. `Stmt` → `IfStmt`
2. `IfStmt` → `Matched`
3. `IfStmt` → `Unmatched`
4. `Matched` → `if ( Expr ) Matched else Matched`
5. `Matched` → `OtherStmt`
6. `Unmatched` → `if ( Expr ) Stmt`
7. `Unmatched` → `if ( Expr ) Matched else Unmatched`

This grammar differentiates between the two different kinds of if statements, those with a matching else `(Matched)` and those without a matching else `(Unmatched)`. The nonterminal `OtherStmt` would be defined with rules for statements other than `if` statements (while, expression, for, . . .). A derivation tree for the string if `( Expr ) if ( Expr ) OtherStmt else OtherStmt` is shown in Fig. 12.

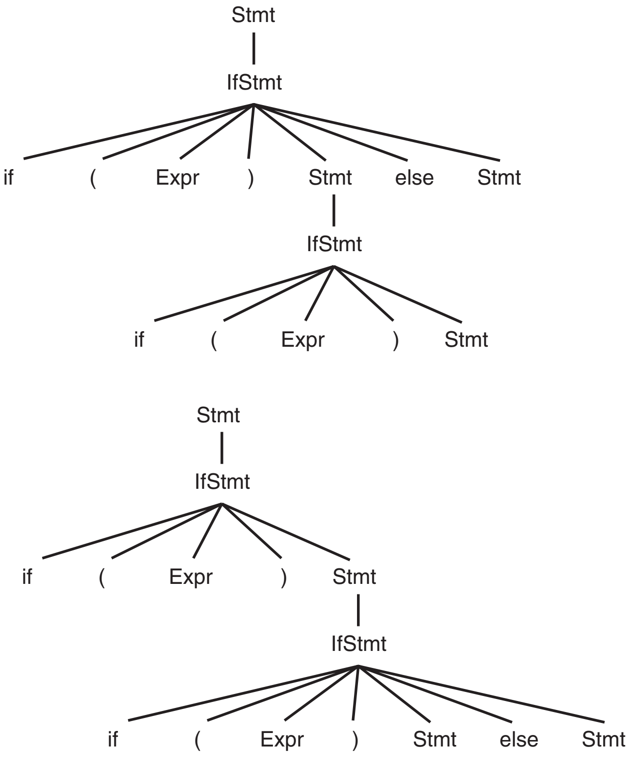


**Figure 11** Two different derivation trees for: `if ( Expr ) if ( Expr ) Stmt else Stmt`.

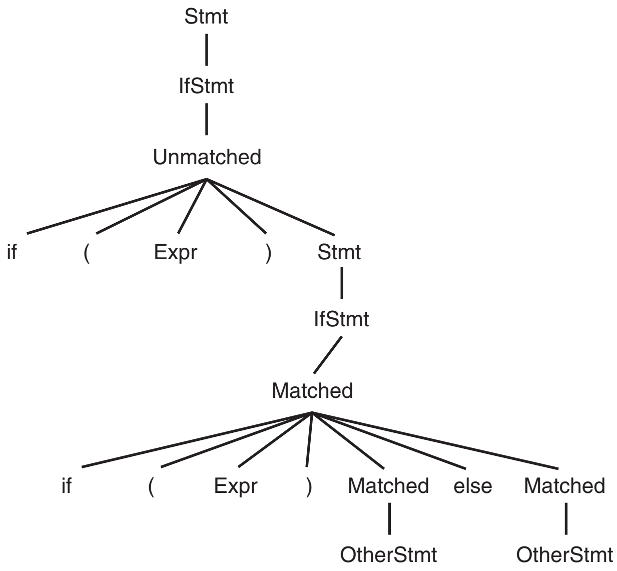


**Figure 12** A derivation tree for `if ( Expr ) if ( Expr ) OtherStmt else OtherStmt` using a nonambiguous grammar.

## B. The Parsing Problem

The reader may recall, from high school days, the problem of diagramming English sentences. The problem is to put words together into groups and assign syntactic types to them, such as noun phrase, predicate, and prepositional phrase. An example of a diagrammed English sentence is shown in Fig. 13. The process of diagramming an English sentence corresponds to the problem a compiler must solve in the syntax analysis phase of compilation.

The syntax analysis phase of a compiler must be able to solve the *parsing problem* for the programming language being compiled: Given a grammar, G, and a string of input symbols, decide whether the string is in L(G); also, determine the structure of the input string. The solution to the parsing problem will be "yes" or "no," and, if "yes," some description of the input string's structure, such as a derivation tree.

A *parsing algorithm* is one which solves the parsing

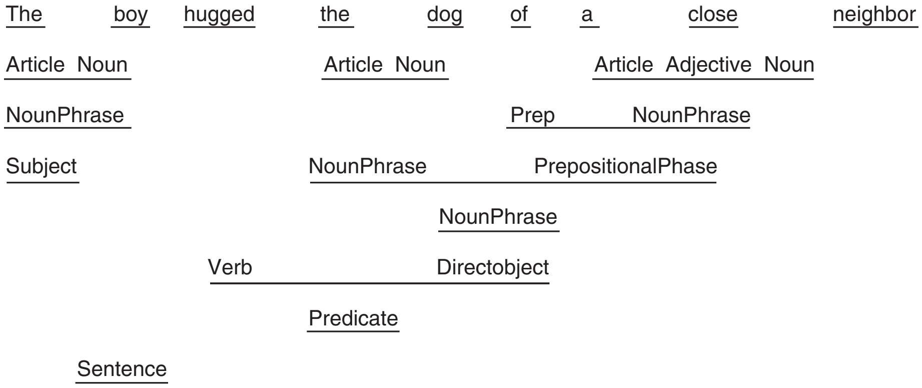


**Figure 13** Diagram of an English sentence.

problem for a particular class of grammars. A good parsing algorithm will be applicable to a large class of grammars and will accommodate the kinds of rewriting rules normally found in grammars for programming languages. For context-free grammars, there are two kinds of parsing algorithms—*bottom up* and *top down*. These terms refer to the sequence in which the derivation tree of a correct input string is built. A parsing algorithm is needed in the syntax analysis phase of a compiler.

There are parsing algorithms which can be applied to any context-free grammar, employing a complete search strategy to find a parse of the input string. These algorithms are generally considered unacceptable since they are too slow; they cannot run in "polynomial time."

## IV. PARSING TOP DOWN

In a top-down parsing algorithm, grammar rules are applied in a sequence which corresponds to a general top-down direction in the derivation tree. For example, consider the grammar:

```
1. S → a S b
2. S → b A c
3. A → b S
4. A → a
```

A derivation tree for the input string abbbaccb is shown in Fig. 14. A parsing algorithm will read one input symbol at a time and try to decide, using the grammar, whether the input string can be derived. A top-down algorithm will begin a derivation with the starting nonterminal and try to decide which rule of the grammar should be applied. In the example of Fig. 14, the algorithm is able to make this decision by examining a single input symbol and comparing it with the first symbol on the right side of the rules. Figure 15 shows the sequence of events, as input symbols are read, in which the numbers in circles indicate which grammar rules are being applied, and the underscored symbols are the ones which have been read by the parser. Careful study of Figs. 14 and 15 reveals that this sequence of events corresponds to a top-down construction of the derivation tree.

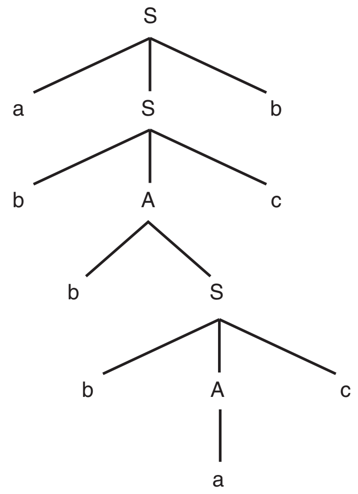


**Figure 14** A derivation tree for `abbbaccb`.

Not all context-free grammars can be parsed top down; however, programming languages are often defined in such a way that it is fairly easy to find a grammar for the language which can be parsed top down (this property is called LL in the literature).

As an example, consider the following grammar for statements in a typical programming language (assume that expressions are defined previously, as `Expr`, and that a `StmtList` is a sequence of zero or more statements):

```
1. Stmt → var = Expr ;
2. Stmt → if ( Expr ) Stmt
3. Stmt → while ( Expr ) Stmt
4. Stmt → { StmtList }
5. StmtList → Stmt StmtList
6. StmtList → ε
```

We wish to see whether the following string is a valid statement as defined by the grammar:

```
{ if ( Expr ) var = Expr ;
  while ( Expr ) var = Expr;
}
```

The derivation is straightforward because the right side of each rule in the grammar begins with a terminal symbol, making it easy to decide which rule to apply at each step. The result is shown below:

```
Stmt ⇒ { StmtList } ⇒ { Stmt Stmt }
     ④
     ⇒ { if ( Expr ) Stmt Stmt }
     ②
     ⇒ { if ( Expr ) var = Expr ;
     ①                Stmt }


     ⇒ { if (Expr ) var = Expr ;
     ③   while ( Expr ) Stmt }


     ⇒ { if (Expr ) var = Expr ;
     ①   while ( Expr ) var = Expr ; }
```

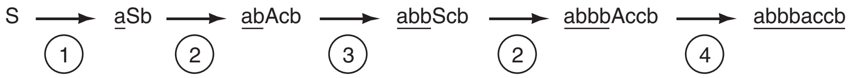


**Figure 15** Sequence of events in a top-down parse.

There is a well-known parsing algorithm called recursive descent, which implements a top-down parse of any input string. The algorithm consists of a function for each nonterminal symbol in the grammar. The purpose of that function is to scan as many input symbols as necessary to find an example of the corresponding nonterminal. The function is written directly from the grammar rules which define its nonterminal symbol; it will read an input symbol for each terminal symbol in the rule, and call a function for each nonterminal symbol in the rule. As an example, the function for the nonterminal Stmt is shown below, using C++ as the implementation language.

```
bool Stmt ()
// Precondition: The first symbol in a
//   Stmt has been read into inp
// Postcondition: If the input
//   constitutes a valid Stmt, all the
//   symbols in the Stmt will be read,
//   and inp will contain the next input
//   symbol, and a true value is returned.
//   Otherwise a false value is returned.
     {  if (inp==Var)    // rule 1
       {  cin >> inp;      // read past
                           // the var
          if (inp== ‘=’)   // check for
            cin >> inp;    // a ‘=’
          else return
            false;
          if (!Expr())     // scan for
            return false;  // an Expr
          if (inp==’;’)    // check for
            cin >> inp;    // a ‘;’
          else return
            false;
          return true;     // found a
                           // complete
                           // Stmt
       }
          if (inp==If)   // rule 2
       {  cin >> inp;      // read past
                           // the if
          if (inp==’(‘)    // check for
            cin >> inp;    // a ‘(‘
          else return
            false;
          if (!Expr())     // scan for
            return false;  // an Expr
          if (inp==’)’)    // check for
            cin >> inp;    // a ‘)’
          else return
            false;
          if (!Stmt())     // scan for
            return false;  // a Stmt
          return true;     // found a
                           // complete
                           // Stmt
       }
       if (inp==While)   // rule 3
       {  cin >> inp;      // read past
                           // the while
          if (inp==’(‘)    // check for
            cin >> inp;    // a ‘(‘
          else return
            false;
          if (!Expr())     // scan for
            return false;  // an Expr
          if (inp==’)’)    // check for
            cin >> inp;    // a ‘)’
          else return false;
          if (!Stmt())     // scan for
            return false;  // a Stmt
          return true;     // found a
                           // complete
                           // Stmt
       }
       if (inp==’{‘)     // rule 4
       {  cin >> inp;      // read past
                           // the ‘{‘
          if               // scan for a
          (!StmtList())    // StmtList
            return false;
          if (inp==’}’)    // check for
          cin >> inp;      // a ‘}’
          else return
            false;
          return true;     // found a
                           // complete
                           // Stmt
       }
       return false;       // no rules
                           // apply,
                           // could not
                           // find a
                           // Stmt
     }
```

Type checking and other semantics can then be added to these functions. For example, they could put out atoms during the parse, or build a syntax tree as the rules of the grammar are applied.

## V. PARSING BOTTOM UP

### A. Shift-Reduce Parsing

A bottom-up parsing algorithm is one which applies grammar rules in a sequence that corresponds to an upward direction in a derivation tree. Most bottom-up parsing algorithms use a technique called shift-reduce parsing. A stack is used to store input symbols and grammar symbols. A shift operation transfers an input symbol from the input string to the top of the stack, and a reduce operation replaces 0 or more symbols on top of the stack with a nonterminal symbol. One way to implement shift-reduce parsing is with tables that determine whether to shift or reduce, and which grammar rule to reduce. This method makes use of two tables to control the parser. The first table, called the *action table,* determines whether a shift or reduce is to be invoked. If it specifies a reduce, it also indicates which grammar rule is to be reduced. The second table, called a *goto table,* indicates which stack symbol is to be pushed on the stack after a reduction. A shift action is implemented by a push operation followed by an advance input operation. A reduce action must always specify the grammar rule to be reduced. The reduce action is implemented by a Replace operation in which stack symbols on the right side of the specified grammar rule are replaced by a stack symbol from the goto table (the input pointer is retained). The symbol pushed is not necessarily the nonterminal being reduced, as shown below. In practice, there will be one or more stack symbols corresponding to each nonterminal.

The columns of the goto table are labeled by nonterminals, and the the rows are labeled by stack symbols. A cell of the goto table is selected by choosing the column of the nonterminal being reduced and the row of the stack symbol just beneath the handle.

For example, suppose we have the following stack and input configuration:

```
Stack     Input

∇S        ab↵
```

in which the bottom of the stack is to the left. The action *shift* will result in the following configuration:

```
Stack     Input

∇Sa       b↵
```

The a has been shifted from the input to the stack. Suppose, then, that in the grammar, rule 7 is:

```
7.  B →Sa
```

Select the row of the goto table labeled ∇, and the column labeled B. If the entry in this cell is push X, then the action reduce 7 would result in the following configuration:

```
Stack     Input

∇X        b↵
```

An LR parsing algorithm is one which finds a Rightmost derivation when scanning from the Left. Figure 16 shows the LR parsing tables for the nonambiguous grammar for arithmetic expressions involving only addition and multiplication shown below. The stack symbols label the rows, and the input symbols label the columns of the action table. The columns of the goto table are labeled by the nonterminal being reduced. The stack is initialized with a ∇ symbol, and blank cells in the action table indicate syntax errors in the input string. Figure 17 shows the sequence of configurations which would result when these tables are used to parse the input string `(var+var)*var`.

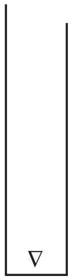
```
Expr → Expr + Term
Expr → Term
Term → Term * Factor
Term → Factor
Factor → ( Expr )
Factor → var
```

The operation of the LR parser can be described as follows:

1. Find the action corresponding to the current input and the top stack symbol.
2. If that action is a shift action:
   a. Push the input symbol onto the stack.
   b. Advance the input pointer.
3. If that action is a reduce action:
   a. Find the grammar rule specified by the reduce action.
   b. The symbols on the right side of the rule should also be on the top of the stack—pop them all off the stack.
   c. Use the nonterminal on the left side of the grammar rule to indicate a column of the goto table, and use the top stack symbol to indicate

| | Action Table | | | | | |
|---|---|---|---|---|---|---|
| | + | * | ( | ) | var | ↵ |
| ∇ | | | shift ( | | shift var | |
| Expr1 | shift + | shift * | | | | Accept |
| Term1 | reduce 1 | reduce 3 | | reduce 1 | | reduce 1 |
| Factor3 | reduce 3 | | | reduce 3 | | reduce 3 |
| ( | | | shift ( | | shift var | |
| Expr5 | shift + | | | shift ) | | |
| ) | reduce 5 | reduce 5 | | reduce 5 | | reduce 5 |
| + | | | shift ( | | shift var | |
| Term2 | reduce 2 | shift * | | reduce 2 | | reduce 2 |
| * | | | shift ( | | shift var | |
| Factor4 | reduce 4 | reduce 4 | | reduce 4 | | reduce 4 |
| var | reduce 6 | reduce 6 | | reduce 6 | | reduce 6 |

| | Goto Table | | |
|---|---|---|---|
| | Expr | Term | Factor |
| ∇ | push Expr1 | push Term2 | push Factor4 |
| Expr1 | | | |
| Term1 | | | |
| Factor3 | | | |
| ( | push Expr5 | push Term2 | push Factor4 |
| Expr5 | | | |
| ) | | | |
| + | | push Term1 | push Factor4 |
| Term2 | | | |
| * | | | push Factor3 |
| Factor4 | | | |
| var | | | |



Initial stack

**Figure 16** Action and goto tables to parse simple arithmetic expressions.

a row of the goto table. Push the indicated stack symbol onto the stack.

d. Retain the input pointer.

4. If that action is blank, a syntax error has been detected.
5. If that action is Accept, terminate.
6. Repeat from step 1.

There are three principle methods for constructing the LR parsing tables. In order from simplest to most complex or general, they are called: simple LR (SLR), look ahead LR (LALR), and canonical LR (LR). SLR is the easiest method to implement, but works for a small class of grammars. LALR is more difficult and works on a slightly larger class of grammars. LR is the most general, but still does not work for all unambiguous context free grammars. In all cases, they find a rightmost derivation when scanning from the left (hence LR). These methods are beyond the scope of this article, but are described in Parsons (1992) and Aho (1986).

## B. Overview of yacc

For many grammars, the LR parsing tables can be generated automatically from the grammar. One of the most popular software systems that does this is available in the Unix programming environment; it is called *yacc* (yet another compiler-compiler). A *compiler-compiler* is a program which takes as input the specification of a programming language (in the form of a grammar), and produces as output a compiler for that language. By itself, yacc is really just a parser generator yielding a program which checks for syntax,

```
Stack                          Input  Action          Goto

∇                      (var+var)*var
                                      shift (
∇ (                     var+var)*var ↵
                                      shift var
∇ (var                      +var)*var ↵
                                      reduce 6        push Factor4
∇ (Factor4                  +var)*var ↵
                                      reduce 4        push Term2
∇ (Term2                    +var)*var ↵
                                      reduce 2        push Expr5
∇ (Expr5                    +var)*var ↵
                                      shift +
∇ (Expr5+                    var)*var ↵
                                      shift var
∇ (Expr5+var                    )*var ↵
                                      reduce 6        push Factor4
∇ (Expr5+Factor4                )*var ↵
                                      reduce 4        push Term1
∇ (Expr5+Term1                  )*var ↵
                                      reduce 1        push Expr5
∇ (Expr5                        )*var ↵
                                      shift )
∇ (Expr5)                        *var ↵
                                      reduce 5        push Factor4
∇ Factor4                        *var ↵
                                      reduce 4        push Term2
∇ Term2                          *var ↵
                                      shift *
∇ Term2*                          var ↵
                                      shift var
∇ Term2*var                           ↵
                                      reduce 6        push Factor3
∇ Term2*Factor3                       ↵
                                      reduce 3        push Term2
∇ Term2                               ↵
                                      reduce 2        push Expr1
∇ Expr1                               ↵
                                      Accept
```

**Figure 17** Sequence of configurations when parsing `(var+var)*var`.

but since it is possible for the user to augment it with additional features, it can be used to generate a complete compiler. An available public domain version of yacc is called *bison.* There are also personal computer versions of yacc which use Pascal as a base language instead of C.

yacc generates a C function named yyparse(), which is stored in a file named y.tab.c. This function calls a function named yylex() whenever it needs an input token. The yylex() function may be written by the user and included as part of the yacc specification, or it may be generated by the lex utility. A diagram showing the flow of data needed to generate and compile software is shown in Fig. 18. It assumes that yylex() is

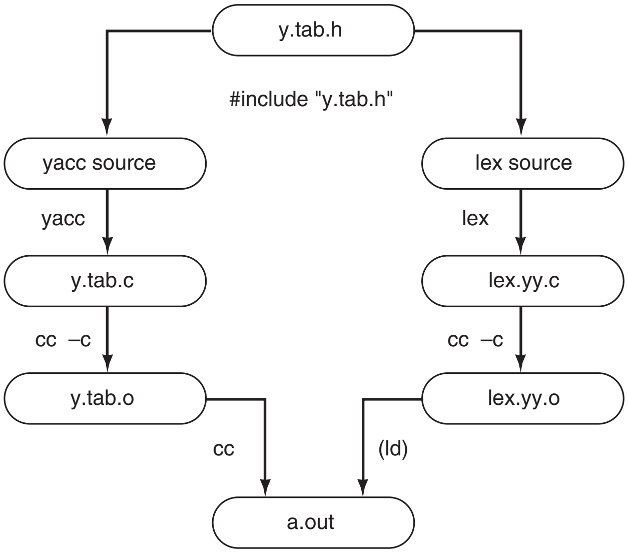


**Figure 18** Generation and compilation of software using lex and yacc.

being generated by lex and that definitions needed in both `yyparse()` from yacc and `yylex()` from lex are stored in the header file `y.tab.h`.

## C. Structure of the yacc Source File

The input to yacc is called the *yacc source file*. It consists of three parts, which are separated by the %% delimiter:

Declarations
%%
Rules
%%
Support Routines

The Declarations section contains declarations of token names, stack type, and precedence information which may be needed by yacc. It also may contain preprocessor statements (#include or #define) and declarations to be included in the output file, y.tab.c.

The Rules section is the grammar for the language being specified, such as Java. This is the most important part of the yacc source file. Each rule is of the form:

```
nonterminal:   α    {action}
              |β    {action}
              |γ    {action}
               .
               .
               .
               ;
```

where α, β, and γ are definitions of the nonterminal. The vertical bar designates alternative definitions for a non-terminal, as in Backus–Naur form (BNF). An action may be associated with each of the alternatives. This action is simply a C statement which is invoked during the parsing of an input string when the corresponding grammar rule is reduced. The rules may be written in free format, and each rule is terminated with a semicolon.

The third section of the yacc source file contains support routines, i.e., C functions which could be called from the actions in the Rules section. For example, when processing an assignment statement, it may be necessary to check that the type of the expression matches the type of the variable to which it is being assigned. This could be done with a call to a type-checking function in the third section of the yacc source file.

# VI. CODE GENERATION

## A. Introduction to Code Generation

Up to this point we have ignored the architecture of the machine for which we are building the compiler, i.e., the target machine. By *architecture,* we mean the definition of the computer's central processing unit as seen by a machine language programmer. Specifications of instruction-set operations, instruction formats, addressing modes, data formats, CPU registers, input/output instructions, etc., all make up what is sometime called the *conventional machine language* architecture (to distinguish it from the microprogramming level architecture which many computers have). Once these are all clearly and precisely defined, we can complete the compiler by implementing the *code generation* phase. This is the phase which accepts as input the syntax trees or stream of atoms as put out by the syntax phase, and produces, as output, the object language program in binary coded instructions in the proper format.

The primary objective of the *code generator* is to convert atoms or syntax trees to instructions. In the process, it is also necessary to handle register allocation for machines that have several general purpose CPU registers. Label atoms must be converted to memory addresses. For some languages, the compiler has to check data types and call the appropriate type conversion routines if the programmer has mixed data types in an expression or assignment.

Note that if we are developing a new computer, we don't need a working model of that computer in order to complete the compiler; all we need are the specifications, or architecture, of that computer. Many designers view the construction of compilers as made up of two logical parts—the front end and the back end. The front end consists of lexical and syntax analysis and is machine-independent. The back end consists of code generation and optimization and is very machine-dependent, consequently this section commences our discussion of the back end, or machine-dependendent, phases of the compiler.

## B. Converting Atoms to Instructions

If we temporarily ignore the problem of forward references (of Jump or Branch instructions), the process of converting atoms to instructions is relatively simple. For the most part all we need is some sort of case, switch, or multiple destination branch based on the class of the atom being translated. Each atom class would result in a different instruction or sequence of instructions. If the CPU of the target machine requires that all arithmetic be done in registers, then an example of a translation of an ADD atom would be as shown, below, in Fig. 19; i.e., an ADD atom is translated into an LOD (load into register) instruction, followed by an ADD instruction, followed by an STO (store register to memory) instruction.

Most of the atom classes would be implemented in a similar way. Conditional branch atoms (called TST atoms in our examples) would normally be implemented as a Load, Compare, and Branch, depending on the architecture of the target machine. The `MOV` (move data from one memory location to another) atom could be implemented as a `MOV` (Move) instruction, if permitted by the target machine architecture; otherwise it would be implemented as a Load followed by a Store.

Operand addresses which appear in atoms must be appropriately coded in the target machine's instruction format. For example, many target machines require operands to be addressed with a base register or data segment register and an offset from the contents of that register. If this is the case, the code generator must be aware of the presumed contents of the base register, and compute the offset so as to produce the desired operand address. For example, if we know that a particular operand is at memory location 1E `(hex)`, and the contents of the base register is `10` (hex), then the offset would be `0E`, because `10 + 0E = 1E`. In other words, the contents of the base register, when added to the offset, must equal the operand address.

```
(ADD, A, B, T1)    →    LOD    R1,A
                        ADD    R1,B
                        STO    R1,T1
```

**Figure 19** Translation of an ADD atom to instructions.

## C. Single Pass versus Multiple Passes

There are several different ways of approaching the design of the code generation phase. The difference between these approaches is generally characterized by the number of passes which are made over the input file. For simplicity, we will assume that the input file is a file of atoms. A code generator which scans this file of atoms once is called a *single pass* code generator, and a code generator which scans it more than once is called a *multiple pass* code generator.

The most significant problem relevant to deciding whether to use a single or multiple pass code generator has to do with forward jumps. As atoms are encountered, instructions can be generated, and the code generator maintains a memory address counter, or program counter. When a Label atom is encountered, a memory address value can be assigned to that Label atom (a table of labels is maintained, with a memory address assigned to each label as it is defined). If a Jump atom is encountered with a destination that is a higher memory address than the Jump instruction (i.e., a forward jump), the label to which it is jumping has not yet been encountered, and it will not be possible to generate the Jump instruction completely at this time. An example of this situation is shown, below, in Fig. 20 in which the jump to Label L1 cannot be generated because at the time the JMP atom is encountered the code generator has not encountered the definition of the Label L1, which will have the value 9.

A JMP atom results in a CMP (Compare instruction) followed by a JMP (Jump instruction) in this example.

There are two fundamental ways to resolve the problem of forward jumps. Single pass compilers resolve it by keeping a table of Jump instructions which have forward destinations. Each Jump instruction with a forward reference is generated incompletely (i.e., without a destination address) when encountered, and each is also entered into a *fixup table,* along with the Label to which it is jumping. As each Label definition is encountered, it is entered into a table of Labels, along with its address value. When all of the atoms have been read, all of the Label atoms will have been defined, and, at this time, the code generator can revisit all of the Jump instructions in the Fixup

```
Atom                Location     Instruction
(ADD,  A,  B,  T1)  4            LOD    R1,A
                    5            ADD    R1,B
                    6            STO    R1,T1
(JMP,L1)            7            CMP    0,0,0
                    8            JMP    ?
(LBL,L1)                                     (L1  =  9)
```

**Figure 20** Problem in generating a jump to a forward destination.

table and fill in their destination addresses. This is shown in Fig. 21, below, for the same atom sequence shown in Fig. 20. Note that when the (JMP, L1) atom is encountered, the Label L1 has not yet been defined, so the location of the Jump (8) is entered into the Fixup table. When the (LBL, L1) atom is encountered, it is entered into the Label table, because the target machine address corresponding to this Label (9) is now known. When the end of file (EOF) is encountered, the destination of the Jump instruction at location 8 is changed, using the Fixup table and the Label table, to 9.

Multiple pass code generators do not require a Fixup table. In this case, the first pass of the code generator does nothing but build the table of Labels, storing a memory address for each Label. Then, in the second pass, all the Labels will have been defined, and each time a Jump is encountered its destination Label will be in the table, with an assigned memory address. This method is shown in Fig. 22 which, again, uses the atom sequence given in Fig. 20.

Note that, in the first pass, the code generator needs to know how many machine language instructions correspond to an atom (three to an ADD atom and two to a JMP atom), though it does not actually generate the instructions. It can then assign a memory address to each Label in the Label table.

A single pass code generator could be implemented as a subroutine to the parser. Each time the parser generates an atom, it would call the code generator to convert the atom to machine language and put out the instruction(s) corresponding to that atom. A multiple pass code generator would have to read from a file of atoms, created by the parser.

## VII. OPTIMIZATION

### A. Introduction to Optimization

In recent years, most research and development in the area of compiler design has been focused on the optimization phases of the compiler. *Optimization* is the process of improving generated code so as to reduce its potential running time and/or reduce the space required to store it in memory. Software designers are often faced with decisions which involve a space-time trade-off—i.e., one method will result in a faster program, another method will result in a program which requires less memory, but no method will

```
                                            Fixup  Table      Label  Table
Atom             Loc      Instruction       Loc     Label     Label     Value
(ADD,A,B,T1)     4        LOD   R1,A
                 5        ADD   R1,B
                 6        STO   R1,T1
(JMP,L1)         7        CMP   0,0,0
                 8        JMP   0           8       L1
(LBL,L1)                                                      L1        9
...
EOF
                 8        JMP   9
```

**Figure 21** Use of the fixup table and label table in a single pass code generator for forward jumps.

```
Begin First Pass:                                Label Table
Atom            Loc      Instruction             Label    Value
(ADD,A,B,T1)    4-6
(JMP,L1)        7-8
(LBL,L1)                                         L1       9
...
EOF
Begin Second Pass:
Atom            Loc      Instruction
(ADD,A,B,T1)    4        LOD   R1,A
                5        ADD   R1,B
                6        STO   R1,T1
(JMP,L1)        7        CMP   0,0
                8        JMP   9
(LBL,L1)
...
EOF
```

**Figure 22** Forward jumps handled by a multiple pass code generator.

do both. However, many optimization techniques are capable of improving the object program in both time and space, which is why they are employed in most modern compilers. This results from either the fact that much effort has been directed toward the development of optimization techniques, or from the fact that the code normally generated is very inefficient and easily improved.

The word "optimization" is possibly a misnomer, since the techniques that have been developed simply attempt to improve the generated code, and few of them are guaranteed to produce, in any sense, optimal (the most efficient possible) code. Nevertheless, the word optimization is the one that is universally used to describe these techniques, and we will use it also. Some of these techniques (such as register allocation) are normally handled in the code generation phase and will not be discussed here.

Optimization techniques can be separated into two general classes: local and global. *Local optimization* techniques normally are concerned with transformations on small sections of code (involving only a few instructions) and generally operate on the machine language instructions which are produced by the code generator. On the other hand, *global optimization* techniques are generally concerned with larger blocks of code and will be applied to the intermediate form, atom strings, or syntax trees put out by the parser. Both local and global optimization phases are optional, but may be included in the compiler as shown in Fig. 23, i.e., the output of the parser is the input to the global optimization phase, the output of the global optimization phase is the input to the code generator, the output of the code generator is the input to the local optimization phase, and the output of the local optimization phase is the final output of the compiler. The three compiler phases shown in Fig. 23 make up the back end of the compiler discussed in Section VI, though the global optimization phase could be included in the front end.

In this discussion on improving performance, we stress the single most important property of a compiler—that it preserve the semantics of the source program. In other words, the purpose and behavior of the object program should be exactly as specified by the source program for all possible inputs. There are no conceivable improvements in efficiency which can justify violating this promise.

Having made this point, there are frequently situations in which the computation specified by the source program is ambiguous or unclear for a particular computer architecture. For example, in the expression `(a + b) * (c + d)` the compiler will have to decide which addition is to be performed first (assuming that the target machine has only one Arithmetic and Logic Unit). Most programming languages leave this unspecified, and it is entirely up to the compiler designer, so that different compilers could evaluate this expression in different ways. In most cases it may not matter, but if any of `a, b, c,` or `d` happen

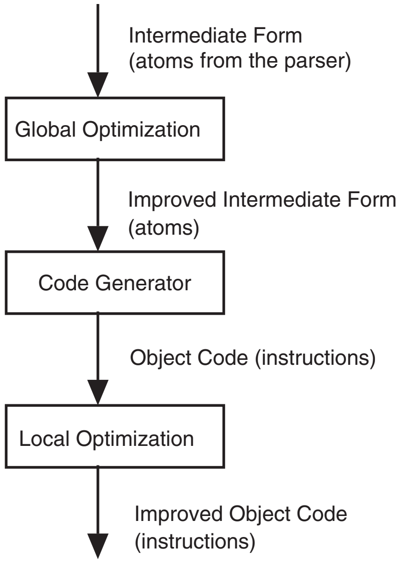


**Figure 23** Sequence of optimization phases in a compiler.

to be function calls which produce output or side effects, it may make a significant difference. Languages such as C, Lisp, and APL, which have assignment operators, yield an even more interesting example:

```
a = 2; b = (a*5 + (a = 3));
```

Some compiler writers feel that programmers who use ambiguous expressions such as these deserve whatever the compiler may do to them.

A fundamental question of philosophy is inevitable in the design of the optimization phases. Should the compiler make extensive transformations and improvements to the source program, or should it respect the programmer's decision to do things that are inefficient or unnecessary? Most compilers tend to assume that the average programmer does not intentionally write inefficient code, and will perform the optimizing transformations. A sophisticated programmer or hacker who, in rare cases, has a reason for writing the code in that fashion can usually find a way to force the compiler to generate the desired output.

One significant problem for the user of the compiler, introduced by the optimization phases, has to do with debugging. Many of the optimization techniques will remove unnecessary code and move code within the object program to an extent that run-time debugging is affected. The programmer may attempt to step through a series of statements which either doesn't exist, or occur in an order different from what was originally specified by the source program.

To solve this problem, many compilers include a switch with which optimization may be turned on or off. When debugging new software, the switch is off, and when the software is fully tested, the switch can be turned on to produce an efficient version of the program for distribution. It is essential, however, that the optimized version and the nonoptimized version be functionally equivalent (i.e., given the same inputs, they should produce identical outputs). This is one of the more difficult problems that the compiler designer must deal with.

Another solution to this problem, used by IBM in the early 1970s for its PL/1 compiler, is to produce two separate compilers. The *checkout compiler* was designed for interactive use and debugging. The *optimizing compiler* contained extensive optimization, but was not amenable to the testing and development of software. Again, the vendor (IBM in this case) had to be certain that the two compilers produced functionally equivalent output.

## B. Global Optimization

As mentioned previously, global optimization is a transformation on the output of the parser. Global optimization techniques will normally accept, as input, the intermediate form as a sequence of atoms (three-address code) or syntax trees. There are several global optimization techniques in the literature—more than we can hope to cover in detail. Therefore, we will look at the optimization of common subexpressions in basic blocks in some detail, and then briefly survey some of the other global optimization techniques.

A few optimization techniques, such as algebraic optimizations, can be considered either local or global. Since it is generally easier to deal with atoms than with instructions, we will include algebraic techniques in this section.

### 1. Basic Blocks and DAG

The sequence of atoms put out by the parser is clearly not an optimal sequence; there are many unnecessary and redundant atoms. For example, consider the C++ statement:

```
a = (b + c) * (b + c) ;
```

The sequence of atoms put out by the parser could conceivably be as shown in Fig. 24.

Every time the parser finds a correctly formed addition operation with two operands it blindly puts out an ADD atom, whether or not this is necessary. In the above example, it is clearly not necessary to evaluate the sum `b + c` twice. In addition, the MOV atom is

```
(ADD, b, c, T1)
(ADD, b, c, T2)
(MUL, T1, T2, T3)
(MOV, T3,, a)
```

**Figure 24** Atom sequence for `a = (b + c) * (b + c);`.

not necessary because the MUL atom could store its result directly into the variable a. The atom sequence shown in Fig. 25 is equivalent to the one given in Fig. 24, but requires only two atoms because it makes use of common subexpressions and it stores the result in the variable `a`, rather than a temporary location.

In this section, we will demonstrate some techniques for implementing these optimization improvements to the atoms put out by the parser. These improvements will result in programs which are both smaller and faster, i.e., they optimize in both space and time.

It is important to recognize that these optimizations would not have been possible if there had been intervening Label or Jump atoms in the parser output. For example, if the atom sequence had been as shown in Fig. 26, we could not have optimized to the sequence of Fig. 25, because there could be atoms which jump into this code at Label L1, thus altering our assumptions about the values of the variables and temporary locations. (The `TST` atom compares `b` and `c` for equality and branches to label `L3` if true. The atoms in Figure 26 do not result from the given C++ statement, and the example is, admittedly, artificially contrived to make the point that Label atoms will affect our ability to optimize.)

By the same reasoning, Jump or Branch atoms will interfere with our ability to make these optimizing transformations to the atom sequence. In Fig. 26 the MUL atom cannot store its result into the variable `a`, because the compiler does not know whether the conditional branch will be taken.

The optimization techniques which we will demonstrate can be effected only in certain subsequences of the atom string, which we call *basic blocks*. A basic block is a section of atoms which contains no Label or branch atoms (i.e., LBL, TST, JMP). In Fig. 27, we show that the atom sequence of Fig. 26 is divided into three basic blocks.

```
(ADD, b, c, T1)
(MUL, T1, T1, a)
```

**Figure 25** Optimized atom sequence for `a = (b + c) * (b + c) ;`.

```
(ADD, b, c, T1)
(LBL, L1)
(ADD, b, c, T2)
(MUL, T1, T2, T3)
(TST, b, c,, 1, L3)
(MOV, T3,, a)
```

**Figure 26** Example of an atom sequence which cannot be optimized.

Each basic block is optimized as a separate entity. There are more advanced techniques which permit optimization across basic blocks, but they are beyond the scope of this article. We use a Directed Acyclic Graph (DAG) to implement this optimization. The DAG is *directed* because the arcs have arrows indicating the direction of the arcs, and it is *acyclic* because there is no path leading from a node back to itself (i.e., it has no cycles). The DAG is similar to a syntax tree, but it is not truly a tree because some nodes may have more than one parent and also because the children of a node need not be distinct. An example of a DAG, in which interior nodes are labeled with operations, and leaf nodes are labeled with operands, is shown in Fig. 28.

Each of the operations in Fig. 28 is a binary operation (i.e., each operation has two operands), consequently each interior node has two arcs pointing to the two operands. Note that in general we will distinguish between the left and right arc because we need to distinguish between the left and right operands of an operation (this is certainly true for subtraction and division, which are not commutative operations). We will be careful to draw the DAG so that it is always clear which arc represents the left operand and which arc represents the right operand. For example, in Fig. 28 the left operand of the addition labeled `T3` is `T2`, and the right operand is `T1`. Our plan is to show how to build a DAG from an atom sequence, from which we can then optimize the atom sequence.

We will begin by building DAG for simple arithmetic expressions. Directed acyclic graphs can also be

| | |
|---|---|
| (ADD, b, c, T1) | Block 1 |
| (LBL, L1) | |
| (ADD, b, c, T2) | Block 2 |
| (MUL, T1, T2, T3) | |
| (TST, b, c,, 1, L3) | |
| (MOV, T3,, a) | Block 3 |

**Figure 27** Basic blocks contain no LBL, TST, or JMP atoms.

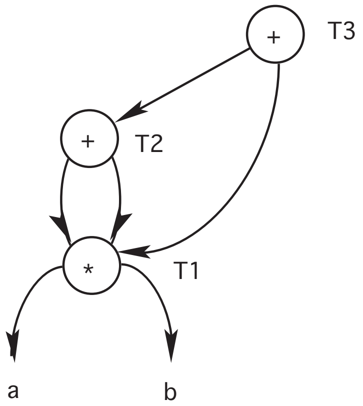


**Figure 28** Example of a DAG.

used to optimize complete assignment statements and blocks of statements, but we will not take the time to do that here. To build a DAG, given a sequence of atoms representing an arithmetic expression with binary operations, we use the following algorithm:

1. Read an atom.
2. If the operation and operands match part of the existing DAG (i.e., if they form a sub-DAG), then add the result Label to the list of Labels on the parent and repeat from step 1. Otherwise, allocate a new node for each operand that is not already in the DAG, and a node for the operation. Label the operation node with the name of the result of the operation.
3. Connect the operation node to the two operands with directed arcs, so that it is clear which operand is the left and which is the right.
4. Repeat from step 1.

As an example, we will build a DAG for the expression a * b + a * b + a * b. This expression clearly has some common subexpressions, which should make it amenable for optimization. The atom sequence as put out by the parser would be:

```
(MUL, a, b, T1)
(MUL, a, b, T2)
(ADD, T1, T2, T3)
(MUL, a, b, T4)
(ADD, T3, T4, T5)
```

We follow the algorithm to build the DAG, as shown in Fig. 29, in which we show how the DAG is constructed as each atom is processed.

The DAG is a graphical representation of the computation needed to evaluate the original expression in which we have identified common subexpressions. For example, the expression a * b occurs three times in the original expression a * b + a * b + a * b. The three atoms corresponding to these subexpressions store results into T1, T2, and T4.

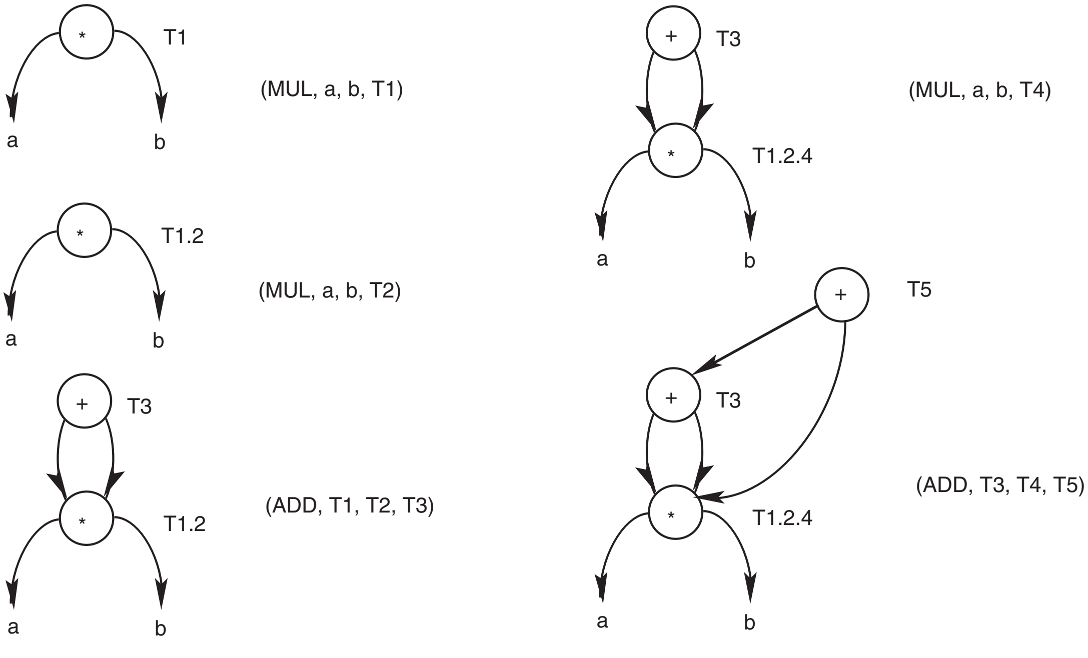


**Figure 29** Building the DAG for a * b + a * b + a * b.

Since the computation need be done only once, these three atoms are combined into one node in the DAG labeled T1.2.4. After that point, any atom which uses T1, T2, or T4 as an operand will point to T1.2.4.

We are now ready to convert the DAG to a basic block of atoms. The algorithm given below will generate atoms (in reverse order) in which all common subexpressions are evaluated only once:

1. Choose any node having no incoming arcs (initially there should be only one such node, representing the value of the entire expression).
2. Put out an atom for its operation and its operands.
3. Delete this node and its outgoing arcs from the DAG.
4. Repeat from step 1 as long as there are still operation nodes remaining in the DAG.

This algorithm is demonstrated in Fig. 30, in which we are working with the same expression that generated the DAG of Fig. 29. The DAG and the output are shown for each iteration of the algorithm (there are three iterations).

A composite node, such as T1.2.4, is referred to by its full name rather than simply T1 or T2 by convention, and to help check for mistakes. The reader should verify that the three atoms generated in Fig. 30 actually compute the given expression, reading the atoms from bottom to top. We started with a string of five atoms, and have improved it to an equivalent string of only three atoms. This will result in significant savings in both run time and space required for the object program.

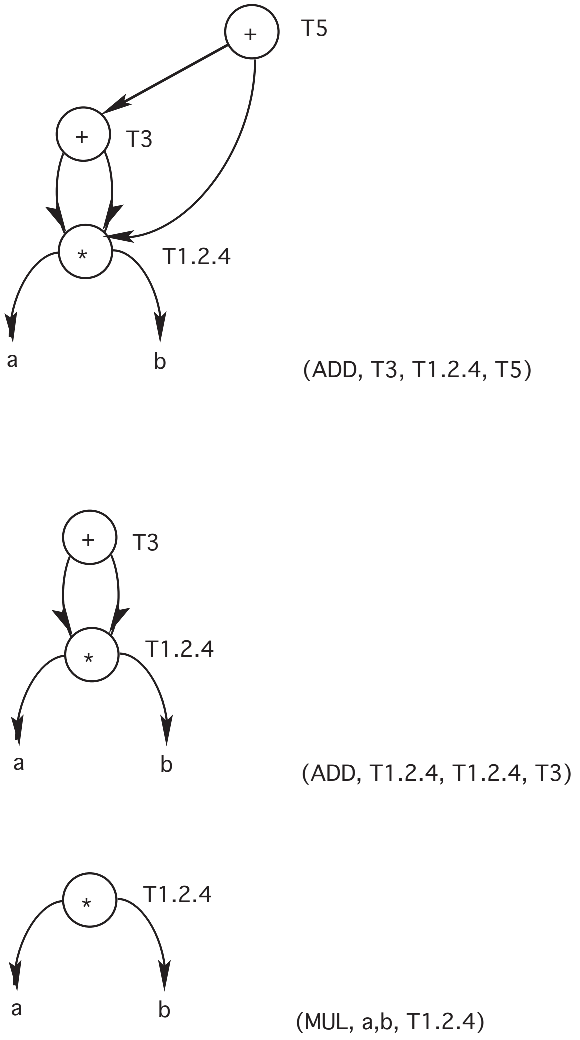


**Figure 30** Generating atoms from the DAG for a * b + a * b + a * b.

## 2. Other Global Optimization Techniques

We will now examine a few other common global optimization techniques, however, we will not go into the implementation of these techniques.

*Unreachable code* is an atom or sequence of atoms which cannot be executed because there is no way for the flow of control to reach that sequence of atoms. For example, in the following atom sequence the MUL, SUB, and ADD atoms will never be executed because of the unconditional jump preceding them.

```
(JMP, L1)
(MUL, a, b, T1)
(SUB, T1, c, T2)    ⇒     (JMP, L1)
(ADD, T2, d, T3)          (LBL, L2)
(LBL, L2)
```

Thus, the three atoms following the JMP and preceding the LBL can all be removed from the program without changing the purpose of the program. In general, a JMP atom should always be followed by a LBL atom. If this is not the case, simply remove the intervening atoms between the JMP and the next LBL.

*Data flow analysis* is a formal way of tracing the way information about data items moves through the program and is used for many optimization techniques. Though data flow analysis is beyond the scope of this article, we will look at some of the optimizations that can result from this kind of analysis.

One such optimization technique is *elimination of dead code,* which involves determining whether computations specified in the source program are actually used and affect the program's output. For example, the program in Fig. 31 contains an assignment to the variable a which has no effect on the output since a is not used subsequently, but prior to another assignment to the variable a. The first assignment is dead code and can be removed from the program.

Another optimization technique which makes use of data flow analysis is the detection of loop invariants. A loop invariant is code within a loop which deals with data values that remain constant as the loop repeats Such code can be moved outside the loop, causing improved run time without changing the program's semantics. An example of loop invariant code is the call to the square root function (sqrt) in the program of Fig. 32.

Since the value assigned to a is the same each time the loop repeats, there is no need for it to be repeated; it can be done once before entering the loop (we need to be sure, however, that the loop is certain to be executed at least once). This optimization will eliminate 999 unnecessary calls to the sqrt function.

The remaining global optimization techniques to be examined in this section all involve mathematical transformations. The reader is cautioned that their use is not universally recommended, and it is often possible, by employing them, that the compiler designer is effecting transformations which are undesirable to the source programmer. For example, the question of the meaning of *arithmetic overflow* is crucial here. If the unoptimized program reaches an overflow condition for a particular input, is it valid for the optimized program to avoid the overflow? (Be careful; most computers have run-time traps designed to transfer control to handle conditions such as overflow. It could be that the programmer intended to trap certain input conditions.) There is no right or wrong answer to this question, but it is an important consideration when implementing optimization.

*Constant folding* is the process of detecting operations on constants, which could be done at compile time rather than run time. An example is shown in Fig. 33 in which the value of the variable a is known to be 6, and the value of the expression a * a is known to be 36. If these computations occur in a loop, constant folding can result in significant improvement in run time (at the expense of a little compile time).

Another mathematical transformation is called *reduction in strength.* This optimization results from the fact that certain operations require more time than others on virtually all architectures. For example, multiplication can be expected to be significantly more time-consuming than addition. Thus, the multiplication 2 * x is certain to be slower than the addition x + x. Likewise, if there is an exponentiation operator, x ↑ 2 is certain to be slower than x * x.

A similar use of reduction in strength involves using the shift instructions available on most architectures to speed up fixed point multiplication and division. An integer multiplication by a positive power of two is equivalent to a left shift, and an integer division by a positive power of two is equivalent to a right shift. For example, the multiplication x*8 can be done faster simply by shifting the value of x three bit positions to the left, and the division x/32 can be done faster by shifting the value of x five bit positions to the right.

Our final example of mathematical transformations involves *algebraic transformations* using properties such as commutativity, associativity, and the distributive property, all summarized in Fig. 34. We do not believe that these properties are necessarily true when dealing with computer arithmetic, due to the finite precision of numeric data. Nevertheless, they are employed in many compilers, so we give a brief discussion of them here.

Though these properties are certainly true in mathematics, they do not necessarily hold in computer arithmetic, which has finite precision and is subject to overflow in both fixed-point and floating-point representations. Thus, the decision to make use of these properties must take into consideration the programs which will behave differently with optimization put

```
{
     a = b + c * d;     //This statement has no effect and can be removed
     b = c * d / e;
     c = b - 3;
     a = b - c;
     cout << a << b << c ;
}
```

**Figure 31** Elimination of dead code.

```
{
     for (i=0; i<1000; i++)
          { a = sqrt (x);          //loop invariant
            vector[i] = i * a;
          }
}
{    a = sqrt (x);               //loop invariant
     for (i=0; i<1000; i++)
          {
            vector[i] = i * a;
          }
}
```

**Figure 32** Movement of loop invariant code.

into effect. At the very least, a warning to the user is recommended for the compiler's user manual.

The discussion of common subexpresssions above would not have recognized any common subexpressions in the following:

```
a = b + c;
b = c + d + b;
```

but by employing the commutative property, we can eliminate an unnecessary computation of `b + c`:

```
a = b + c;
b = a + d;
```

A multiplication operation can be eliminated from the expression `a * c + b * c` by using the distributive property to obtain `(a + b) * c`.

Compiler writers who employ these techniques create more efficient programs for the large number of programmers who want and appreciate the improvements, but risk generating unwanted code for the small number of programmers who require that algebraic expressions be evaluated exactly as specified in the source program.

```
{
     a = 2 * 3;         //a must be 6
     b = c + a * a;     //a * a must be 36
}
{
     a = 6;
     b = c + 36;
}
```

**Figure 33** Constant folding.

## C. Local Optimization

In this section we discuss local optimization techniques. The definition of *local* versus *global* techniques varies considerably among compiler design textbooks. Our view is that any optimization which is applied to the generated code is considered local. Local optimization techniques are often called *peephole* optimization, since they generally involve transformations on instructions which are close together in the object program. The reader can visualize them as if peering through a small peephole at the generated code.

There are three types of local optimization techniques which will be discussed here: load/store optimization, jump over jump optimization, and simple algebraic optimization. In addition, register allocation schemes could be considered local optimization, though they are generally handled in the code generator itself.

The parser would translate the expression `a + b - c` into the following stream of atoms:

```
(ADD, a, b, T1)
(SUB, T1, c, T2)
```

The simplest code generator design as presented in Section VI, would generate three instructions corresponding to each atom: Load the first operand into a register (LOD), perform the operation, and store the result back to memory (STO). The code generator would then produce the following instructions from the atoms:

```
LOD R1,a
ADD R1,b
STO R1,T1 // unnecessary
LOD R1,T1 // unnecessary
```

```
a + b == b + a                  Addition is commutative
(a + b) + c == a + (b + c)      Addition is associative
a * (b + c) == a * b + a * c    Multiplication distributes over addition
```

**Figure 34** Algebraic identities.

```
SUB R1,c
STO R1,T2
```

Notice that the third and fourth instructions in this sequence are entirely unnecessary since the value being stored and loaded is already at its destination. The above sequence of six instructions can be optimized to the following sequence of four instructions by eliminating the intermediate Load and Store instructions as shown below:

```
LOD R1,a
ADD R1,b
SUB R1,c
STO R1,T2
```

For lack of a better term, we call this a *load/store optimization*. It is clearly machine dependent.

Another local optimization technique, which we call a *jump over jump optimization,* is very common and has to do with unnecessary jumps. Often greater efficiency can be obtained by rewriting the conditional logic. A good example of this can be found in a C++ compiler for the statement `if a>b) a = b;`. It might be translated into the following stream of atoms:

```
(TST, a, b,, 3, L1)
(JMP, L2)
(LBL, L1)
(MOV, b,, a)
(LBL, L2)
```

A reading of this atom stream is "Test for a greater than `b`, and if true, jump to the assignment. Otherwise, jump around the assignment." The reason for this somewhat convoluted logic is that the TST atom uses the same comparison code found in the expression. The instructions generated by the code generator from this atom stream would be:

```
     LOD  R1,a
     CMP  R1,b,3  // Is R1 > b?
     JMP  L1
     CMP  0,0,0   // Unconditional
                     Jump
     JMP  L2
L1:
     LOD  R1,b
     STO  R1,a
L2:
```

It is not necessary to implement this logic with two Jump instructions. We can improve this code significantly by testing for the condition to be false rather than true, as shown below:

```
     LOD  R1,a
     CMP  R1,b,4  // Is R1 ≤ b?
     JMP  L1
     LOD  R1,b
     STO  R1,a
L1:
```

This optimization could have occurred in the intermediate form (i.e., we could have considered it a global optimization), but this kind of jump over jump can occur for various other reasons. For example, in some architectures, a conditional jump is a "short" jump (to a restricted range of addresses), and an unconditional jump is a "long" jump. Thus, it is not known until code has been generated whether the target of a conditional jump is within reach, or whether an unconditional jump is needed to jump that far.

The final example of local optimization techniques involves simple algebraic transformations which are machine dependent and are called *simple algebraic optimizations*. For example, the following instructions can be eliminated:

```
MUL R1, 1
ADD R1, 0
```

because multiplying a value by 1, or adding 0 to a value, should not change that value. (Be sure, though, that the instruction has not been inserted to alter the condition code or flags register.) In addition, the instruction (MUL R1, 0) can be improved by replacing it with (CLR R1), because the result will always be 0 (this is actually a reduction in strength transformation).

## SEE ALSO THE FOLLOWING ARTICLES

C and C++ • COBOL • Evolutionary Algorithms • Object-Oriented Programming • Optimization Models • Pascal • Programming Languages Classification • User/System Interface Design

## BIBLIOGRAPHY

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers—Principles, techniques, and tools*. Reading, MA: Addison-Wesley.

Bergmann, S. (1994). *Compiler design—Theory, tools, and examples*. Dubuque, IA: Brown.

Chomsky, N. (1958). Certain formal properties of grammars. *Information and Control,* 2:2.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control,* 8:6.

Lewis, P. M., Rosenkrantz, D. J., and Stearns, R. E. Attributed translations. *Journal of Computer and System Sciences,* 9:3.

Parsons, T. W. (1992). *Introduction to computer construction.* New York: W. H. Freeman.

Pollack, B. W., ed. (1972). *Compiler techniques.* Princeton, NJ: Auerbach.

Rosen, S., ed. (1967). *Programming systems and languages.* New York: McGraw-Hill.

Sammet, J. E. (1969). *Programming languages: History and fundamentals.* Englewood Cliffs, NJ: Prentice Hall.

Wexelblat, R. L., ed. (1981). *History of programming languages.* New York: Academic Press.

# Computer-Aided Design

**George Gustav Savii**
*"Politehnica" University of Timisoara*

## GLOSSARY

**boss** As used in this article, a raised part or protruding ornament on a flat surface.

**Cartesian coordinates** A rectangular system of coordinates to locate points in the model space.

**chamfer** A beveled edge or corner, usually cut at a 45 degree angle.

**fillet** A rounded edge or corner.

**Lambert's cosine law** Relates the amount of reflected light to the cosine of the angle between the incident ray and the surface unit normal at incidence point.

**lead time** The period of time required from the decision to make a product to the beginning of actual production.

**polyline** A string of lines that may contain a number of line segments and/or curve arcs connected together.

**COMPUTER-AIDED DESIGN (CAD)** is a process used mainly in engineering, based on design techniques, using computer systems, as in the creation of complex mechanical parts and equipment, wiring diagrams, buildings, etc. To design means to create a kind of plan to work from when producing something (a house, a car, a dress, an artistic unit, etc.) in order to meet some requirements, especially functional. The plan can contain specification of dimensions, materials, and operations. Computer-aided design can be considered any design that makes extensive use of computer/information technology. More specifically, CAD concerns the productivity tools that facilitate the design process. These tools are used in both parts of the design process: analysis and synthesis.

Design is an intelligent human information-processing activity requiring many skills and much knowledge. Automation of routine design tasks increases productivity of designers and design engineers, thus leaving more time for the creative activities. The ability to design in three dimensions increases the designers conceptual capacity and creativity, improves design quality, and creates a geometric database useful for analysis and production operations. Design databases can be electronically transferred to engineering analysis and simulation programs, which will alleviate costly testing of prototypes, and to manufacturing, improving quality and productivity and reducing lead time. Because electronic databases, rather than drawings, are sent to customers and suppliers, this reduces the costs of producing, storing, and managing blueprints and speeds communications with customers and suppliers.

## I. INTRODUCTION

### A. The Design Process

The design processes are quite different, depending on the object to be created, on the dimension and structure of the company where the process takes place, and the type of the design. Figure 1 presents the structure of a typical design process. The input data for the design process consist of functional requirements and initial design specification for the product to be created and of general and specific knowledge. The output of the design process consists of the final design specification. This specification is used to analyze the design and to predict its performance. To get more realistic results, a physical model can be created by rapid prototyping based on CAD data. The results of the analysis are compared to the functional requirements and a redesign process is used to correct the design specification. This cyclic process continues until satisfactory performance is achieved.

## B. CAD Components and Tools

Computer-aided design systems make use of a number of procedures from software libraries to perform their functions. The main areas that must be covered by the software libraries are numerical methods, geometry, graphics, interfacing, and database management.

## C. History and Trends of CAD Development

The first domains of CAD in which applications were developed were analysis using finite-elements methods, simulation of dynamic systems, and optimization. The development of graphical display systems leads to more graphics-oriented applications (like solid modeling), with user-friendly interfaces and high-quality visualization. Increasing the computing power of processors made possible the integration of useful algebraic manipulations, including usual databases and spreadsheets.

At present, the large design problems require cooperative, and perhaps distributed, design problem solvers, as well as negotiation. Using the resources available on the World Wide Web provides additional challenges. Internet is changing the way professionals collaborate CAD projects. Professionals can work as a team and share CAD applications and documents (models, drawings, libraries, etc.) with others, thousands of miles away. CAD data can be shared in hybrid workflows without the need for explicit translations.

The future of the CAD industry lies mainly in the drawing automation, artificial intelligence techniques, and extensive use of virtual environment technologies for visualization and collaborative working.

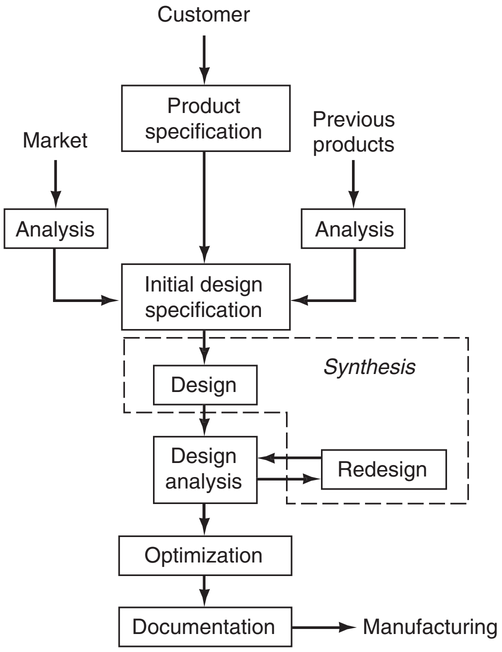


**Figure 1** The design process.

# II. CAD SOFTWARE

## A. Graphics Standards and CAD Data Exchange

There are two main standards groups specific to CAD: for CAD systems development and for CAD data exchange.

The standards for CAD systems development aim to increase the portability of the CAD applications, making them platform (hardware and operating system) independent. The first important standards in this category were Graphics Kernel System (GKS) and Programmer's Hierarchical Interactive Graphics System (PHIGS), describing the functional interfaces between applications and the graphical input/output devices. Of wide use today are OpenGL and DirectX. OpenGL is a cross-platform standard for 3-D rendering and 3-D hardware acceleration. OpenGL is a procedural rather than descriptive interface. DirectX provides a standard development platform for Windows based PCs by enabling software developers to access specialized hardware features without having to write hardware-specific code.

The data exchange standards are intended to make possible the exchange of graphical information between heterogeneous systems. The first important standard in this group has been Initial Graphics Exchange Specification (IGES). A most complex standard is Product Data Exchange Using STEP (PDES), which is application-oriented and contains relevant information for the entire life cycle of a product.

For limited purposes, graphical formats native to common applications are also used for graphical data exchange between homogeneous/heterogeneous systems (DXF, HPGL, etc.).

## B. Coordinate Systems

A coordinate system is a reference system that locates the position of a point. For example, in the 3-D space, a coordinate system gives a triad of numbers that locate a point in space by its distance from three fixed planes that intersect one another. If the planes intersect one another at right angles, the system is called Cartesian orthogonal.

For special purposes, other types of 3-D systems of coordinates can be used: oblique, spherical, and cylindrical. The spherical system is useful for expressing the position of the camera (the viewpoint) with respect to the scene representing a geometric model to be visualized, using longitude and latitude.

### 1. Coordinate Systems in CAD Models

It is possible to define more than one coordinate system in the space of a geometric model of a design. The main, global system is usually called the *model/world coordinate system*. This system is used to store the geometry of the created model. For the user to be able to create new objects in different positions and with different orientations, new coordinate systems with arbitrary position and orientation can be defined. These systems are usually called *working/user coordinate systems*. The visualization process requires another special coordinate system, a bi-dimensional system, for the bi-dimensional surface of the display or paper. This is called the *screen/paper coordinate system*. In addition, coordinate systems are attached to parts and assemblies.

### 2. Coordinate Transformations

The relation between the coordinates of a point in two different systems can be obtained by coordinate transformation. Depending on the relative position of the two systems, there are two basic transformations: *translation* and *rotation*. These two can be combined to produce a complex transformation. A third transformation relates systems with different numbers of dimensions or the projection. For example, a projection is applied to represent a 3-D model onto a 2-D screen.

### 3. Specifying Coordinates

Coordinate values can be entered in two formats: *absolute* or *relative*. The absolute coordinates always have as reference point the origin of the coordinate system. In relative format, the coordinates are measured from the last point entered.

## C. Software Modules

The CAD software applications are usually structured in modules corresponding to the main groups of functions: draw, edit, visualization, data storage and management, system control, and special features.

The *draw module* enables the creation of graphical entities: lines, circles/ellipses, arcs, text, dimensions, symbols, etc. The *edit module* enables changing the existing drawing entities by applying geometric and other types of transformations: move, duplicate/copy, erase, scale, trim, stretch, etc. The *visualization module* performs the display of the design model on the screen and printing/plotting on paper. The *data storage and management module* allows the user to store and manage drawing data. The design can be stored as a set of files on the hard disk; these files can be manipulated in the usual way: copied, moved, deleted, organized in directories and subdirectories, etc. The management of the files includes translation between different data exchange formats. The *system control module* gives the user the possibility to control how the CAD application works, allowing him/her to set the working environment suiting his/her needs. For example, the user can choose the units of measure used, the style for lines and dimensions, screen geometric and color resolutions, etc. The working environment can be saved as a named prototype (template) model/drawing. The *special features* are related to layer creation and management, links to spreadsheets and databases, insertion of new, user developed functions, etc.

Modern CAD applications have modules or connections to modules for engineering analysis (computer-aided engineering, CAE) and modules for manufacturing (computer-aided manufacturing, CAM). This makes the way from model to material product very short: an engineer can model a machine part using CAD modules, then the solid model is transferred into a CAE module for finite-element analysis; if the results are acceptable, the model is transferred into a CAM module that produces a sequence of commands (a computer numerical control (CNC) program) for the machine-tool.

## D. Add-On Programs

To enhance the power of CAD software, there are a number of separate programs available, which work as an extension to accomplish specific tasks. For example, an architectural add-on program brings a rich library of building entities like doors, windows, and special functions, such as for an automatic creation of the roof. A piping design program includes special features to draw pipes and check for interference.

By using powerful specialized modules, general-purpose CAD applications can become specialized systems. Typical examples are the Geographical Information Systems (GIS).

CAD applications have been developed in a wide range of domains. On the market there are also systems for computer-aided design of printed circuit boards, integrated circuits (especially VLSI), communication networks, power networks, complex molecules, drugs, fashion, etc.

## E. Symbol Libraries

An important step in design automation can be made by using symbol libraries. Standardized symbols and other graphic entities (both 2-D and 3-D) that are often used can be stored in special files and directories, representing symbol libraries. The entities in these libraries can be readily accessed and inserted in the current design whenever needed.

As an annex to their catalogs, many manufacturers of add-on equipment (electric, hydraulic, pneumatic, fastening, etc.) offer graphic libraries in electronic format containing the 3-D models and 2-D drawings of their products.

## F. Database Link

CAD applications usually have a module for linking to internal or external alphanumeric databases. The databases can contain attributes of the graphic entities (dimensions, color, weight, manufacturer, dealer, etc.).

## G. Knowledge-Based Processing in CAD

To help the designer in making decisions, knowledge-based modules or applications can be used. Knowledge bases contain expertise found during solving previous projects and from related activities. To date, most knowledge-based systems (also known as expert systems) developed for design have been used to assist the designer in refining a design, and suggesting alterations (diagnostic problem). Expert systems are also useful in choosing the type of elements to be used: motors, transmissions, fastenings, materials, etc. (configuration problem). Abstraction can be used to discover to which general design the current design belongs. Acquisition techniques can request and integrate new design knowledge. Classification can be used to categorize the requirements or the current state of the design in order to decide what sort of method or analysis might be used. Expert systems can also be used for planning the design process, in order to design subcomponents in an appropriate order.

## H. Intelligent CAD

Intelligent CAD programs are based on artificial intelligence (AI) techniques. These programs are not based purely on mathematics as the calculation programs. Beside mathematics, they analyze forms, shapes, arrangement of objects, patterns, colors, etc. They can draw conclusions even if the resulting statements are not completely true or false (using fuzzy logic techniques), can recognize patterns and shapes (using neural networks), and can optimize procedures (using genetic algorithms).

## I. Concurrent Engineering

Concurrent engineering, also known as simultaneous engineering, involves a parallel development across product life cycle activities, from conception through disposal, including quality, cost, schedule, and user requirements, using technologies such as CAD and manufacturing. Using shared databases, customers, designers, and production managers can simultaneously evaluate a proposed product design. Multifunctional teams set product and process parameters early in the design phase. As a result, the design of the product, as it evolves, can incorporate the requirements of the user, special needs for marketing, and any limitations of the production process. Concurrent engineering is designing for assembly, availability, cost, customer satisfaction, maintainability, manageability, manufacturability, operability, performance, quality, risk, safety, schedule, social acceptability, and all other attributes of the product.

Concurrent engineering tends to reduce cycle time and costs when appropriately applied.

# III. GEOMETRIC MODELING

## A. Curves

The point is a zero-dimensional entity, but it is important in CAD because it defines a location in space. A point in a 3-D Cartesian coordinate system can be represented by a triplet $\{x, y, z\}$. Besides the coordinates, a point can have other values associated, such as an identifier and attributes (color, etc.).

A curve is a one-dimensional entity. The most fundamental of all curves is the straight line. A straight line segment can be defined in terms of the end points; for example, in a 3-D coordinate system: $\{x_1, y_1, z_1\}$, $\{x_2, y_2, z_2\}$. A general space curve can be defined in terms of the starting point, variable tangent direction, and curve length.

Next in complexity to the straight line are the quadratic curves: circle, ellipse, parabola, and hyperbola. The basic methods for constructing arcs and circles start from: (1) center point and radius, (2) three points, (3) end points and angle, (4) start point, angle, and radius, (5) two tangents and a point, and (6) three tangents.

The complex shapes of many objects require more complex curves than the quadrics, which are planar. However, the complexity of the shape must not involve a similar complexity in the mathematical expression. Three types of complex synthetic curves are widely used in CAD systems: Hermite cubic splines, Bézier curves, and B-splines. The Hermite cubic spline connects two data (end) points and utilizes a cubic equation defined by the positions and tangent vectors at these points. The parametric equation of a cubic spline segment is given by

$$\vec{Q}(u) = \sum_{i=0}^{3} \vec{C}_i \cdot u^i$$

where $u$ is the parametric dimension ($0 \leq u \leq 1$) and $\vec{C}_i$ are the polynomial coefficients. A Bézier curve is defined from $n + 1$ control points $\vec{P}_i$:

$$\vec{Q}(u) = \sum_{i=0}^{n} \Phi_i(u) \cdot \vec{P}_i$$

where $u$ is the parametric dimension and $\Phi_i$ are the basis (blending) functions. Bézier curves have two major disadvantages: (1) it is very difficult to make the curve pass through any specific points other than the end points and (2) the effect of moving a control point is global.

The B-spline curve is also based on an open polygon with $n + 1$ points, but the blending functions are nonzero only over a range of $k$ control points, being characterized by degree $k-1$. This makes the effect of moving a control point local.

Usually the nodes of the B-spline are equally distributed. To increase the complexity of the generated curves, a nonuniform distribution of nodes is used. If a B-spline curve is defined using basis functions that are algebraic ratios of polynomials, a rational B-spline curve is obtained. If both variations are used, the nonuniform rational B-splines (NURBS) curves are obtained, which are widely used for modeling surfaces of high complexity.

To model complex shapes, polylines are used. These are strings of line and/or curve segments connected together. More complex types of linear entities used in CAD are text, dimension, and hatch patterns. Text is used to write notes, specifications, and to describe the components of a design.

Dimensions are used to indicate size and position (with tolerances). Sometimes they also contain information about surface finishing. An advanced type of dimension is the *associative dimension*. This is used in a mode of dimensioning that automatically updates dimension values when the dimension size (bounding object) is changed. This speeds up the process of adding and editing dimensions in a design.

The hatch patterns are used to represent various materials (especially in cross sections) and finishes, to emphasize portions of the design, etc. The *associative hatching* conforms to its bounding objects so that if the bounding objects are modified, the hatch is automatically adjusted.

## B. Surfaces

A surface is a 2-D entity. A general surface embedded in 3-D can be modeled by dividing it into an assembly of patches. A patch is defined as the basic mathematical element to model a composite surface. Both analytic and synthetic surface entities are used in CAD systems. Analytic entities include plane surface, lofted surface, surface of revolution, and tabulated cylinder. Synthetic entities include the bi-cubic Hermite patches, Bézier patches, B-spline patches, and Coons patches.

Extensively used to design complex shapes, as in car bodies, glassware, shoes, household appliances, etc., are the sculptured surfaces. A sculptured surface, also called free-form surface, is defined as a collection of interconnected and bounded parametric patches together with blending and interpolation formulas. Usually, it is a complex synthetic surface.

### 1. Analytic Surfaces

The simplest of all surfaces is the plane. In building CAD objects, only limited portions of planes are used, the limits being curves. Reasonably complex objects can be built using only planes, resulting in polyhedral or faceted models.

The surface of revolution and the tabulated cylinder are swept surfaces, obtained by moving a curve in a given manner. To obtain a surface of revolution, a planar curve is rotated a given angle about an axis. A tabulated cylinder results from translating a space planar curve along a given direction or from moving a straight line along a given planar curve. The parametric representation of a tabulated cylinder is:

$$\vec{Q}(u,v) = \vec{P}(u) + v \cdot \vec{n}$$

where $\vec{P}(u)$ represents the planar curve and $\vec{n}$ is the cylinder axis.

To design a complex surface that passes through a family of curves, like the fuselage of an aircraft or the bottom hull of a vessel, the lofting method is used. A lofted surface can be defined as:

$$\vec{Q}(u,v) = \sum_i \Phi_i(v) \cdot \vec{P}_i(u)$$

where $\vec{P}_i(u)$ represents the family of defining curves. A ruled surface is a simple lofted surface that interpolates linearly between two boundary curves.

### 2. Synthetic Surfaces

The most popular method in surface modeling is the tensor product method, which defines a patch by mapping a rectangular domain described by the parametric dimensions:

$$\vec{Q}(u,v) = \sum_i \sum_j \Phi_{ij}(u,v) \cdot \vec{P}_{ij}$$

The Bézier and B-spline patches are members of this class, with $\vec{P}_{ij}$ denoting the control points, and using two sets of basis functions, one for each dimension. They are extensively used, for example, in defining the complex shape of the car body. Surfaces that are more complex can be generated by blending two families of curves. The Coons patches are members of this class.

## C. Surface Models

Surface models provide information on the surfaces connecting the object edges. Typically, a surface model consists of curve entities that form the basis to create surface entities. In a mechanical design, each entity is usually represented by a collection of patches.

Despite their similar look, there is a fundamental difference between surface and solid models. Surface models define only the geometry of their corresponding objects and store no information regarding the topology of these objects.

## D. Solid Models

Solid models are advanced representations of real physical objects, also taking into account the content, not only the outline. Therefore, these models have volume, and if given a density the computer can calculate many physical properties, such as mass, center of gravity, and moments of inertia. These calculations can be performed regardless of how irregularly shaped the parts. Computer-aided analyses can also be performed, for instance, the finite-element analysis to determine the strain and stresses.

The major types of solid models are half-spaces, constructive solid geometry (CSG), boundary representation (B-rep), sweep based, enumeration based, analytical solid models (ASM), parametric, and feature based models.

Half-spaces can be considered directed surfaces: each one of them divides the model space into two infinite regions, one filled with material and the other empty. Half-spaces can be combined to construct various solids. Modeling with half-spaces is cumbersome to use and can easily lead to invalid solids.

### 1. Constructive Solid Geometry (CSG) Models

In the CSG models, the geometry of a physical object is stored as a binary tree of Boolean operations applied to a limited set of simple geometric objects (primitives). The nodes represent operations, and the leaves are primitives. The main advantage of these methods is the easiness of making design changes. The main disadvantage is the complexity of the calculations needed to evaluate the tree for graphical representation.

An example of a CSG model for a simple object is presented in Fig. 2. The primitives are a hexahedron (prism) and a cylinder, and the operation is subtraction.

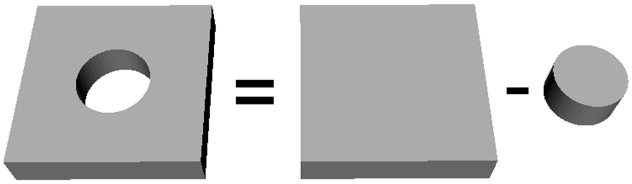


**Figure 2** CSG modeling of a simple object.

## 2. Boundary Representation (B-Rep) Models

A geometric model in which the physical objects are represented by their surfaces is called a B-rep model. Care must be taken regarding validity, self-intersection, and continuity of the surfaces defining a physical object. The B-rep models have the advantage of high speed and high-quality graphical visualization. An example of a B-rep model for the same simple object CSG modeled in Fig. 2 is presented in Fig. 3.

## 3. Sweep-Based Models

The sweep modeling methods are based on an entity of a lower order (e.g., 2-D) that is moved (translated or rotated) in a given manner to produce a higher order (e.g., 3-D) entity. The surfaces are generated by sweeping curves, while the solids by sweeping surfaces. Extrusion (Fig. 4) and revolution (Fig. 5) are particular types of sweeping.

Complex sweeping can be performed with scale change, by joining different end shapes, or along a spiral.

## 4. Enumeration-Based Models

The enumeration models consider the 3-D space split into units of similar form that have a flag attached indicating if the respective unit is occupied by the object. For instance, the octree method uses recursive subdivisions and cubic units. This method is quite rarely used in CAD.

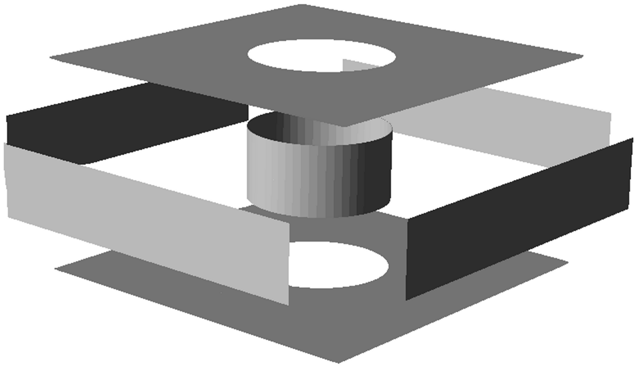


**Figure 3** B-rep model of a simple object.

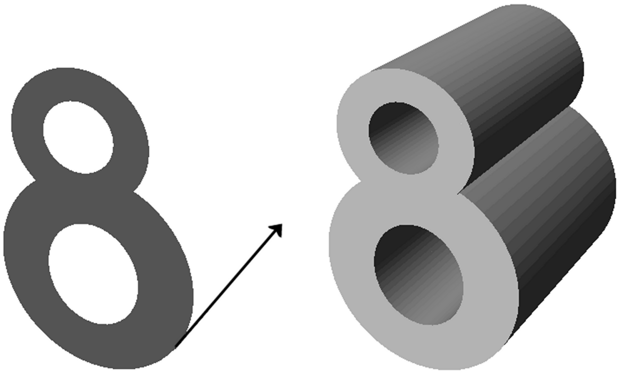


**Figure 4** Model generated by extrusion.

## 5. Analytical Solid Models (ASM)

ASM is an extension of the tensor product method used to represent surfaces. It defines a hyperpatch (parametric solid) by mapping a cubical domain described by the parametric dimensions into the modeling space. An object is represented as an assembly of non-overlapping hyperpatches. Most used are the Bézier and the B-spline hyperpatches. Using Boolean operations, hyperpatches can be combined to model more complex objects.

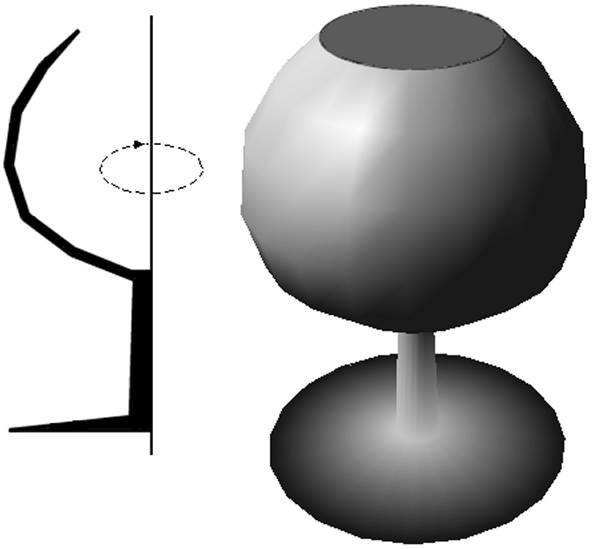


**Figure 5** Model generated by revolution.

### 6. Parametric Models

The parametric models are based on parametric representations of entities. The parameters control the various geometric properties of the entity, such as the length, width, height, radius, etc. Some parametric modelers also allow constraint equations to be added to the models. These can be used to construct relationships between parameters, or impose dimensional or geometric constraints between entities (distances, parallelism, etc.).

### 7. Feature-Based Models

A feature is anything cut from the original piece from which the part will be produced. Examples of features are holes, fillets, chamfers, bosses, and pockets. Feature-based modelers allow features to be associated with specific edges and/or faces. When the reference (edge or face) is moved, the feature moves along with it, keeping the original relationships.

Feature-based models are mainly based on a combination of CSG and parametric models.

## E. Geometric Databases and Data Structures

All the information related to the geometry of a model is kept in special databases. Information about each type of geometric entity is stored using appropriate data structures. For example, the structure for a straight-line segment (usually simply called line) can contain either (1) an identifier, the coordinates of the first end point, and the coordinates of the second end point or (2) an identifier, the coordinates of the first end point, the direction, and the length of the line. Higher order entities can be defined using linked lists of lower order entities. Complex entities may require the use of trees and graphs.

## F. Geometric Transformations

Geometric transformations are needed to give an entity the needed position, orientation, or shape starting from existing position, orientation, or shape. The basic transformations are scaling, rotation, translation, and shear. Other important types of transformations are projections and mappings.

By scaling relative to the origin, all coordinates of the points defining an entity are multiplied by the same factor, possibly different for each axis. Scaling can be also performed relative to an arbitrary point. Mirroring is a special kind of scaling, with one or more scaling factors negative.

By translation, all coordinates of the points defining an entity are modified by adding the same vector quantity.

Rotation is performed by premultiplying the coordinates of the points defining an entity by a special rotation matrix, dependent on the rotation angles.

Shear produces a deformation by forcing contacting parts or layers to slide upon each other in opposite directions parallel to the plane of their contact.

Mappings are complex transformations, in CAD usually consisting of applying a bi-dimensional image on the 3-D surface of an entity. They are used to improve the aspect of modeled objects, adding texture information.

Projections are transformations between systems with different numbers of dimensions. The most important use of projections is for rendering 3-D models on screen or paper (2-D geometric entities). Traditional drafting uses orthographic projections (parallel to one of the coordinate axis). To give the sensation of depth to the rendered scenes, the perspective projection is applied. In such a projection, all lines in the scene that are not parallel to the screen (projection plane) converge in one point for each direction. In the simplest case, all the lines perpendicular to the screen converge in one point.

# IV. GRAPHICS AIDS

The efficiency of CAD activities can be improved using special elements, like layers, grids, snapping, graphic groups, colors, and geometric modifiers. These form the group of graphic aids present in all important CAD systems.

## A. Layers

Data in a project can be more efficiently handled if grouped based on content or reference. For example, in a mechanical design, all auxiliary entities like dimensions and notes can be grouped together. For a complex design, the grouping must be more elaborate, in a hierarchical style. For example, in the case of a building design, data can be grouped by stories, each story having groups for electrical cabling, water, drainage, gas, etc. For each such a group, a *layer* is defined in the design. Each layer can be made visible/invisible, active/inactive (unlocked/locked), deleted, duplicated, etc. In this manner, an engineer can work on the full project, but only having permission to edit entities in the layer he is responsible for. The layers are considered to be transparent.

## B. Grids

A grid of aligned equidistant points can be displayed on the screen, to help the user place the new entities (aligned, for example). Grids also give an idea about the distances between entities, the grid points being defined in the model space. The grid points are usually aligned to the axes of coordinates.

## C. Snapping

An important aid in creating valid entities is snapping. There are two types of snapping: grid and object. In grid-snapping mode, the new points can be created only at grid points. In object-snapping mode, the point locations are indicated using existing graphics entities as a reference. The object-snapping mode depends on the active geometric modifier (end point, center, tangent, etc.).

## D. Geometric Modifiers

To facilitate the graphical input/output operations, a set of commands defining the positioning mode of new entities with respect to the existing ones can be defined. These commands are called geometric modifiers and they influence, for example, the way the object-snapping mode works. Most common modes are end point, midpoint, center, intersection, perpendicular, tangent, nearest, and node.

## E. Graphic Groups

To simultaneously process in the same manner more than one entity, graphic groups can be defined by associating a number of graphic entities. Any action is performed on all group members. The groups can be temporary or permanent.

## F. Dragging

Dragging is a technique of moving (translating) a graphics entity around with a locating device. The image of the entity is moved anchored to the cursor of the locating device. It is used for implementing the entity creation and editing functions.

## G. Colors

To make the CAD model more expressive, the user can choose individual colors for lines and surfaces, as well as for the background. Besides colors, different line types can be used: continuous, dash, dotted, dash-dot, etc. Special line types are normally used for aiding elements, like center lines, construction lines, and hidden lines.

# V. GRAPHICS EDITING

A proper definition of the actions for graphics editing, combined with graphics aids, can increase the efficiency of design activity.

## A. Entity Selection

Before an editing action can be performed, at least one entity must be selected. Main selection modes are single entity selection, selection of all displayed entities, graphics group selection, chain selection (contiguous entities), and window/fence selection. Sometimes it is possible to define filters to be used in entity selection.

## B. Editing Operations

The main editing operations applied to graphic entities are *verification* (displaying and checking the properties of the entities), *duplication* (in a different position/layer/etc.), *array* (rectangular or circular), *measuring* (distances, angles, lengths, etc.), *division* (of a line in segments of equal length, etc.), *offsetting* (creating parallel entities to existing ones), *stretching* (moving a portion/points of an entity and deforming the connecting region), *trimming* (cutting a section from or extending a line or surface to end on another entity), and *property editing* (changing entity attributes: layer, color, etc.).

If the user makes a mistake in issuing a command or in entering any data, or does not accept the result of an operation (drawing, editing, etc.), he can return to the former situation by using a built-in function (commonly known as *Undo*) that reverses the effect of the last command (or set of commands) entered.

## VI. VISUALIZATION OF MODELS

The designer can verify the constructed model, determine spatial relationships, interpret numerical results of analyses, etc., at any stage, by visualization on screen or on paper.

The displayed image of a model is created by mathematical projection using a virtual camera. The user can pan and zoom to display different regions of the design. Each combination of position, orientation, and focal length of the camera defines a view or a viewpoint. Views and viewpoints can be named and stored within the design.

To create engineering drawings, parallel projections (orthographic) and perspective projections (axonometric) are used, in combination with special representations of hidden lines.

The simplest and fastest mode of 3-D model visualization is *vertices,* but it is very seldom used in CAD, not showing any edge or surface. Second in complexity, the *wireframe* mode represents all the edges in the model. The advantage is the speed of visualization, but there is an important disadvantage in the ambiguity of the image.

To obtain more realistic images of the design, a group of techniques known as rendering are used.

### A. Rendering

Rendering denotes the techniques used to create images with a high degree of realism. A first step can be hidden lines and hidden surfaces removal. Next steps can be shading, coloring, shadowing, transparency, mirroring, depth-cueing, texturing, and animation.

To increase the speed of rendering, many functions—like drawing lines, circles, arcs, triangles, color and texture filling—are implemented in the display (video) adapter, forming a graphics accelerator. The accelerators are based on processors (graphics coprocessors) that are specialized for computing graphical transformations, so they achieve better results than the general-purpose main processor of the computer. In addition, they free up the main processor to execute other commands while the graphics accelerator is handling graphics computations.

### B. Hidden Line, Hidden Surface, and Hidden Solid Removal

The first step in adding realism to a scene representing a CAD model is achieved by removing the lines, surfaces, and volumes that are hidden by other entities.

A simple algorithm uses the observation that the surfaces having the normal vector oriented to the back (away from the viewpoint/camera) are not visible. This is based on the convention that the normal vectors are always oriented outward relative to the volume they enclose. The algorithm is therefore called the *backface cull.* Another algorithm is the *painter's algorithm,* which draws the faces on the screen in the order of decreasing distances from the projection plane (screen). These algorithms are limited in the type of faces they can handle.

A more general, but also more complex algorithm, is the *z-buffer* method. The positive $z$ is usually toward the screen (camera). For each pixel on the screen, a memory location (in the $z$-buffer) stores the highest $z$ value found until that moment for the entities drawn at that pixel, and the corresponding color. After all entities are searched, the color in the $z$-buffer is the one corresponding to the nearest point to the camera.

### C. Shading

Assigning a displayed color to each point of an entity depending on its own color and position (normal unit vector), on illumination and on the viewers position, strongly improves the realism of the rendered scene. The simplest shading, *flat-shading,* is obtained by assigning a single color to each polygon (face). By interpolation, a smooth-shaded image can be obtained. In the *Gouraud* algorithm the interpolation across the polygon is applied to shades. An alternative is to interpolate the normal vector across the polygon, which is done in the *Phong* algorithm. Better results produce the *ray tracing* (or *ray cast*) algorithms, but they are more resource-demanding. These algorithms follow the light ray from the pixel on the screen to the light sources or background, taking into account any optical effect, such as reflection, refraction, and dispersion. The best rendering quality can be achieved by *radiosity* methods, which calculate the energy equilibrium conditions for all surfaces in the scene.

Figure 6 presents the model of an object visualized using four different techniques: wire-frame, hidden line/surface, flat shading, and interpolated shading.

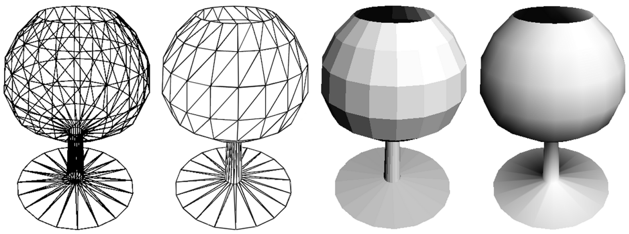


**Figure 6** The model of an object visualized using four different techniques: (a) wire-frame, (b) hidden line, (c) flat shading, and (d) interpolated shading.

## D. Coloring

The color can be characterized by three parameters, usually the *hue* (dominant wavelength), *saturation* (content of pure color), and *intensity* (energy). For practical purposes, there are many color-specifying systems (color spaces): red, green, blue (RGB), cyan, magenta, yellow (CMY), hue, lightness, saturation (HLS), hue, saturation, value (HSV), etc. The RGB system is additive and is used in color displays. The equivalent for printers and plotters is the subtractive system CMY. The HLS system is preferred by artists and designers, being more intuitive. The HSV system is very similar to HLS.

## E. Shadowing

When the viewer's position differs from that of the light source, some regions of the scene are only partially illuminated or not at all, forming shadows. Penumbra contains the partially illuminated regions. The complete shadow (umbra) regions are usually not completely dark, because of the ambient light.

## F. Transparency and Reflection

More realism can be added to the rendered scenes by considering transparency and reflection. Real objects let a part of the incident light travel through, reflect another part, and absorb the rest. Crossing the limit between two transparent materials changes the direction of the ray of light by an amount depending on the ratio of the refraction coefficients of the respective materials. The transparency effect is usually implemented using the *alpha blending*. In this technique, the alpha value for each element drawn reflects the opacity of that object. The displayed pixel is drawn with a color that is a weighted sum of the colors of the contributing objects.

The reflection phenomenon is such that the normal vector to the reflective surface bisects the angle between the incident and emergent light. The light reflected from real objects contains both diffuse and specular components. Lambert's cosine law governs the diffuse reflection. For the specular reflection, an empirical approximation is often used, derived from Lambert's cosine law, mainly by introducing a shininess factor as the power of the cosine function.

## G. Depth-Cueing

Light is partially or totally absorbed when passing through matter, even through air. A simple technique to create the illusion of depth is by depth modulation of light intensity. This method is called *depth-cueing* or *intensity-cueing*. The intensity of each pixel is reduced depending on the distance from viewer to the corresponding point in the model. By adding coloring, a realistic *fog* effect can be obtained.

A more general method applies a *blurring* effect to the scene.

## H. Texturing

A first impression about the material of an object can be given by properly combining transparency (including refraction), mirroring ("specular" effect), and diffusion. More realism can be obtained by applying (mapping) a pattern on object surfaces. Texture mapping adds pictorial details without adding any time-consuming geometry.

The next step is suggesting the irregularities of the surfaces, usually achieved by applying local perturbations. Bump mapping can give the visual illusion of the presence in the surface of small bumps, holes, carvings, engravings, etc., without complicating the geometry by using sculptured surfaces.

A special technique to obtain realistic irregular surfaces is based on fractals.

Figure 7 presents a perspective view of a CAD model of a room, rendered using interpolated shading, textures, and transparency.

## I. Aliasing

Because display devices (screen displays, printers, plotters) are almost all raster devices, presenting an image

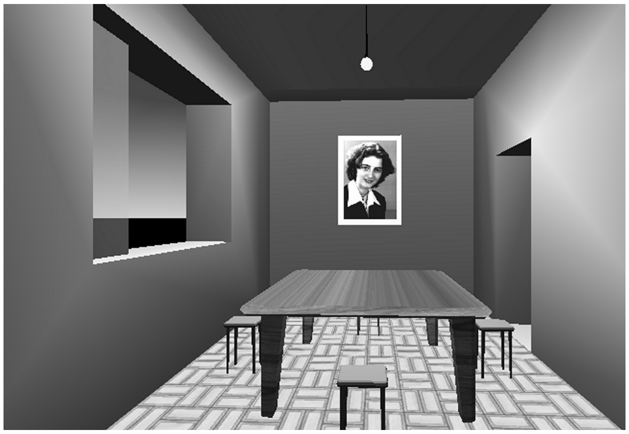


**Figure 7** Perspective view of a CAD of a room, rendered with shading, textures, and transparency.

made of discrete elementary regions (the pixels), some distorting effects can appear, globally known as *aliasing*. Some examples are jagged or stepped diagonal lines and little moving objects that temporary disappear from the screen.

To prevent the negative effects of the aliasing, some *anti-aliasing* methods must be applied. For example, a simple method, often used, is to apply a little blur to the lines. This makes them wider, but smoother.

### J. Plotting

To obtain a hard copy from the design, a plotter or printer is used. The resulting drawings are still needed in not-computer-integrated workshops. Parameters controlling the size and the quality of the plots can be specified. The fist step is composing the drawing layout: specifying the paper size and the views to be plotted along with their position, size, and scale. The second important step is the pen assignment: choosing the association between the plotter pen colors and line types and the model/design colors and line types. For example, the different colors in the computer-internal model (used for on-screen display) can be materialized by different line weights on the paper.

### K. Animation

Animation makes possible a complex examination of the modeled scenes, just like rotating a handheld object, walking around a car, or walking through the rooms of a house. More, the dynamic behavior of complex equipment can be examined before it is materialized. Examination of dynamic behavior implies powerful simulation techniques to generate the data (time-dependent sequences of positions and orientations) needed for animation.

### L. Virtual Reality

Virtual reality technologies provide improved visualization of product by allowing the user to coexist (immersed) in the same space as the product model, therefore gaining a better appreciation of product geometry and aesthetics. It also provides improved interaction with design in terms of more intuitive model manipulation and functional experimentation. The designer can effectively interact with the product model directly rather than using the conventional 2-D mouse and cursor.

### M. Rapid Prototyping

The rapid prototyping (or “desktop manufacturing”) makes possible the creation of a physical model (prototype) starting from the CAD numerical model, usually in 24–48 hours. This model can be used for design analysis and evaluation in the early stages of the product design process, when the changes still involve low costs. In some cases, the prototype can be used directly for manufacturing actual parts, for example, using low-pressure casting, vacuum forming, sand casting, or direct injection molding.

The main rapid prototyping techniques are based on: photopolymerization, laser-cutting the cross-sectional outline in the top layer, transforming a powder in solid objects by fusion using laser beams, melting a thermoplastic filament at given coordinates, gluing powder grains using liquid adhesive, depositing drops of thermoplastic or wax materials, etc. The solidification process is performed on successive thin layers of material.

## VII. ASSEMBLY MODELING

Most manufactured products are assemblies of components. Assembly modeling provides a logical structure for grouping and organizing components into assemblies. Design and engineering teams no longer need to create parts in isolation and hope that when the product goes together, everything fits. They can create parts in the context of the entire product. Assembly design has significant impact on many downstream activities such as process planning, production planning and control, and packaging. The modeling representation of hierarchical relationships and mating conditions are specific to assembly modelers.

## A. Assembly Description and Management

The assembly structure can be represented in a hierarchical assembly tree. The nodes in the tree represent subassemblies and the terminal nodes (the leaves) represent individual parts. An assembly description contains pointers to the individual components used, and constraints used to position the components with respect to one another. Positional constraints refer to orientation and location of the parts, and to relationships between features of different parts. This ensures that a design change is automatically propagated throughout all components of an assembly.

Specific functions of the *assembly manager* module are creation of subassemblies from parts, creation of assemblies from subassemblies and parts, control of the relative placement of parts and subassemblies, regeneration of assemblies and subassemblies after modification, creation of exploded assemblies, and interference and clearance checks.

Because the idea of entire product assembly presumes multiple users, data management has become a critical element of assembly modeling software. One key issue is to alert users to changes made by others that might affect their work.

Assembly analysis may include interference checking, mass properties, kinematic and dynamic analysis, and finite-element analysis.

## B. Mating Conditions

Rather than specifying the transformation matrices needed to place each part in the assembly, mating conditions can be specified. Mating conditions capture and maintain the intended fit of the components. Surface and edge mating constraints are used to locate and orient components with respect to each other. Constraint solving methodologies take the surface/edge mating constraints and automatically transform them into transformation matrices.

# VIII. MASS PROPERTY CALCULATIONS

Based on the fact that a 3-D solid contains a complete description of the geometry of an object, it is possible to obtain additional information from the graphical database. This can refer, for example, to area, volume, centroid (center of mass), moment of inertia, etc. All these are integral proprieties, i.e., calculable by integration. These properties are usually needed for some other activities in design, such as calculating the quantity of paint to apply to the surface of an object, the quantity of material needed to manufacture a part, or performing dynamic simulation.

For a usual representation of a general line, $\vec{P}(s)$, the length of the line is computed as the integral

$$L = \int_a^b \sqrt{\vec{P}\,' \cdot \vec{P}\,'} \cdot ds$$

where $\vec{P}'$ denotes the derivative with respect to $s$, and $a$ and $b$ are the values of the parametric coordinate $s$ for the line end points.

To determine the area of a parametric surface, the integration is performed along both parametric coordinates. To calculate the volume closed by parametric surfaces, the volume integral is usually transformed into a surface integral.

The mass of a general solid is calculated by integration over all elementary volumes of known density. The center of mass can be calculated by weighted integral of the elementary masses, the weighting factors being the positions of elementary masses. A weighted integral of elementary masses can also be used for calculating moments of inertia, with weighting factors being second-order function of elementary mass positions.

# IX. FINITE-ELEMENT MODELING AND ANALYSIS

In the analysis part of a design, it is often required to obtain information about the planar or spatial distribution of field variables. The equations describing the behavior of the field are partial differential equations with some imposed boundary conditions. The complexity of the equations requires some approximating but simpler methods for solving. The finite-element method (FEM) is a numerical analysis technique for obtaining approximate solutions to engineering problems.

## A. Phases of the Finite-Element Analysis

The first step for a finite-element analysis (FEA) is to approximate the region of interest with finite elements defined by nodes. This is achieved by properly subdividing the region. The shape of the finite elements (triangle or quadrangle in 2-D, tetra-, penta-, or hexahedron in 3-D, etc.) is chosen to allow the use of a simple interpolation function to approximate the distribution of a dependent variable within an element. In the second step, the variation of the field variable is approximated within each element by a polynomial and the equilibrium equations for each element are written. Next, the equations for all elements in the region are assembled, resulting in a set of algebraic equations to be solved for the dependent variables in each node. In the last step, the results are interpreted. The whole problem is equivalent to a minimum energy problem.

## B. Mesh Generation

The collection of nodes represents the finite-element mesh. CAD/CAE systems perform an automatic mesh generation, starting from the shape and approximate dimensions of elements given by the user. The user can enter a hint about the number of nodes on some boundaries. Finer generated meshes produce higher accuracy results, but are more resource-demanding (time and computer memory). This is why, for a given object, the general mesh can be relatively coarse, but in the subregions with problems (high dependent variable gradient, as near the stress concentrators), a finer mesh can be requested. An advanced FEM/FEA program can detect by itself such subregions and proceed consequently.

## C. Imposing Boundary Conditions

To solve the set of partial differential equations, some consistent boundary conditions must be specified. For example, in a stress/strain problem, the conditions are in the form of applied forces/pressures and/or displacements. In a temperature field problem, the boundary conditions can be heat fluxes and/or temperatures.

## D. Lumping External Applied Loads

Because the boundary conditions can be specified only in existing nodes, distributed loads must be transformed in equivalent systems of lumped (concentrated) loads.

Figure 8 illustrates an example of finite element analysis. Figure 8a shows the boundary conditions and the generated mesh. The arrows represent the lumped external load, and the triangles—the blocked nodes (unable to move). In Fig. 8b, the results of the stress analysis using FEM are shown. The brightness is proportional to stress, as shown on the scale.

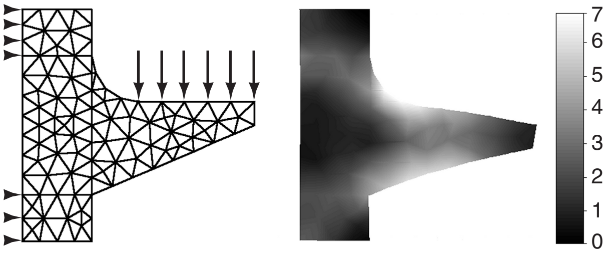


**Figure 8** FEM analysis of a console: (a) boundary conditions and mesh and (b) deformed console with calculated stresses.

## E. Finite-Difference Method

For some types of problems, like analysis of fluid flow or thermal conduction, a simpler method than FEA, but very similar to it, can be applied: the finite-difference method. After mesh generation, partial derivative equations are written for each node, and then the partial derivatives are replaced by finite difference operators, before assembling the equations for all nodes. This leads to a matrix (linear) problem.

## F. Boundary-Integral Method

The boundary-integral method finds the field distribution by determining the appropriate factors for the superposition of a set of characteristic functions. The method involves information related only to the boundary of the region of interest, so the mesh is generated only on the boundary.

# X. USER INTERFACES

User interfaces must facilitate the two-way exchange of information between the user and the software application. There are two kinds of interfaces: textual and graphical.

## A. Hardware

Considering the input functions, there are four logical types of input devices: for text input (e.g., the keyboard), valuator (producing a single numerical value, like a potentiometer), selector (producing a choice from a predefined set), and locator (producing a set of numerical values, like the coordinates of a point, obtained by mouse click).

The physical devices can act as one or more logical devices. Most common interactive graphical devices are keyboard, mouse, graphics/digitizing tablet, joystick, trackball, thumbwheels, and touch-sensitive screen. To work with the six degrees of freedom of an object in the 3-D space used by the CAD models, special devices with more than the usual two degrees of freedom were created.

### B. Language-Based Interfaces

Language-based interfaces work based on messages, i.e., strings of ASCII characters. The messages must conform to a grammar that is specific to each application.

### C. Graphical User Interfaces

The graphical user interfaces (GUI) are based on complex visual communication. Most users consider this type of interface friendlier. The two main components of a GUI are the windows system and the menus system. Usually the window manager and the menu manager are components of the operating system, so the window frames and the menus of all applications on a computer look similar. Some applications offer the possibility to choose the GUI style, for example, to look like running under another operating system.

### D. Windows Managers

Complex models in CAD require simultaneously working on or visualizing several views of the model. This is possible by opening each view in a separate window. All windows can be entirely displayed (tiled), which decreases the size of each window, or can be overlapped, which allows each window to be almost as large as the entire screen. The possibly hidden windows can be restored by mouse-clicking on their respective icons on a toolbar.

### E. Menu Managers

A menu is a list of items from which the user can select. The items in the list can be actions, objects, values, etc. Besides the usual textual menus and the corresponding toolboxes (where text is replaced by icons), the menus can have special graphical appearance: push buttons, radio buttons (array of push buttons, only one active at a time), sliders, even editable input boxes (numeric or text). Usually, in CAD applications the menu system is complex, with items organized in a hierarchical tree. To get easier access to the menu items, most of the menu items can be mapped on the active surface of a graphics (digitizing) tablet.

### F. Error Handling

It is of great importance to have reliable error handling in a CAD system. Even most experienced users make errors while typing or selecting items from menus or entities from the model. This is why a high degree of interactivity is needed for the CAD software. Good software will produce a short description of each error detected and, possibly, a hint to solve the problem. A context-sensitive help is of great value.

### G. Journal Files

A journal file is a record of the commands that have been sent to an application, a history of the input messages. Journal files are useful in debugging and maintenance, helping in tracking the commands that lead to an error situation.

Journal files are also useful in work automation, making possible the extraction of a sequence of commands that is to be used later again, possibly in a repeated manner, to accomplish the same actions. The extracted sequence (usually named *macro*) can be saved in a script file, possibly edited, than independently executed when needed.

## XI. ELECTRONIC COMPUTER-AIDED DESIGN

Computer systems have gained an important place in the design and the production of electronic products. One major field is computer-aided printed circuit board (PCB) design (CAPCBD). The other field is computer-aided design of integrated circuits, especially VLSI.

### A. Computer-Aided Design of Integrated Circuits

Highly complex integrated circuits designs, like VLSI circuits used for microprocessors and memories, have become too difficult to draft manually. Therefore, circuit designers use CAD methods to lay them out. A CAD system can display all or part of a circuit diagram to verify a correct design, change the function of a logic device, simulate the behavior of an integrated circuit while its design is in progress, and store very large diagrams.

There are currently four types of CAD tools that support the physical design of an integrated circuit. The first is the geometric approach, where a general-purpose interactive graphic system is used to create the exact shape of a structure on an integrated circuit mask. The second is the symbolic approach, where details are hidden and the designer works with symbols that the CAD system converts to exact geometries. In the cell-based approach, individual function cells and performance data can be completely characterized and their specifications stored in a computerized cell library. Finally, in the procedural method, cells are automatically placed in a complete integrated circuit layout that uses a procedural language to describe their placement and interconnections.

### B. Computer-Aided Printed Circuit Board Design

Starting from the product proposition, in the system design stage a visual model and product specification are obtained. By detailed design, a functional model is created. With computer-aided PCB design, it's possible to test the PCB on its functionality in an early development stage. After the model is adjusted to work properly, a prototype is built and tested. The design documentation consists of all data such as plot data, drawings, test data, etc. The latest CAD systems deliver the documentation tailored for many kinds of production machines.

An important issue of designing PCBs is fitting in checkpoints (like dimension checks). This can be done automatically by the CAD system.

The main modules of a CAPCBD application are

- Schematic design editor, which enables the designer to input schematic circuits and interactively assigns them to symbol components; the design editor also catalogs schematic circuits in the design database
- Layout editor, which creates board figures, places parts on boards, and wires analog patterns
- Schematic explorer, which allows the user to browse the design database.

The created design data can be processed by a manufacturing data conversion program to create data for automatic PCB manufacturing machines (NC data) used in printed-wiring board manufacture (processing and assembly).

## SEE ALSO THE FOLLOWING ARTICLES

Computer-Aided Manufacturing • Computer-Integrated Manufacturing • Computer Assisted Systems Engineering • Structured Design Methodologies • Systems Design • User/System Interface Design • Virtual Reality

## BIBLIOGRAPHY

Encarnação, J. L. (1990). *Computer aided design: Fundamentals and system architectures,* 2nd revised edition. New York: Springer Verlag.

Farin, G. (2001). *Curves and surfaces for computer aided geometric design,* Fifth edition. Boston: Academic Press.

Kunwoo, L. (1999). *Principles of CAD/CAM/CAE systems.* Reading, MA: Addison-Wesley.

McMahon, C., and Browne, J. (1998). *CADCAM: Principles, practice and manufacturing management.* Reading, MA: Addison-Wesley.

Taylor, D. L. (1992). *Computer-aided design.* Reading, MA: Addison-Wesley.

Zeid, I. (1991). *CAD/CAM theory and practice.* New York: McGraw-Hill.

# Computer-Aided Manufacturing

**Anita Lee-Post**
*University of Kentucky*

## GLOSSARY

**advanced manufacturing technology** Cutting-edge scientific development for practical applications in any production related activities or processes.

**CAM technology** Scientific development in using computers to automate and/or support any production related activities or processes.

**computer-aided production engineering** Using computers to automate and/or support the design of production related activities or processes.

**digital manufacturing** Using computers (discrete signals of 1s and 0s) to execute and control various production related activities or processes.

**manufacturing strategy** Long-term planning that focuses on gaining competitiveness through innovative production.

**virtual manufacturing** Using network technology to operate and manage various production related activities or processes that are not in close physical proximity.

**THE MEANING OF COMPUTER-AIDED MANUFACTURING (CAM)** is first introduced. A brief history of manufacturing in the United States is then reviewed to highlight the strategic role that CAM can play to restore the competitive edge in United States manufacturing. The key enabling CAM technologies are described to give a current picture of how CAM is being successfully applied in United States manufacturing. The article concludes with a depiction of what the future will hold in the next generation of CAM technologies.

## I. INTRODUCTION

Computer-aided manufacturing (CAM) refers to the use of computer technology in the production process. It is the use of computers to program, direct, and control production equipment in the fabrication of manufactured items. Specifically, computer hardware, software, database, and telecommunication technologies are used together with numerical control machines, expert systems, vision systems, LASER, industrial robots, as well as other programmable material handling, storage, and control devices to form an automated, integrated, and flexible manufacturing system. To compete successfully in today's global economy, manufacturing companies realize the importance of continually improving their products by addressing product quality, cost, delivery, design changes, and the like simultaneously. The key to the success of such an undertaking is the application of computer technology to manage, implement, and control the entire production process. With the rapid rate of technological advancement, it is imperative to keep abreast with the current status of computer applications in manufacturing in order to unlock the full potential of these technologies. This chapter is written precisely with such a purpose in mind—to provide the information needed for those interested in exploiting advanced manufacturing technologies for competitive excellence.

## II. MANUFACTURING IN THE UNITED STATES

American manufacturing in pre-1840 was home and guild based, relying on labor-intensive manual operations using primitive technology. Manufacturing took place in small factories powered by unreliable water supplies and seasonally recruited workers from local farms. Production was broken down in distinctive stages, each being performed by craftsmen in specialized trades with no interest in centralizing or simplifying production. Adam Smith's concept of division of labor and the invisible hands of capitalism kept production processes small and fragmented with an emphasis on specialization.

It was not until the first phase of the industrial revolution, characterized by technological innovation and alternative energy source discovery, which brought about the concept of mass production through vertical integration and interchangeable parts. Human labor was substituted by capital innovations that mechanized manual operations to achieve economies of scale. Many previously distinctive operations were centralized. The availability of steam and coal as power supplies freed manufactures from depending on water sources and brought freedom of industrial location. The concept of interchangeable parts further reduced the need for specialized skills. The result was an emphasis on machine instead of people specialization.

Large-scale production facilities for mass production required a mass distribution system of raw materials and finished goods. This was critical for initiating the second phase of the industrial revolution, occurring between 1850 and 1880. The resulting innovation in transportation and communication had made possible high-volume production in modern integrated facilities supported by mass distribution systems of retailers. Big businesses including railroads, steel, oils, etc., with an emphasis on efficiency, high throughput, and low unit cost became commonplace. Further speeding up production, in 1913 Henry Ford sought a nonstop continuous workflow under the concept of a moving assembly line. The potential of economies of scale for producing a single product was fully exploited by the 1920s. However, hindered by the depression of the 1930s and the war of the 1940s, American manufacturing did not reach its global dominance until the 1950s. By this time, though, a lack of competition during this golden era had caused a lax attitude in American manufacturing in regard to production details, product quality, continual improvement, and customer service. In addition, the narrow focus on efficiency turned out to be an obstacle to United States manufacturers' ability to adapt. As a result, from 1960 to 1980 United States companies' manufacturing market share dropped from 99 to 92% domestically and from 25 to 15% globally. During this period, the United States compared unfavorably with many other industrialized countries in various economic indicators such as national income statistics, unemployment rate, and productivity improvements, as shown in Figs. 1–3, respectively. Increased imports during these last decades cost millions of manufacturing jobs and billions of dollars of trade deficit. Numerous attempts were made to address the concern of the United States' decline in manufacturing dominance. Harvard's R. B. Riech described this issue aptly:

> The central problem of America's economic future is that the nation is not moving quickly enough out of high-volume standardization production. The extraordinary success of the half-century of the management era has left the United States a legacy of economic inflexibility. Thus, our institutional heritage now imperils our future.

Doll and Vonderembse in their 1991 article attributed the decline to the difficulty of United States firms to adapt to the changing manufacturing environment. They used a model to characterize the evolution of manufacturing systems as three stages: (1) Craft shops, (2) Industrial systems, and (3) Post-industrial enterprises. The craft shops stage demanded skilled artisans, the shortage of which drove the emphasis of automation in the industrial stage. The view of manufacturing as a functional area focusing on efficiency worked well in the industrial stage but proved to be too narrowly defined for the post-industrial stage. Post-industrial manufacturing, they posited, should be viewed as an enterprise that involved a complex network of interdependent value-added activities from raw material preparation, through fabrication, to assembly, to distribution, and after sale service. A lack of understanding of how best to manage the post-industrial manufacturing enterprise hindered United States firms from adapting adequately to the changing manufacturing environment.

What was the result of United States manufacturers' response to the decline? The third annual *Industry Week* census of manufacturers in December 1999 provided the most up-to-date assessment of the effort taken by United States manufacturers in regard to the situation:

> Manufacturers in the U.S. are struggling in their efforts to improve. Gains are measured in small steps rather than swaggering strides. Corporate executives

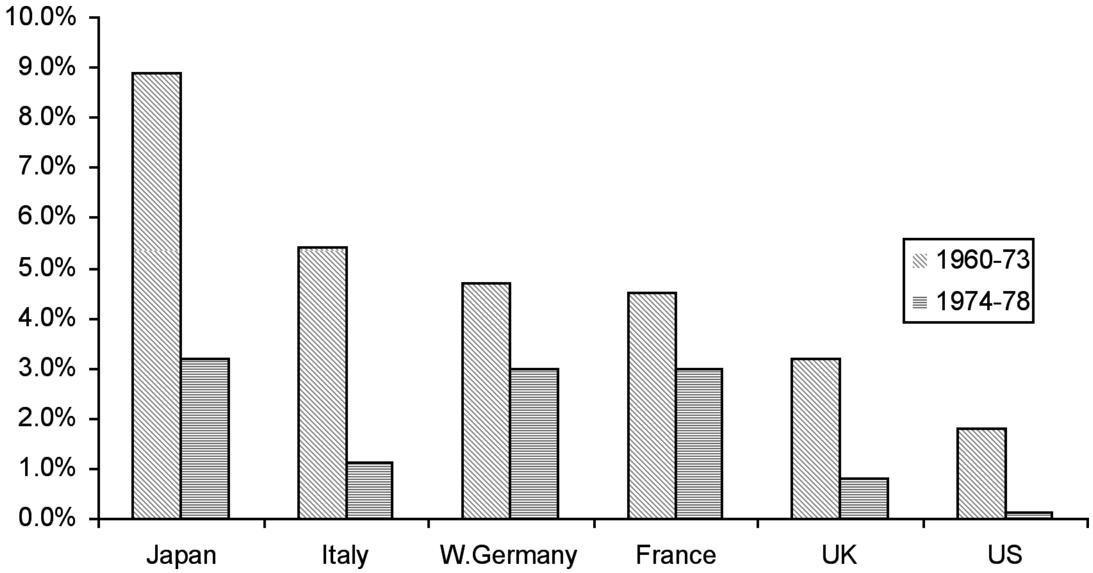


**Figure 1** Percent change in national income per employed person.

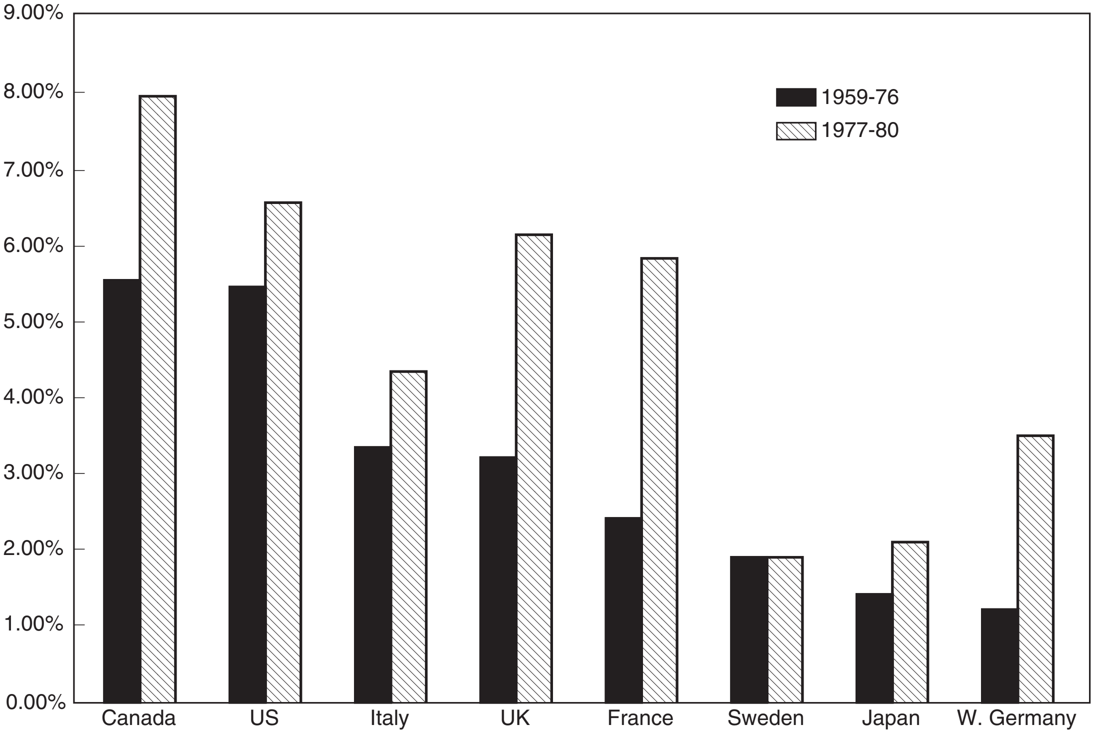


**Figure 2** Average unemployment rates.

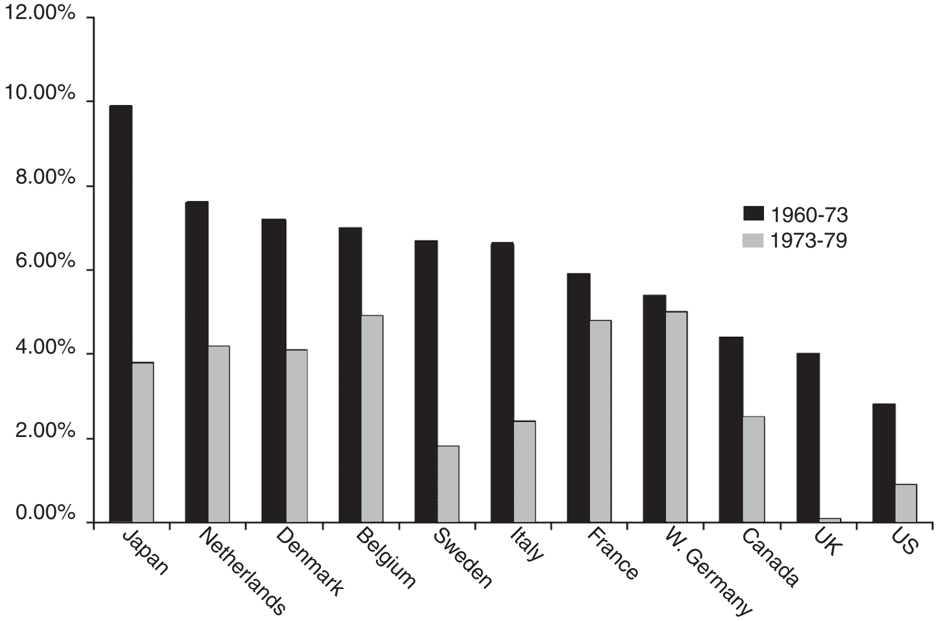


**Figure 3** Comparative productivity improvements.

and plant-level managers do not seem to share the same vision for their companies' futures. New technologies are being thrust in front of organizations faster than many can grasp how best to use them. The quest for excellence in manufacturing is far from over.

## III. THE STRATEGIC ROLE OF CAM

Manufacturing was first recognized as a competitive weapon in a 1956 article by Miller and Roger. Skinner warned that top management often overlooked the potential of manufacturing to strengthen a company's competitive ability. Effective manufacturing allows a company to compete in price, quality, efficiency, delivery, responsiveness, flexibility, innovation, and customer service. CAM provides the means for companies to develop specific manufacturing strategies and capabilities, most notably by combining traditional economies of scale (efficiency wrought by volume) with economies of scope (efficiency wrought by variety) to achieve both flexibility and efficiency. Other reported benefits ascribed to CAM include:

- Greater flexibility
- Faster response to changes in demand and product design
- Greater control
- Repeatability of processes
- Reduced waste
- Faster throughput
- Distributed processing capability
- Product variety
- Small lot sizes

A research conducted by *Industry Week* in July 1999 provided evidence of not only the attainment of aforementioned benefits from computer applications, but also the relationship between the magnitude of these

**Table I** Benefits Related to the Level of Computer Technology Expenditure

| | Heavy spenders<br>% | Light spenders<br>% |
|---|---|---|
| 5-Year productivity increase | 99.5 | 45.5 |
| 5-Year median productivity increase | 60.1 | 28.8 |
| 5-Year manufacturing cost reduction | 27.5 | 19.6 |

benefits and the level of computer technology spending. The findings, summarized in Table I, showed that plants whose computer technology expenditures amounted to 4% or more of the plant budget (heavy spenders) enjoyed greater productivity and cost advantages than plants that spent less than 4% of the budget on computer technology expenditures (light spenders).

Specific benefits offered by CAM are documented in a 1995 report in *Manufacturing Engineering,* as detailed in Table II.

Today manufacturing has reached a post-industrial era that is characterized by increasing complexity, uncertainty, product variety, and rapid development in technology, as well as intense global competition and change. CAM is one of the approaches to manufac-

**Table II** **Specific Benefits Reported by CAM Users**

| Company | Technology | Benefits | Product |
|---|---|---|---|
| Akromold, Cuyahoga Falls, OH | CAD/CAM software from Matra Datavision | • Speeds production of injection and compression molds<br>• Completes toolmaking jobs in two days instead of two weeks<br>• Eliminates the building of support geometry for toolpaths<br>• Eases dealings with customer part files | Automotive, housewares, electronics, and appliance applications |
| Christchurch Hospital, Christchurch, New Zealand | • CAN Wintec machining center<br>• AutoCAD<br>• SurfCAM | • Automates the error prone and labor intensive prostheses production process | Titanium plate prostheses to repair defects in a human skull |
| CNC Products, Highland, NY | • Mastercam software from CNC Software, Inc. | • Generates gouge-free and verified toolpaths<br>• Saves part design, machining, and processing time | Camera and other complex part designs |
| Hendrick Motorsports, Charlotte, NC | • CNC machining center<br>• Coordinate measuring machine<br>• CAM software from Camax: SmartCAM and Camand | • Reduces a 200-hour manufacturing process to 38 hours<br>• Eliminates 15 setups<br>• Reduces processing costs by 70% | Race car engines |
| Star Metal Products | • Machining centers from Fadal Engineering Co.<br>• Virtual Gibbs CAM system from Gibbs & Associates | • Maximizes machine uptime<br>• Reduces complex trigonometric calculations from hours to seconds<br>• Saves hours of prove-out time and eliminates scraps due to programming errors by real-time part rendering<br>• Reduces NC programming time from 55 hours to 3.5 hours<br>• Reduces program run time<br>• Reduces engineering changes time<br>• Reduces programming and machining times from 250 hours to a day | Tooling required for producing nuts and bolts |

turing that enables manufacturing systems to be adaptive, innovative, and efficient.

## IV. CAM TECHNOLOGY

The goal of CAM is to tap into the processing power and speed, storage capacity, and interactive graphical interface of a computer to automate complex production tasks. The strategic incentive is the resulting increased productivity, which will help keep American manufacturing competitive in the global marketplace. It enables the automation and computer support of all the production activities on the shop floor to manufacture parts designed with computer-aided design (CAD) and analyzed with computer-aided engineering (CAE). CAM technologies rely on computers to control and operate equipment such as robots, machine tools, programmable controllers, and machining centers, etc., to produce a variety of parts efficiently and flexibly. The following subsections will take a closer look at some of these CAM technologies. Web sites that provide pictures and/or further explanations of each of these technologies are provided in Table III.

### A. NC, CNC, and DNC

Among the earliest applications of computer technology in manufacturing is numerical control (NC). The first NC machine was developed by Parsons Corp. and Massachusetts Institute of Technology under the support of the United States Air Force in 1954. The machine achieved the desired accuracy by using numbers to move tools in milling turbine blades. Since then, NC has developed into a sophisticated technology that drives modern machining centers and flexible manufacturing systems.

Electronic Industries Association defined NC as: "a system in which actions are controlled by direct insertion of numerical data at some point. The system must automatically interpret at least some portion of this data." A typical NC system is made up of hardware and software that controls machine tools and other production equipment. NC hardware consists of a machine-control unit (MCU) and the machine tool, as shown in Fig. 4. A MCU is made up of a data processing unit (DPU) and a control loop unit (CLU). NC data are read and decoded by the DPU into control signals under the supervision of the CLU. The NC software can be divided into executive software that controls the NC hardware, application software that performs specific machining tasks, and front-end software that serves as a user interface.

NC application software, also called a part program, differs in NC capabilities, modeling features, data translation support, and postprocessing. NC capabilities define the type of manufacturing processes and the degree of movements that a given NC machine can perform. Customized part programs support NC machines for various cutting and fabricating processes such as milling, drilling, turning, electrical discharge machining (EDM), forming, nesting, punching, nibbling, and mold and die production, among others. They also define the "degree of freedom" from two to five axis movements for the NC machines. A 2-axis part program runs NC machines in x and y directions. A 2½-axis allows an additional z direction positioning, but this z direction is not automatically controlled. A 3-axis allows movements in the x, y, and z directions simultaneously. A 4-axis adds rotary movement to the movement in the other three directions. A 5-axis adds spindle or table movement to the 4-axis.

Modeling features of a part program can be defined by its level and type of modeling. Level of modeling refers to the degree of integration between CAD and CAM. Types of modeling include 2D/3D, wireframe, surface, and solid. Solid modeling is quickly replacing surface as the modeling techniques of

**Table III** Web Sites for CAM Technologies

| CAM technologies | URL | Comments |
|---|---|---|
| NC machines | www.bostomatic.com/b_mach.htm | Pictures of NC machines |
| FMS | www.mazak.com/palltech.htm | Pictures of FMS components |
| Expert systems | www.cs.cmu.edu/Web/Groups/AI/html/faqs/ai/expert/part1/faq.html | Frequently asked questions |
| Machine Vision | http://www.vision1.com/gl.html | Glossary of terms |
| Robots | http://www.bricengineeredsystems.com/ | Pictures of industrial robots |
| LASER | http://www.electrox.com/solmain.html | Laser industrial solutions |

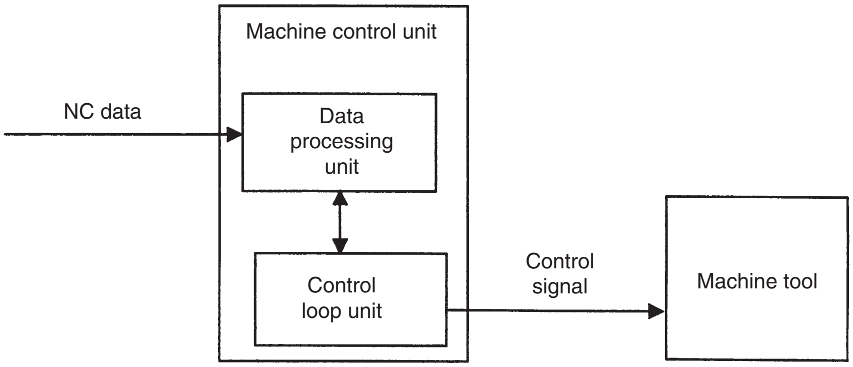


**Figure 4** Components of a typical NC system.

choice. Depending on the modeling features supported by the part program, various machining strategies for a geometrically defined part can be created. They include point-to-point or continuous-path tool path motion, materials selection, machines selection, tooling selection, speeds/feeds selection, and offsets selection.

A part program's data translation support allows data exchange between CAD and CAM. The data translator enables CAD files generated in such standard data exchange format as Initial Graphics Exchange Specification (IGES), Data Exchange Format (DXF), or Standard for The Exchange of Product model data (STEP) to be interpreted. In this way, part geometry from the CAD model can be translated into a form that makes sense for machining applications such as process planning, fixture design and selection, and manufacturability evaluation. This resulting output is called a cutter-location (CL) file.

A part program's postprocessor converts CL data into machine-specific codes called machine control data or "G" codes, which the machine controller understands. The postprocessor can be either customized or generic. A customized postprocessor, called a C-post, is designed for a specific machine tool environment. A generalized postprocessor, called a G-post, offers "universal" postprocessing. Users can use the user configuration file provided to modify the G-post to their specific machining environment.

The development of NC is considered a primary factor in the progress of CAM. In fact, the earlier meaning of CAM was limited to the part programming with its goal to replace skilled machinists in manufacturing metal parts with complex geometry and shape with a programmable machine. Under NC, instructions to control machine tools for grinding, drilling, cutting, milling, punching, bending, and other manufacturing operations are coded on paper or magnetic tapes and later decoded by electromechanical controllers. Manufacturers can quickly program, set up, and run the machine tools used in various cutting and fabricating operations. However, time-consuming and error-prone manual labor is still involved in preparing, loading, and unloading of data tapes as well as controlling individual machines. Improvements in the machine hardware enable the development of machining centers that can perform multiple machining operations. This together with the development of more sophisticated computer technology gave rise to computer numerical control (CNC). Under CNC, several machines or machining centers are under the control of one computer that provides NC program storage, interactive graphical interface, and easy on-machine programming. Further improvements in computer technology, in particular, the development of communication networks, led to distributed numerical control (DNC). Under DNC, various CNC systems are networked together, allowing the gathering and disseminating of both downstream and upstream shop-floor data for an integrated control of machining operations in real time. Future NC machines are expected to include machine intelligence that calls for some kind of planning and adaptive control capabilities.

## B. Flexible Manufacturing

The desire to produce a variety of products in low volume with the ability to accommodate last-minute design changes in a cost-effective manner has prompted the advent of flexible manufacturing systems (FMSs). The convergence of NC, CNC, DNC, as well as the automatic material handling systems such as conveyors, towline with carts, wire-guided carts, and automated guided vehicles (AGVs) provide the technological basis for an FMS. Nof, Bullers and Whinston in 1980 defined an FMS as: "an automated manufacturing system consisting of computer controlled machines linked together with automated material handing systems and capable of simultaneously producing multiple part types." A study by Dimitrov in 1990 reported that approximately 750 FMSs were installed in 26 countries. In the United States, some of the better known companies such as General Electric, Ingresall Milling Machine Company, General Motors, Chrysler, Cadillac, Ford, Hughes, Pratt & Whitney, & Allen Bradley had installed either partial or complete FMSs. In all cases, there have been reported improvements in quality, labor costs, inventory costs, and responsiveness to changes. In fact, FMS is the number one technology among twelve emerging technologies that the United States Department of Commerce identified in 1990 that would create an estimate $1 trillion investment spending by the year 2000.

One of the challenges in FMS installation and operation is the control of the complex network of equipment and shop floor activities of such a system. The National Institute of Standards and Technology (formerly the National Bureau of Standards) proposed a five-level factory control hierarchy: facility, shop, cell, workstation, and equipment, as shown in Fig. 5. The lowest equipment level controls various production equipment including NC machines, robots, conveyors, AGVs, automated storage and retrieval systems, etc. The workstation level provides the integration and interface controls of these equipment. The interaction between workstation and material handling, sequencing, and scheduling of parts through the system are the responsibility of the cell level. The integration of multiple cells and the planning and management of inventory occur at the shop level. Finally at the facility level, manufacturing and business plans are constructed and analyzed to ensure proper management of all production activities within the manufacturing system.

FMS comes with its promise to enable manufacturers to achieve two conflicting goals, namely low volume and low cost production in response to rapid market changes. The key to such promise lies in the way flexibility is pursued in an FMS. Eight types of flexibility have been defined. They are

1. Setup or machine flexibility—the ability to make machine changeover of tools, fixtures, and programs to produce a new set of products
2. Process flexibility—the ability to produce parts in different ways
3. Convertibility or product flexibility—the ability to change the system to produce new products easily and quickly
4. Routing flexibility—the ability to process a part through different routes so that production will not be interrupted in case of machine breakdown, labor shortage, or material unavailability
5. Volume flexibility—the ability to run the system profitably at different production volumes

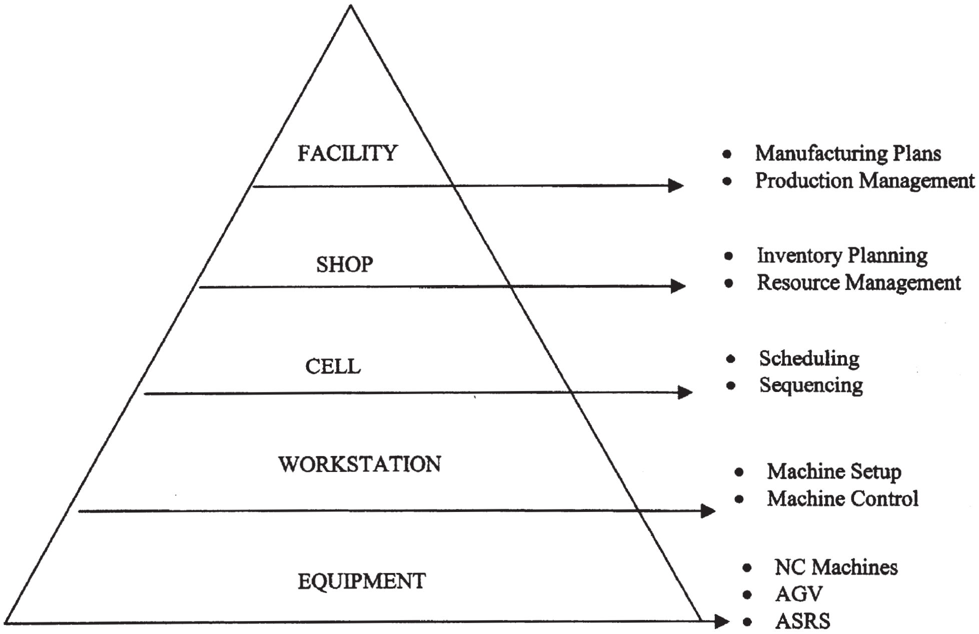


**Figure 5** A control hierarchy for FMS.

6. Expandability flexibility—the ability to cost effectively expand the facilities as needed
7. Operation flexibility—the ability to interchange the order of operations for each part type
8. Production flexibility—the ability to produce a wide range of part types

Upton in 1995 provided the following guideline to manufacturers pursuing flexibility in their factory:

1. Identify the type of flexibility the factory needs
2. Determine the type of workforce or equipment the factory needs to enhance flexibility
3. Define ways to measure the type of flexibility sought
4. Develop a plan to orchestrate the right mix of machines, computer systems, and people

He stressed that flexibility can only be achieved when managers emphasized its importance and practiced it.

## C. Expert Systems

The CAM technologies discussed so far rely primarily on massive data storage, fast retrieval, and powerful computational capabilities of computers to manipulate numeric data using well-defined algorithms. Unlike these technologies, expert systems focus on heuristic reasoning of symbolic data that represent knowledge of the problem domain and human experts to solve difficult problems that require special expertise. According to the British Computer Society's Specialist Group on Expert Systems, an expert system is regarded as:

> . . . the embodiment within a computer of a knowledge-based component, from an expert skill, in such a form that the system can offer intelligent advice or make an intelligent decision about a processing function. A desirable additional characteristic, which many would consider fundamental, is the capability of the system, on demand, to justify its own line of reasoning in a manner directly intelligible to the inquirer. The style adopted to attain these characteristics is rule-based programming.

Waterman gave the following description on the structure of expert systems, as shown in Fig. 6:

> An expert system contains a knowledge base, a dialog structure and an inference engine, which consists of an interpreter and a scheduler. The knowledge base is the collection of the domain knowledge. The dialog structure provides communications and interaction with the user during the operation and processing of the expert system. The inference engine contains the general problem solving knowledge. The interpreter makes decisions on how to apply the rules for inferring new knowledge and the scheduler prioritizes the rules in the appropriate order.

Building expert systems involves the following steps:

1. Knowledge acquisition whereby the knowledge and methods used in problem solving are captured from the knowledge source

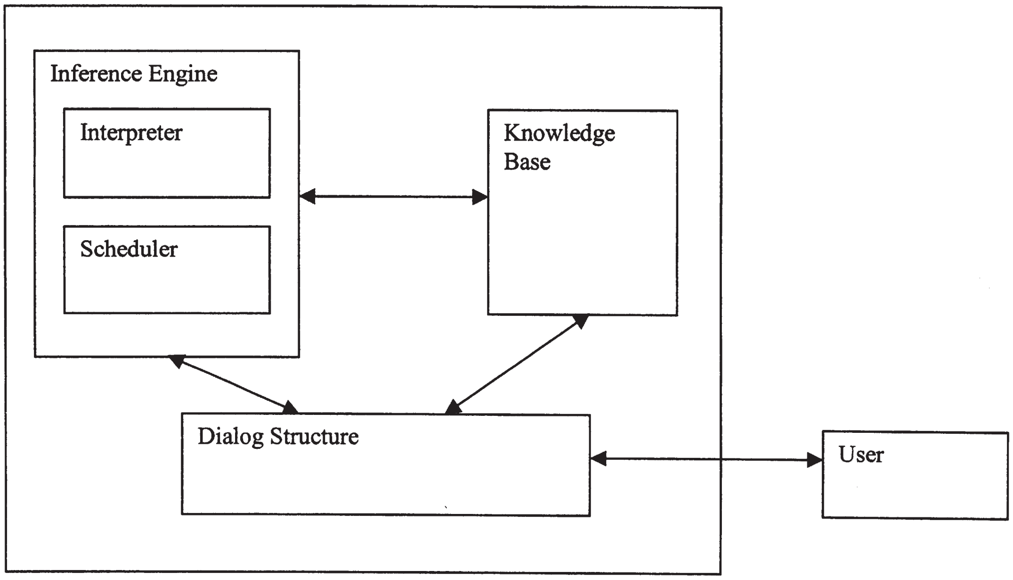


**Figure 6** Structure of an expert system.

2. Knowledge representation whereby the knowledge acquired is stored and organized in a form that the computer understands
3. Knowledge verification and validation whereby the knowledge acquired and represented is proved to be accurate and useful in problem solving

Expert system development tools such as expert system shells have facilitated the building of expert systems. Indeed, as Fig. 7 indicates, there was an exponential growth in the number of expert systems developed from 1980 to 1992.

The early success of using expert systems in planning, medical diagnosis, military analysis, geological exploration, financial consultation, etc., has drawn interests in developing expert systems for manufacturing. A wide range of manufacturing applications, from NC program generation, facility layout, FMS simulations, production scheduling and rescheduling, quality control, and process planning to production control has been reported. A growing number of industries are now applying expert systems in manufacturing, including automobile, glass, oil, aerospace, computers, and electronics, to name a few. Companies that invested over $100 million in expert systems and achieved results include Boeing Computer Services' Connector Assembly Specification Expert, Digital Equipment Corporation's XCON, Hewlett Packard's Agatha system, and Lockheed-Georgia Corporation's GenPlan. Reported benefits of these expert systems include increased speed in completing complex tasks, improved quality, improved decisions, reduced cost, reduced skill labor required, reduced training time, reduced errors, retention of volatile or portable knowledge, and enhanced knowledge sharing and accessibility. Table IV describes a selected sample of expert systems in manufacturing and their related benefits in detail.

Despite numerous examples of successful utilization of expert systems among American companies, a 1991 survey of Fortune 500 companies conducted by Wong, Chong, and Park found that 48% of the 98 respondent companies were not using expert systems because of top management's lack of knowledge about these systems. They suggested that the popularity of expert systems might increase if the systems are properly integrated with operation research techniques, in particular, simulation. These expert simulation systems can boost management executives' confidence in expert systems by allowing a "test run" of the systems in a simulated environment so that the behavior and promises of these systems can be observed or demonstrated.

Besides integrating with simulation, other problem-solving techniques such as optimization and forecasting can be combined with expert systems technology to form hybrid systems that address ill-structured and complex manufacturing problems more effectively and flexibly. Another trend is to have several expert systems embedded within an overall problem-solving system that considers subjective and qualitative types of factors in decision making. The individual expert systems can be smaller, having less than 200 rules, and specializing in solving problems from a particular domain. The results from each expert systems can be shared or aggregated within the overall problem solving system. As the range and capabilities of expert systems are extended, more applications in manufacturing such as forecasting produc-

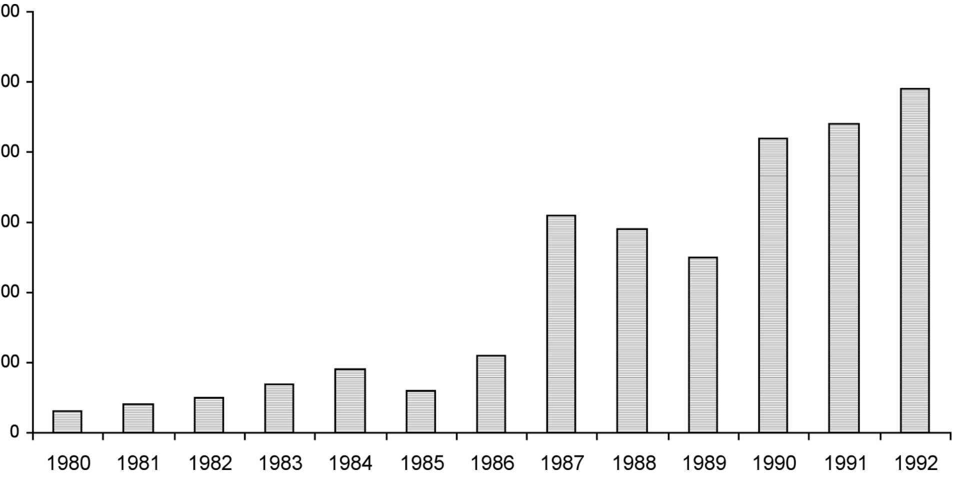


**Figure 7** The number of expert systems developed.

**Table IV** Manufacturing Applications of Expert Systems

| Company | Expert systems | Application | Benefits |
|---|---|---|---|
| American Airlines | MOCA | Schedule airline maintenance | • Provides quality (95% first acceptance rate) and reliable (99% accurate) schedule<br>• Allows maintenance controllers to perform their task effectively<br>• Eliminates maintenance scheduling paperwork<br>• Reduces the number of flight breaks<br>• Increases the average flying time for an aircraft<br>• Increases in flight capacity without an increase in the number of staff members<br>• Generates a saving of $500,000 annually |
| Digital Equipment Corp. | ECAPP | Process planning | • Reduces product introduction times<br>• Generates high quality process plans<br>• Reduces machine lost time<br>• Facilitates communication between design and manufacturing engineers<br>• Enhances responsiveness to dynamic environmental changes |
| Digital Equipment Corp. | XCON, XSEL, XFL | Configure computers at digital to meet customer specification | • Generates tens of millions of dollars in savings annually<br>• Gives customers the maximum flexibility in ordering their computers<br>• Outperforms human in computer configuration<br>• Reduces a three-hour configuration task to 15 minutes<br>• Reduces the number of nonmanufacturable systems from 30 to 1% |
| Hewlett Packard | AGATHA | Test and diagnose computer boards | • Reduces a manufacturing bottleneck<br>• Eliminates diagnosis backlogs<br>• Saves diagnostic cost<br>• Provides a thorough, consistent level of diagnosis continuously<br>• Reduces diagnostic times by 40–80%<br>• Reduces technician training time by 25% |
| Nippon Steel | QDES | Design of steel products | • Reduces design cycle time by 85%<br>• Improves design accuracy by 30%<br>• Generates a saving of $200,000 annually |

tion demand, integrated production planning and scheduling, location planning, and inventory management will be solved collectively by expert systems in the near future.

## D. Computer Vision

Computer vision enables the exploitation of another unconventional capability of a computer, namely, the image processing and pattern recognition ability. The idea of having a computer see and interact with its environment emerged in the late 1950s and 1960s. The high cost of implementation and technical expertise needed slowed down its adoption in manufacturing. It was not until the 1980s before computer vision technology became more widely used for inspection, quality control, dimensional gauging, print verification, code reading, assembly verification, sorting, and machine guidance. The pressure of increasing miniaturization, rapidly advancing technology, intense competition, and rising labor costs encouraged the acceptance of vision technology as a solution to quality, process control, and productivity, especially in the semiconductor and electronics industries. Today, these industries account for half of the vision market. The sophisticated imaging capabilities of vision systems are indispensable in making electronic products. The automotive sector is the next large user of machine vision, accounting for 30% of the 1999 vision market. Automotive applications go beyond the usual focus on productivity and quality issues to part identification. General Motors Corp., for example, uses machine vision to match the right tires with the target vehicle.

A computer vision system consists of components that perform three tasks: image transformation, image analysis, and image understanding, as shown in Fig. 8. Devices such as a camera, photodiode array, charge-coupled, or charge-injection device array capture an image of an object as an analog signal. During the image transformation stage, the analog signal is then converted to a digital form as pixels in a grid of image array, which is stored in the computer memory as a picture matrix. Before the image can be understood by the computer, it needs to go through the image analysis stage whereby the edges of the object are detected by a process called pixel differentiation which essentially converts a picture matrix into a binary matrix of lines. The resulting line drawings allow shape and image interpretation when augmented with additional knowledge about the object such as color, texture, depth, and its environment in the image understanding stage.

The quest for cost-effective quality control, flexibility, and easy product changeover has extended vision capabilities by combining them with sensors and X-rays. Vision sensor systems are well suited in automating basic inspection tasks whereas vision X-ray systems are ideal in inspecting sealed products.

## E. Robots

The need for more flexible machines to automate manufacturing tasks has led to the development of industrial robots. The Robotics Institute of America defines a robot as "a reprogrammable multifunctional manipulator designed to move material, parts, tools, or other specialized devices through variable programmed motions for the performance of a variety of tasks."

A robot typically consists of four components: frame (or arm), tooling (or gripper), power system (or power source), and controller (or keyboard/joystick). The movement of a robot frame can be designed using one or a combination of the four basic configurations: (1) Cartesian has three linear axes, (2) cylindrical has two linear axes and a rotary axis, (3) spherical has one linear axis and two rotary axes, and (4) articulated has three rotary axes. The controller determines the range of movement that a robot takes: from the simplest point-to-point, to the intermediate continuous-path, to the most complex computed trajectory. Depending on the design purpose of a robot, it can be powered by (1) hydraulics for moving heavy loads, (2) pneumatics for high-speed movement, or (3) electricity for precise and quiet motion.

The first industrial robots that appeared in the mid-1960s were simple pick-and-place devices for material transfer. By the 1980s, a new generation of robot systems were commonly used in welding, painting, machine loading and unloading, foundry activities, die casting, injection molding, heat treating, glass handling, product inspection, and material handling. It is estimated that 46,000 industrial robots have been installed in the United States. Automotive and metal-working industries are the biggest robot users in the United States. Robots outperform humans in terms of consistency, predictability, and reliability. They are especially desirable to replace human in hazardous, toxic, strenuous, and poor working situations. Robotic

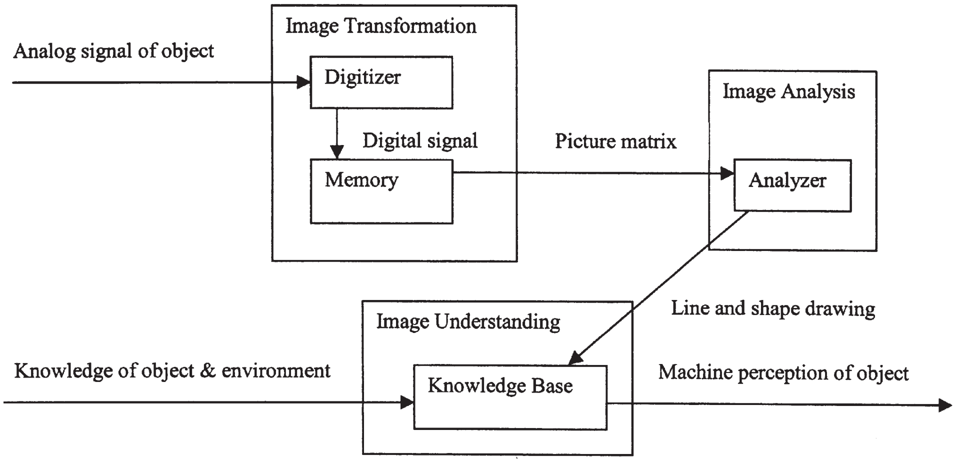


**Figure 8** Components of a computer vision system.

sensors that detect and interpret range, proximity, sound, touch, and force are particularly useful in guiding the robots through unmanned operations, performing delicate gripping and assembly, and positioning parts more accurately and precisely.

An emerging process in building parts called shape deposition manufacturing has become a new area of robotic application under study at Carnegie-Mellon by Kanade, Reed, and Weiss since 1994. Shape deposition builds a part by "growing" it layer by layer following a cross-sectional description of the part structure. The cross-sectional description is generated by slicing a 3-D CAD model into sections. The ease and speed in going from part design to fabrication is the principal advantage of deposition manufacturing. Another advantage is that the process is fixture and tool free, thus making it almost effortless in operating the deposition equipment. The robot's role in shape deposition system is to integrate robotic thermal spraying with deposition. Various materials, including metals, plastics, and ceramics can be sprayed onto patterned surfaces by a robot in layers. The multiple layers, when fused, can be separated from the pattern, forming a shell with the desired shape of the substrate surface. Injection molds, dies, or electric discharge machining electrodes can be fabricated easily in this way. In addition, novel structures or assemblies that are deemed infeasible with conventional manufacturing processes can be made. For example, composites and laminates of metals, plastics, and ceramics can be formed. Assemblies that are of high packing density, composed of selective material deposition within each layer, or made up of components formed and embedded in a single structure can be put together. Industrial applications of robots are sure to grow in other non-traditional areas.

## F. LASER

LASER stands for light amplification by simulated emission of radiation. It is an alternative source of energy in the form of a coherent and concentrated light source converted from electricity. By focusing the energy onto a small area, a LASER can heat up or vaporize the target material. The contributions of LASER lie in three major areas: machining, metrology, and prototyping.

In machining, LASER enables tool-less and fast material processing. Welding, cutting, drilling, heat treatment, and other machining operations are done in a noncontact process. By controlling the power density of a LASER beam, it can work on various materials from organic, to alloy, to refractory metals. Drilling with LASER can produce holes with diameters ranging from 0.005–0.05 inches and precise length-to-diameter ratios up to 100. Welding with LASER produces parts with low-distortion in high speed. Heat treating with LASER is ideal for distortion-free, localized surface hardening to depths of up to 2 mm.

In metrology, the high-precision advantage of LASER makes it especially attractive to gauging, inspection, calibration, interferometry, holography, scanning, and alignment applications. LASER applications in interferometry determine distance or thickness of a part. An original LASER beam is split into measuring and reference beams. These beams are reflected off from their respective reflectors to produce interference fringes. The distance or thickness of the part to be measured is then derived from these fringes. LASER-based inspection allows for in-process scanning for hot-rolled or extruded material. Its simplicity also leads to the development of portable scanning devices. When combined with holography, it can be used to detect defects and produce data on an object's stress or vibration tolerance.

In prototyping, stereolithography allows rapid generation of physical models from a CAD database to facilitate new product development and evaluation. Stereolithography is a layering procedure using ultraviolet LASER beam to solidify the free surface of a liquid photopolymer resin forming cross sections of the object. These cross sections are added layer by layer to produce the solid model for design testing and analysis. Prototyping used to be a time consuming task of highly skilled machinists and technicians who needed to produce a physical item of the product so that its fit, accessibility, and other related aspects of the product could be studied and determined. With the use of LASER in stereolithography, the process to make a physical prototype of a product can be done quickly and inexpensively. More importantly, rapid prototyping has been an enabling technology for the development of a new approach in producing a product called concurrent engineering.

The goal of concurrent engineering is to design and manufacture a product in an integrated simultaneous fashion. With concurrent engineering, various processes involved in producing a product, including design, manufacture, support, maintenance, test, inspection, reliability, safety, and disposability are considered in a systematic instead of serial manner. Rapid prototyping promotes the consideration of these processes at the outset. Physical models are generated based on CAD drawings. These physical models can then be used for prototype evaluation and product performance simulation before actual production can take place. As a result, issues related to product development and manufacture such as design, process, quality, cost, maintenance, and repair can be considered systematically.

## V. CURRENT STATUS OF CAM

The key technologies that help define the current status of CAM include computers, NC machines, and CAD. Computers are the enablers for CAM applications. As computers become faster, cheaper, smaller, accessible, PC-based, graphically interfaced, and multimedia oriented, the same are expected from today's CAM applications. NC machines bring about the need for CAM. The introduction of DNC makes the integration of production with shop-floor control a reality. Today's CAM applications are expected to offer shop-floor programming software as well. For example, Mitron/GenRad's CIMbridge permits assembly line balancing to increase productivity. Orbotech's Xpert assists in balancing fabrication issues to minimize yield loss. CAM has long been the receiving end of model data generated by CAD systems. A bidirectional interface is currently needed so that just as design intent is communicated to the manufacturing side, production constraints can be relayed to the design side. Data reusability is the goal of such CAD/CAM integration. A 3D CAD solid model needs to be geometrically correct, transferable to NC program, and compatible with the company's manufacturing capabilities. It is a question of data exchange rather than model quality. A number of industry and government groups such as the International Electrotechnical Commission, the International Standards Organization, the Japanese Printed Circuit Association, the European Standards Commission, etc., are working together to solve the CAD-to-CAM data exchange problem by establishing formal data formatting standards. At the same time, software vendors are developing integrated CAD/CAM systems, further enhancing the capabilities of today's CAM applications. A look at a sample of today's CAM software provided in Table V gives a glimpse of the current status of CAM technology.

What was the first-hand experience with CAM technologies in United States industry? A factory automation survey conducted by Rosenthal in 1982–1983 provided some insights. He found that a major barrier to the widespread use of CAM was a general lack of understanding by potential users of their needs for factory automation technology. This is in contrast to leading users of CAM whose success in adopting and implementing CAM was contingent on the following:

1. Their understanding of CAM technologies was based on a systemic view of their manufacturing requirements and activities.
2. Their factory automation initiatives investments were evaluated on a broad set of criteria such as product quality improvement, lead time reduction, and new product developments.
3. Their analysis of CAM benefits were driven by long-term strategic rather than short-term productivity implications
4. Their analysis of CAM costs included long-term cost of not adopting these technologies.
5. Their willingness to take risks and proceed in an incremental fashion showed their general feeling that they could not postpone decisions until improved technologies became available.

These findings suggested that manufacturing managers should reflect on their organization's current strategy toward factory automation. They should start with an assessment of the adequacy of their knowledge base toward CAM technologies. Having established a full awareness of their needs for improved production processes, their interests in long-term strategic benefits of CAM technologies should lead to a realistic analysis of both tangible and intangible costs, as well as benefits. Only then can the strategic implications of CAM be fully realized.

## VI. FUTURE DEVELOPMENTS OF CAM

What is next for CAM? Daratech in Cambridge, MA, predicts that worldwide spending for CAM software and services will reach $6.6 billion in 2000, a 16.4% growth from 1999. Computers will continue to advance along their current trend making CAM systems more powerful, affordable, and accessible. Integration and intelligence will be the driving forces behind the next generation CAM systems.

"Open" system architecture will enable full data integration across multiple computer platforms without the need for integrated CAD/CAM software. The gap between design and manufacturing will be bridged. Integration downstream will bring about CAM/NC systems that are feature-based in which both topological characteristics and parametric information of a part design are provided. Users will be able to define and use both design and manufacturing features for both part modeling and machining.

Another form of integration called "virtual factory" will lead a dramatic shift in the environment in which CAM systems operate. According to Upton and

**Table V** CAM Software

| CAM software | Manufacturer | Features |
|---|---|---|
| CAM systems | Bridgeport Machines, Inc., Bridgeport, CT<br>www.machinetoolsonline.com/storefronts/bridge.html | • Windows feature-based system converts part information to a complete manufacturing process plan, including automatic tool selection, speed and feed settings |
| CAD/CAM Version 7.0 | Cimatron Technology Inc., Burlington, ON<br>www.cimatron.com | • Solid modeling capabilities range from wire-frame and surfaces to parametric solids<br>• NURBS surfaces can be created and modified dynamically<br>• 2 ½ to 5-axis milling, drilling, turning, punching, and wire EDM toolpaths generation |
| VERICT | CGTech, Irvine, CA<br>www.cgtech.com | • Multi-axis milling, drilling, and turning<br>• Optimizes NC tool paths automatically<br>• Simulates and optimizes NURBS interpolation |
| OneSpace | CoCreate Software Inc., Fort Collins, CO<br>www.cocreate.com | • A CAD-neutral 3D modeling tool<br>• Supports virtual collaboration in product development |
| CADDS Version 5 | Product Development Company, Needham, MA<br>www.ptc.com/products/cadds/index.htm | • 2 ½-, 3-, and 5-axis machining |
| MasterCAM | CNC Software Inc., Tolland, CT<br>www.mastercam.com | • Intel-based 2 ½-axis milling and drilling package<br>• 5-axis positioning and lathe operation handling<br>• Creates CAD geometry and wire EDM |
| PowerMILL | Delcam, Birmingham, United Kingdom<br>www.delcam.com | • Multi-axis machining<br>• Object linking and embedding for design and modeling |
| Esprit-X | DP Technology, Camarillo, CA<br>www.dptechnology.com | • Toolpath and feed rate calculation<br>• Spindle speed adjustment in 3-D machining |
| Virtual Gibbs CAM | Gibbs & Associates, Moorpark, CA<br>www.gibbsnc.com | • Quick resequencing of operations, tool size modifications, feed and speed changes, and multiple part programming |
| CATIA/Cadam Solutions Version 4 | IBM, White Plains, NY<br>www.catia.com | • Generative machining and assisted manufacturing that captures manufacturing and process know-how and automates repetitive NC functions |
| Helix Engineering | MicroCADAM, Inc., Burbank, CA<br>www.microcadam.com | • Addresses the needs of sheet metal design and fabrication<br>• Flatwrap and flatpattern provides methods for unbending or flattening a surface or solid<br>• Supports surface and solid modeling |
| Pro/Manufacturing Version 14 | Parametric Technology Corp., Waltham, MA<br>www.ptc.com | • Generates tool paths directly from Pro/ENGINEER solid models<br>• Parametric modeling and modification |
| EdgeCAM 4.0 | Pathtrace Systems Inc., Walnut, CA<br>www.pathtrace.com | • Focuses on machining, especially 2- and 4-axis turning, mill-turning, rotary and multiplane milling, family of parts machining, and post processing<br>• Fully integrated CAD/CAM with feature recognition intelligence |

*(continues)*

**Table V** CAM Software *(continued)*

| CAM software | Manufacturer | Features |
|---|---|---|
| I-DEAS Generative Machining | Structural Dynamic Research Corp., Milford, OH<br>www.sdrc.com | • Interactive machining<br>• Feature-based machining helps users improve the manufacturing process, not just tool path creation |
| SurfCAM Version 6.0 | Surfware Inc., San Fernando, CA<br>www.surfware.com | • Machines undercut surfaces in one setup<br>• Parametric design capabilities<br>• Creates optimized tool paths according to user-defined tolerances |
| CAMWorks 99 | Teksoft Inc., Scottsdale, AZ<br>www.teksoft.com | • Generates NC code on solid model<br>• Stores preferences for how particular part features should be machined |
| Unigraphics | Unigraphics Division, Electronic Data Systems, Cypress, CA<br>www.ugs.com | • Variable-axis surface machining<br>• Provides more cutting methods for different geometry<br>• Provides shop-floor oriented programming system (UG/SHOP) |
| VISI-CAM Solid Machining | Vero International, Inc., Gloucestershire England<br>www.vero-software.com | • 2D through 3D surface and solids machining<br>• Provides tool path calculations, offsets, CNC codes, and simulation from solid and surface models |

McAfee, a virtual factory is: "a community of factories each focused on what it does best, all linked by a network that would enable them to operate as one, flexibly and inexpensively, regardless of their location."

The emergence of these "virtual" factories built on a network of business relationships will radically change the way products are manufactured. Companies with different CAD systems, team members from engineering, marketing and management, suppliers, and others can collaborate electronically on viewing, discussing, changing, and documenting a design of a product. Bilateral electronic linkages will be the means for a company to connect with its suppliers and customers. Manufacturers will rely on the Internet to improve their supply-chain efficiency. Companies with different computer systems can exchange information about inventory levels and delivery schedules easily. Using the Internet to collaborate with partners, transmit orders, invoices, payments, or procurement requests will be a common practice. On the customers' front, help desks and other forms of customer service such as order-entry, invoices and/or payments, and relationship management will be handled electronically. The Internet will also be seen as the natural medium for new product development and business forecasting with customers and/or suppliers.

A number of companies such as McDonnell Douglas and Dell Computer are already operating in a virtual manufacturing environment. McDonnell Douglas has built a highly effective virtual factory since mid-1993 linking several thousand suppliers, partners, and customers. This arrangement allows McDonnell Douglas to build prototypes of complex new parts rapidly. It also helps McDonnell Douglas finding the best suppliers quickly through an electronic bidding system. Manufacturing schedules are coordinated so that warnings of time overruns can be issued early when subcontractors are behind on their subassemblies. McDonnell Douglas' virtual factory has made it possible for both long- and short-term partners to collaborate easily, securely, cheaply, and conveniently. Dell Computer has successfully used the Internet to take customer orders on-line and then orchestrated production tailored to each customer's specification, thus, generating an impressive 160% return on invested capital in 1999. Dell viewed one of the changes that was brought about by the Internet as the dramatic reduction in cost of connections and linkages. The near zero cost to communicate with suppliers and customers allows a company to scale quickly, become more flexible, and manage supplier networks and customer relations in a more dynamic fashion. A company can focus on what it does best.

In order to support a virtual manufacturing environment, future CAM systems need to be Internet "ready," focusing on collaborative and communicative functions. For example, the Boeing 777 jet was developed using virtual prototyping involving only electronic instead of physical models. On another plane, a virtual reality system called Virtuosi was developed by a group of researchers at the University of Nottingham to customize clothing design. The system provided a 3-D viewing and manipulation of fashion designs over the Web. Clothes so designed were demonstrated by voice-activated mannequins on virtual runways in the system.

Artificial intelligence research will allow the development of smart CNC controls that would machine a part in the most efficient manner automatically. The more "intelligent" CAM systems can optimize NC tool paths, provide knowledge-based machining, and allow the best practices or process expertise be captured and stored in templates as standards for manufacturing. These fully automated and highly integrated CAM systems permit any part design changes to be rippled through all aspects of the manufacturing processes without human intervention.

Another advance in artificial intelligence research on self-evolving robots will have an impact on CAM. A report in the August issue of the journal *Nature* described how scientists at Brandeis University in Waltham, MA, had successfully created a robotic manufacturing system that designed and built self-evolving and self-generating robot-like machines. Through a self-selection process, the best generations of robot-like machines were created from plastic. The machines were powered by motors and controlled by a neural network on a microchip. Because of the self-evolving nature of these robots, they were essentially designed for free. One of the promises this advancement can bring on CAM is a more economical approach to robotics. The cost of designing and building a robot will be reduced from millions of dollars to just a few thousand dollars. In the future, the use of these inexpensive robots to assemble parts, clean up spills, and perform many other specific tasks in a factory will become a reality.

## VII. CONCLUSION

As the use of computers in manufacturing broadened, CAM users ultimately can study and control manufacturing from product development to production as one seamless process. CAM will soon become synonymous with computer-aided production engineering, digital manufacturing, virtual manufacturing, and digital factory.

An article in the January 2000 issue of *Manufacturing Engineering* reiterated that future CAM software will be smarter, easier-to-use, more affordable, and PC-based. CAM technology will be a natural solution to cost reduction, error reduction, lead-time reduction, and productivity improvement. Advances in computer technology will certainly accelerate the progress of CAM. CAM and other advanced manufacturing technologies will play an increasingly significant role in reshaping a company's manufacturing strategy and restoring our competitive edge through excellence in manufacturing.

## SEE ALSO THE FOLLOWING ARTICLES

Accounting • Computer-Integrated Manufacturing • Management Information Systems • Operations Managements • Productivity

## BIBLIOGRAPHY

Chang, T. C., Wysk, R. A., and Wang, H. P. (1998). *Computer-aided manufacturing,* 2nd edition. Englewood Cliffs, NJ: Prentice Hall.

Ettlie, J. E. (1998). *Taking charge of manufacturing.* San Francisco, CA: Jossey-Bass.

Hopp, W. J., and Spearman, M. L. (1996). *Factory physics: Foundations of manufacturing management.* Boston, MA: Irwin/McGraw-Hill.

Karwowski W., and Salvendy, G., eds. (1994). *Organization and management of advanced manufacturing.* New York: John Wiley & Sons.

Kendall, K. E., ed. (1999). *Emerging information technologies: Improving decisions, cooperations, and infrastructure.* Thousand Oaks, CA: Sage Publications.

Lee, A. (1994). *Knowledge-based flexible manufacturing systems (FMS) scheduling.* New York: Garland Publishing.

Machover, C. (1996). *The CAD/CAM handbook.* New York: McGraw-Hill.

Magaziner, I. C., and Reich, R. B. (1982). *Minding America's business: the decline and rise of the American economy.* New York: Law & Business.

Mital, A., and Anand, S., eds. (1994). *Handbook of expert systems applications in manufacturing: Structures and rules.* London, UK: Chapman and Hall.

Vajpayee, S. K. (1995). *Principles of computer-integrated manufacturing.* Englewood Cliffs, NJ: Prentice Hall.

# Computer Assisted Systems Engineering (CASE)

**Donna Weaver McCloskey**
*Widener University*

## GLOSSARY

**context diagram** A context diagram is a high-level data flow diagram, which shows the external entities with which an information system interacts. It shows the boundaries of the system and the major information flows between the system and the external entities.

**data dictionary** A part of the repository, the data dictionary stores all of the data definitions. This includes field names and aliases, data descriptions, default values, format, range of acceptable values, and other data validation checks.

**data flow diagram (DFD)** A data flow diagram shows the flow of data from external entities, and processes and data stores within an information system.

**entity-relationship (ER) diagram** A diagram used to model the database structure. It contains the fields and table structure along with how the tables relate to one another.

**object** The building block of object-oriented systems. Each object contains data and the functions that can be performed on that data. Once an object is created it can then be reused, thus saving time on future development projects.

**object-oriented (OO) analysis and design** OO analysis and design is a system development methodology that is based on an object orientation. OO modeling techniques closely link and integrate analysis and design activities. UML has become the standard modeling tool for this methodology. System implementation is typically an OO programming language or OO database.

**rapid application development (RAD)** RAD is a system development methodology in which system developers and end users work jointly and in real time to develop the system. RAD makes extensive use of prototyping and promises better and cheaper systems, which can be deployed more rapidly than using the traditional system development life cycle methodology.

**repository** The repository is a centralized database that stores all of the information regarding a system development project, including the data dictionary, diagrams, prototypes, generated code, and notes.

**system development life cycle (SDLC)** The SDLC is a linear and sequential methodology for the development of an information system. While different authors break the steps down into a variety of steps, the main items include preliminary investigation, analysis, design, implementation, and maintenance.

**unified modeling language (UML)** UML is an integrated set of modeling techniques for depicting and constructing an OO information system.

**CASE,** computer assisted (sometimes aided) system (sometimes software) engineering tools help manage, control, and complete the analysis, design, implementation, and maintenance of large information systems projects. A CASE tool is a software application that contains features such as program management tools, diagramming tools, prototyping tools, code generators, and an information repository with the capability to support team developed projects. Rapid application development (RAD) and the increasing use of object-oriented tools have contributed to the growth and acceptance of object-oriented (OO) CASE tools.

## I. INTRODUCTION TO COMPUTER-AIDED SOFTWARE ENGINEERING (CASE) TOOLS

Imagine that you are building your dream home. There are a number of types of professionals and trades people that you will have to work with in completing this job. A lawyer may be necessary to review contracts. An architect will assist you in the design of the house. Eventually carpenters, plumbers, electricians, painters, etc., will be needed to make your dream home a reality. Imagine the planning and coordination that must go into such a project. You can't simply make a decision about carpeting today and have it installed tomorrow. There are a variety of lead times to deal with since many items have to be ordered or made. Scheduling becomes a challenge too. Would you want the walls painted before or after the new carpet is installed? Likewise, the walls can't be finished until some of the electrical and plumbing work has been completed. The coordination and communication necessary for such a large project becomes even more essential when changes and modifications are being made. You could have a meeting with your architect to make some modifications to the house, perhaps to add a jacuzzi. But unless your architect is in close contact with your contractors, the jacuzzi might not be ordered or the appropriate subflooring may not be put in place. Unless the contractor is in close contact with the other subcontractors, the electrician may not add the appropriate wiring for the tub.

Imagine now that your product is not one house but an information system. The concept is the same. The development of an information system involves many people and the coordination and communication among the project team members is crucial for the timely development of a quality product. Since the 1970s developers have used computer-aided/assisted software/system engineering (CASE) tools to help manage, control, and complete the analysis and design of large information systems projects. A CASE tool is a software application that supports various aspects of information system creation. CASE tools have evolved from rather simplistic diagramming tools to advanced tools with an integrated repository that supports the development efforts of multiple users. As outlined in Table I, CASE tools may include a variety of components in addition to a diagramming tool, such as project management tools, prototyping tools, code generators, documentation generators, and quality management tools.

The use of a CASE tool offers many potential benefits to an organization. First, CASE tools can have a direct impact on increasing productivity and the speed with which systems are developed. Although the tools do not replace the critical role played by analysts and developers, they do allow certain activities to be performed more efficiently. The automation of diagram creation and maintenance and the ability to generate code automatically certainly help to increase the speed at which systems can be developed. Many researchers have indeed found that the use of a CASE tool results in increased productivity. Finlay and Mitchell (1994) reported on the introduction of a CASE tool to a British manufacturing company. They found that the use of a CASE tool resulted in productivity gains of 85% and a 70% reduction in delivery times.

Second, a CASE tool provides a means of improving communication. Using a central repository, all of the participants are able to access the same data definitions and standards. The repository also contains all of the diagrams, prototypes, notes, and other documentation associated with the project. Thus, in addition to accessing data definitions and standards, team members can work collaboratively to create and modify work started by other team members. While certainly critical within the development of one project, the repository also provides the opportunity for organizational wide data definitions and standards. Frequently an information system will use data, perform calculations, or provide output that is identical or at least very similar to that created by another organizational information system. Having a central repository that is used by all of the organization's development teams would potentially allow the reuse of components from other projects, thus further improving productivity.

Third, because CASE tool usage results in well-documented analysis and design, it can reduce the time and costs associated with system maintenance. Frequently system documentation is performed at the end of the project, when time and budget constraints might force the activity to get less than optimal attention. By using a CASE tool, the analysis and design activities are formally documented throughout the life cycle. A documentation generator can then be used to assist in producing both system and user documentation in a standard format. Providing quality documen-

**Table I** Common Components of Case Tools

| | |
|---|---|
| Project management tools | Includes tools that help manage and track the project. This may include project scheduling, resource allocation tools, and budget trackers. |
| Diagramming tools | A variety of diagramming techniques are used to model the information system. The traditional system development life cycle approach would involve a context diagram, data flow diagrams, entity-relationship diagrams, and structure charts. An object-oriented approach would involve a different type of diagramming, most commonly UML. Diagramming tools allow the analyst to create, store, and modify the many diagrams used in the analysis phase system development. |
| Prototyping | A prototyping tool allows for the relatively quick and easy development of menus, reports, and forms. It typically does not involve programming or have an associated database. Prototyping allows users to get the "look and feel" of the system early in the analysis and design process. |
| Code generator | Code generators allow users to generate executable code and database definition code from prototypes and other design specifications. Typically CASE tools support a variety of programming languages, including COBOL, C, C++, Visual Basic, and Java. |
| Repository | A central area that stores all of the data concerning a project, including the diagrams, prototypes, data definitions, project management information, notes, and other documentation. The repository can be accessed by multiple users and thus facilitates teamwork. The repository also serves as an important resource when future changes or maintenance becomes necessary. A password system maintains who has access to what data and what their privileges are. |
| Documentation generator | This helps to produce both technical and user documentation in standard formats. |
| Quality management tools | Analyze system models, descriptions, specifications, and prototypes for completeness, consistency, and conformity to the accepted rules of the development methodology. |

tation is a critical component since it is infinitely easier to modify and maintain a well-documented system.

Finally, a CASE tool allows for the integration of analysis, design, and implementation activities. CASE tools provide error checking and analyses for completeness and consistency within and between tools or components. For example, various diagramming techniques have a number of rules and standards, which will be checked by the tool. For example, every data flow in a data flow diagram must have a unique name. The CASE tool would check that this naming convention was being followed for each diagram. Likewise, the CASE tool can notifiy the analyst of potential problems between diagrams. When a change is made to one component part of the project, the tool will either update or flag other areas that are impacted. If the project was being managed using paper files, a change to one part of the project would necessitate the manual checking of other potentially impacted areas. Certainly, the large number of updates and modifications could result in errors. By offering error-checking capabilities and by sharing one repository, the CASE tool allows for the seamless integration of analysis, design, and implementation activities, thus systems are developed with fewer errors. Researchers have indeed found that a CASE tool resulted in improved system quality.

CASE tools provide two different and distinct ways of developing system models. The first, and perhaps most common, is forward engineering. In this way a system analyst develops models that are then refined and eventually transformed into program code. Reverse engineering starts with a system that is then modified and refined. The CASE tool takes program code and then generates the system model from it. Some tools support both forward and reverse engineering.

## II. UPPER AND LOWER CASE TOOLS

CASE tools support both the early stages of development, the analysis and design phases, and the later stages, the implementation and maintenance phases. Therefore, CASE tools can be used by a wide variety of people, including project managers, business and database analysts, software engineers, system developers and designers, programmers, and those people responsible for system maintenance. Frequently the tools used to support the two distinct aspects of system development are differentiated as upper and lower CASE. Some CASE tools focus on one or the other, but most are integrated tools that support system development from beginning to end. An integrated CASE tool, frequently called a cross life cycle

CASE tool, may also have features that are used throughout the project, such as project management and scheduling tools (Table II).

## A. Upper CASE

Upper CASE tools (sometimes also referred to as front end CASE tools) are typically used by analysts and designers and include diagramming tools, prototyping tools that include form, report, and user interface generators, and the repository. Upper CASE tools primarily focus on the analysis and design of the information system.

One of the early steps in the traditional system development life cycle is analysis, the modeling of the logical system. This step would include using a number of diagramming techniques to present the scope of the information system, the individuals and other systems that the new information system will interact with, the flow of data, the processes that are done, and the data that is stored and used. Context diagrams, data flow diagrams, and entity-relationship diagrams are commonly used diagrams to model the information system. A CASE tool allows the analyst to create these diagrams relatively easily and professionally, thus saving the analyst the time required to manually draw and then continually redraw the diagrams as changes and additions are made. The tool can also provide some basic checks to ensure that the correct data modeling rules are followed. For example, a CASE tool will not allow a data flow to go unnamed or for a child data flow diagram to have flows that do not match those is the parent data flow diagram. Analysts would rarely work alone on such large projects. Because the project diagrams are stored in a repository, multiple participants will have access and have the ability to make changes and corrections. The automated diagramming tool helps the analyst maintain the same naming conventions and ensures the interoperability of diagrams. The ability to create and maintain data flow diagrams and the data dictionary have been found to have the highest ranking in critical components for a CASE tool.

Prototyping tools allow the developer to quickly develop screens, reports, and forms to give end users the "look and feel" of the information system. By presenting a prototype to the end users early, the developer can gain valuable feedback and additional support for the project. The process of developing the templates is very straightforward. The establishment of templates allows the developer to set common features, such as headings, footers, function key assignments, buttons, and formatting constraints so that each screen, report, and form are consistent. A change to the shared template would then "ripple through" to each item without the need to edit each one.

## B. Lower CASE

Lower CASE tools (sometimes also referred to as back end CASE tools) support the final stages of the system development life cycle, implementation, and maintenance. Lower CASE tools, which often include code and documentation generators, are typically used by programmers and those people involved in the implementation and maintenance of the information system.

A tool for generating code from prototypes and other design specifications is perhaps the most important tool at this stage and certainly has a direct impact on the speed of implementation. The direct generation of executable source code, without the need for programmers to do the actual coding, offers a number of advantages to the system implementation process. First, the time needed to program, test, and maintain the code can be substantially reduced. Given that the design was accurately and completely entered, error-free code can be directly generated. A

**Table II** Comparison of Upper, Lower, and Integrated Case Tools

| | Upper CASE | Lower CASE | Integrated CASE (or cross life cycle CASE) |
|---|---|---|---|
| Primary function | Primarily used during the initial stages of development, including project identification, analysis, and design | Primarily used during the implementation and maintenance phases | Tools that support the ongoing activities of a system development project |
| Features | Diagramming tools for process, logic, and data models and prototyping tools | Code and documentation generators | Project management and scheduling tools |

change in the design would then require just the regeneration of the code. Most CASE tools support a variety of programming languages, including COBOL, C, C++, Visual Basic, and Java, making it easy to migrate from one platform to another.

Complete documentation is critical to the timely and cost-effective maintenance of the developed system. Researchers have found that system maintenance could take up to 400 times longer with poor documentation, whereas good quality documentation could result in an 80% reduction in system maintenance time compared to average documentation. Frequently, documenting the system is put off until the end of the project, when budget and time constraints could cause the effort to be short changed. The use of a CASE tool ensures that the system is being documented throughout the development process. Documentation generators allow information technology professionals to quickly and easily create both user and technical documentation. Because it was developed throughout the system development life cycle, the documentation is often more complete and of higher quality, thus supporting maintenance activities.

Lower CASE tools have made rapid application development (RAD) possible. RAD is a relatively new system development methodology in which system developers and end users work jointly and in real time to develop the system. Unlike a traditional system development life cycle, it is not a sequential and linear approach but rather a collaborative and integrated effort. Developers no longer work on analysis diagrams in relative isolation. RAD involves the end users directly in the entire development process and makes extensive use of prototyping. Due to the high involvement of end users, one of the advantages of RAD is said to be better systems. The streamlined methodology also results in systems that can be deployed more rapidly than using the traditional development life cycle methodology, which results in lower development costs. CASE tools, particularly the prototyping and code generators, have made RAD a feasible methodology that is gaining more and more support, particularly for electronic commerce applications.

## III. USAGE OF CASE TOOLS

### A. Reasons CASE Tools Are Not Used Effectively

CASE tools have a somewhat tarnished image and adoption has been slower than expected. Kremerer (1992) reports that one year after the introduction of a CASE tool 70% of the organizations were not using them and that in 25% of the organizations only one group was using the tools. Finlay and Mitchell (1994) reported on the introduction of a CASE tool to a British manufacturing company and found that the CASE tool was being used, which had resulted in substantial productivity gains and reduced delivery times. Unfortunately, these benefits were not as great as originally anticipated. Not achieving the level of anticipated benefits is just one of the disappointments that have tarnished the introduction of CASE tools.

CASE tools have gained a reputation of not fulfilling their promise. Early on, CASE tools were oversold as being a "cure all" for system development problems. It was misunderstood that CASE tools would serve an active, rather then passive, role in system development. It is important to remember that CASE tools are to aid or assist in the development process, not to replace the vital role of system developers. The belief that CASE tools would serve as some type of "silver bullet" to magically increase system quality while decreasing development time is as absurd as believing that the use of a word processing package would let anyone write a great novel. While a CASE tool can help speed up several tasks and can offer some checks for quality assurance, it is but a tool that can only be used effectively by those people who have the skills and knowledge to complete the project. In fact, a small study of experienced CASE tool users found that CASE tools were not found to have a positive impact on the most critical factors in successful systems development, including involving the client in the development process and setting appropriate boundaries and scope. The tool cannot correct voids in knowledge or experience. A CASE tool can enhance communication between team members and can help track whether a project is on schedule and on budget, but it cannot correct problems in project management, such as unrealistic budgeting and scheduling. The general misunderstanding of the role CASE tools could play in systems design and development resulted in unmet expectations. This is still a burden that current CASE tools are carrying. To separate themselves from the initial bad press surrounding CASE tools, many organizations are no longer using the term CASE tool, but rather integrated application development tool.

Additionally, CASE tools do not come cheap. A premier integrated CASE tool could run in the area of thousands of dollars per user. And the software is just part of the budget necessary for successful adoption. The addition of hardware and training costs further inflates the price tag. The less than spectacular results of early CASE adoption combined with the high costs of implementation have resulted in a relatively flat adoption curve.

CASE tools are complex and powerful tools that can be used throughout the system development life cycle. Ironically, this very feature has been cited as a reason that CASE tools are not used. The complexity of the tool and the belief that the tool is complicated to work with have been found to be substantial factors in the slow adoption by systems professionals.

The successful adoption of a CASE tool requires use by a group of skilled and trained system developers. Unless the tool is accepted and used by the entire group, many of the advantages will not be realized. For example, the advantage of standard data definitions does not exist if all of the team members are not using the CASE tool. For this reason, the support of management has been found to be a significant factor in determining successful CASE usage.

### B. Current Usage

Because of the vast number of products and their differences in scope and capabilities, it is difficult to assess the overall CASE tools market share. The CASE Tool Index web page (http://www.qucis.queensu.ca/Software-Engineering/tools.html) gives a comprehensive list of current CASE tools. In December 2001, there were well over 300 tools listed. Articles and reviews of particular applications, such as those published at Software Development Online (http://www.sdmagazine.com), published research and CASE studies are useful for examining which tools are popular and whether other organizations have found them useful. For example, a 1998 study of the use of CASE tools by system developers found that 76 respondents reported using 13 different CASE tools, with the majority of the responses involving Texas Instrument's IEF (49 respondents) and Sterling Software's ADW (15 respondents). These tools, along with Oracle and Intersolv's Excelerator were consistently ranked highest in an evaluation of features.

### C. Critical Success Factors

Post, Kagan, and Keim (1999) sought to evaluate CASE tools as well as individual features. Systems analysts and developers who used a CASE tool on a day-to-day basis were asked to evaluate the tool on a number of criteria. Results indicated that the prototyping feature did not have a significant impact on the overall evaluation of the CASE tool, indicating that the users were not using the tools for prototyping. The respondents seemed to perceive that no one CASE tool met all of their needs. Some were rated strong in graphics and data dictionary features whereas others were rated strong in analysis. Those tools that were rated strongest in analysis were also rated strongest overall indicating the importance of these features.

## IV. OBJECT-ORIENTED ANALYSIS AND DESIGN

The traditional system analysis and design methodologies do not necessarily support the use of object-oriented (OO) tools. The object orientation involves saving data and the procedures that impact that data as self-contained units, called objects. This approach offers many advantages, including cost reductions and increases in the speed of system development through the reusability of objects. Data and functions are encapsulated, thus simplifying maintenance and reducing errors. Once objects are created they can then be reused in additional applications. This assumes that individual programmers take advantage of previously designed and developed objects and that the designs are compatible. In this respect, it is even more important to have a tool for defining standards, documentation, and facilitating the coordinated design activities in an object-oriented environment.

### A. Unified Modeling Language (UML)

The implementation of an information system in an object-oriented environment involves storing data differently, thus the traditional modeling techniques, such as entity-relationship diagrams, are no longer appropriate. There were a number of independent OO modeling techniques for the analysis and design of OO systems. Grady Booch's "Booch" method, Ivar Jacobson's Object-Oriented Software Engineering method and James Rumbaugh's Object Modeling Technique were unified, resulting in the popular and dominant Unified Modeling Language. UML is an integrated modeling technique that can be used to model a system from analysis, design, and into implementation. Unlike the modeling methods used in the traditional system development life cycle, the modeling techniques are integrated and interrelated resulting in fewer errors and increased speed.

## B. OO CASE Tools

Again, given the number of CASE tools and the variety of features and functions that they offer, there is not a reliable measure of the OO CASE tool market share. As with traditional CASE tools, there are a number of research articles, case studies, and product reviews. For example, Post and Kagan (2001) conducted a survey of American developers who use CASE tools in their day-to-day jobs. Respondents indicated the use of 24 CASE tools that support an object orientation. Rational Rose from Rational Software (34), Computer Associate's Paradigm Plus (18), Aonix's StP/OMT (13), and Texas Instrument's Composer (10) were most frequently identified.

## V. FUTURE DIRECTIONS FOR CASE

Whether they are called CASE tools or not, development tools offer a number of substantive benefits for the development of information systems. Despite the bad press, there is evidence that CASE tools do help develop quality information systems in a reduced amount of time. The use of a development tool throughout the stages of system development can also result in better and more complete documentation, thus facilitating the maintenance of the information system. The speed with which electronic commerce applications need to be developed has contributed to different development methodologies. The increased use of RAD and OO analysis and design will surely continue to support the use of CASE tools.

## SEE ALSO THE FOLLOWING ARTICLES

Data Flow Diagrams • Data Modeling: Object-Oriented Data Model • Program Design, Coding, and Testing • Project Management Techniques • Pseudocode • Quality Information Systems • System Development Life Cycle • System Implementation

## BIBLIOGRAPHY

Aegan, I., Siltanen, A., Sorensen, C., and Tahvanainen, V. (1992). A tale of two countries: CASE experiences and expectations. In *The impact of computer supported technologies on information systems development.* (K. E. Kendall, K. Lyytinen, and J. DeGross, eds), 61–91. Amsterdam: IIP Transactions, North-Holland.

Finlay, P. N., and Mitchell, A. C. (1994). Perceptions of the benefits from the introduction of CASE: An empirical study. *MIS Quarterly,* 18(4): 353–370.

Hanna, M. (1992). Using documentation as a life-cycle tool, *Software Magazine,* 12(12): 41–51.

Iivari, J. (1996). Why are CASE tools not being used? *Communications of the ACM,* 39(10): 94–106.

Kremerer, C. F. (1992). How the learning curve affects CASE tool adoption. *IEEE Software,* 9(3): 23–28.

Norman, R. J., and Nunamaker, J. F. (1989). CASE productivity perceptions of software engineering professionals. *Communications of the ACM,* 32(9): 1102–1108.

Orlikowski, W. J. (1993). CASE tools as organizational change: Investigating incremental and radical changes in systems development. *MIS Quarterly,* 17(3): 309–340.

Post, G. V., and Kagan, A. (2001). User requirements for OO CASE tools. *Information and Software Technology,* 43(8): 509–517.

Post, G. V., Kagan, A., and Keim, R. T. (1998). A comparative evaluation of CASE tools. *The Journal of Systems and Software,* 44(2): 87–96.

Post, G. V., Kagan, A., and Keim, R. T. (1999). A structural equation evaluation of CASE tool attributes. *Journal of Management Information Systems,* 15(4): 215–234.

Senn, J. A., and Wynekoop, J. L. (1995). The other side of CASE implementation: Best practices for success. *Information Systems Management,* 12(4): 7–14.

Sumner, M., and Ryan, T. (1994). The impact of CASE: Can it achieve critical success factors? *Journal of Systems Management,* 45(6): 16–20.

# Computer Hardware

**Ata Nahouraii**
*Indiana University of Pennsylvania*

## GLOSSARY

**analog** Associated with the continuous wave or pulse type signal (such as the human voice). If transmitted to a computer or terminal it must be converted to a digital signal [0,1] before it can be processed.

**array** Any digital image stored as a two-dimensional data and addressable by *x,y* (or row, column) coordinates.

**binary** Binary, or base 2, is a numbering system with only two digits, 0 and 1. Binary is convenient for use with bits which have only two states, on and off.

**bit** The smallest unit of memory in the computer.

**bit depth** The number or bits used to represent each pixel (picture element) in an image determining its color or tonal range.

**byte** A unit of measure equal to 8 binary bits. This standard unit is used to measure file size, computer memory size, and disk or mass storage capacity.

**CCD (charge-coupled device)** A diode that is light-sensitive when charged with electrical voltage and is able to convert light into an electrical charge.

**CMS (color management system)** Measures color uniformity across input and output devices so that final printed results match originals. The characteristics or profiles of devices are normally established by reference to standard 1T8 color targets.

**CMYK (*c*yan, *m*agenta, *y*ellow, blac*k*)** The subtractive primaries, or process colors, used in color printing. Black (K) is usually added to enhance color and to print a true black.

**density** The ability of a material to absorb light. It is a measure of light transmission of a transparent or translucent object or the ratio of the number of bits to the total number of bits in an object. The greater the density area, the more compact the absorbency of the surface.

**digital** Method of data storage and/or transmission wherein each element of information is given a unique combination of numerical values [0,1] or bits. Digital data consist of discrete steps or levels and have a finite state. This is in contrast to analog data which are a continuous form.

**dot** Smallest visible point that can be transmitted, transcribed, or displayed.

**DPI (dots-per-inch)** A method of denoting the resolution of a scanned image, or a digitized image in a file.

**drum scanner** An optical input device that is mounted on a revolving cylinder for digitizing.

**gigabyte (GB)** A unit of measure defined as one billion bytes, or 1,024,000,000 characters.

**LAN (local area network)** A group of connected computers sharing access to printers and other peripheral devices. It may use a wire or optical fiber link for data transfer.

**pixel** Another term for picture element; it is a two-dimensional array of dots that define the form and color of an image. Measurement is indicated as PPI (pixels per inch). The pixel most often refers to screen dots rather than image dots.

**PMT (photomultiplier tube)** A light sensitive tube associated with drum scanners.

**SCSI (small computer system interface)** An internal communications standard for computers, through which hard drives, scanners, and other peripherals transfer data.

**transparency scanner** An optical input system for digitizing images from positive or negative transparency film.

Willing or not, we are being ushered into a new period of technological advances earmarked by the computer. Today, almost everyone is affected by some form of automated information system. The ability of a computer system to perform certain functions effectively and efficiently depends on its hardware and its associated software. The hardware are the physical components that are needed to support the input, processing, and output activities of a given system; software, on the other hand, consists of programs or instructions that make the computer work.

This article describes various types of input/output (I/O) devices that are needed by a user for present and future expected functions. It will address I/O devices that can support personal computers (PC), workstations, midrange computers, mainframe systems, and highly specialized systems known as supercomputers.

## I. HISTORICAL DEVELOPMENT OF I/O DEVICES

Prior to World War II, data were processed either manually or mechanically; however, the war created an urgent need for new data-processing methods. The areas of aircraft design, development of military weapons, and procurement of materials and supplies required more efficient ways to achieve the intended objectives by providing timely reports with vital statistics. This led to the development of automatic calculating machines, wired controlled circuit boards, banks of switches, keys, and dials. With this impetus, a new initiative in the design and development of I/O devices began to surface as shown below.

- 1800s—Punch cards
- 1940s—Magnetic drums
- 1950s—Magnetic tape drives, first "hard" drive, disk packs
- 1960s—Direct access storage
- 1970s—Mass storage subsystems, winchester
- 1980s—Small sized hard/floppy drives
- 1990s—CD-ROM, CD-RW, DVD, Zip, Jaz
- 2000—Online storage, SAN, NAS

It should be noted that the modern I/O devices are very complex, fast, and robust and contain fail-safe redundancy features.

## II. INPUT DEVICES

### A. The Keyboard

One of the most difficult problems facing the development of input devices was that of a keyboard for the delivery of massive amounts of data that would be collected, analyzed, and used to control computer systems.

A computer keyboard is an array of switches, each of which sends a unique signal when pressed. The switches are spring-loaded "push to make" types, so when pressed down they complete the circuit and then break it again when released. Most computer keyboards have been enhanced with plenty of tactile feedback and the ability to issue commands in several combinations. This enhanced design consists of a full array of function keys, a separate set of movement keys for the cursor, as well as specific keys for use with internet connectivity or within applications. An important factor for keys is the force displacement curve which is designed to register how much force is needed to depress a key. In general, keys are designed for 80–100 g of force, whereas game consoles are designed to accept 120 g or higher (see Fig. 1).

A newly developed keyboard is the split keyboard. This type of a keyboard is ergonomically designed to avoid wrist and hand injuries and help avoid carpal tunnel syndrome. The keyboard can enter schematic via touchpad and text by its keys.

### B. The Mouse

In the early 1980s, the first PCs were equipped with the traditional user input device: a keyboard. By the end of the decade, however, a mouse device became essential for running the graphical user interface (GUI) operating systems.

There are two common types of mouse: electromechanical and optical. The electro-mechanical has

Figure 1

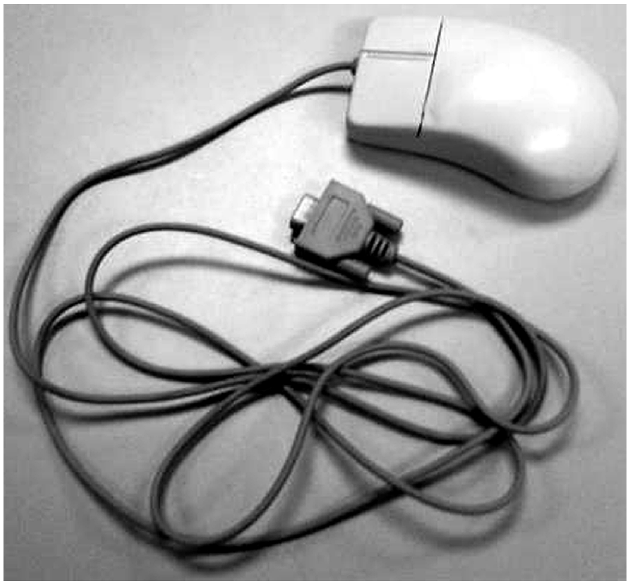
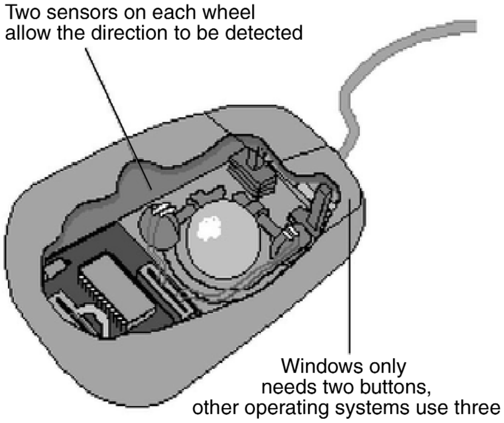


Figure 2

recently replaced its hard rubber ball in the base with a steel ball for weight and a rubber coating for grip, which turns movement into electrical signals. The optical mouse projects a beam of light downward and often uses a special metallic pad, depending on the model selected.

Some tasks such as drawing lend themselves to efficient use of a mouse. Most recently a new mouse, called opto-electronic, has been introduced. Its ball is steel for weight and as it rotates, it drives two rollers, one each for the *x* and *y* axes. It should be noted that tasks such as word processing, which may involve frequent use of cut-and-paste commands for editing, would be awkward and time-consuming without a mouse (see Figs. 2–4).

Figure 3



Two sensors on each wheel allow the direction to be detected
Windows only needs two buttons, other operating systems use three

Figure 4

## C. The Trackball

This pointing device is a stationary, upside-down mouse and generally is used with laptops. The advantage of the trackball over the mouse is that it minimizes hand–eye coordination, as well as reducing extra desk space needed to navigate the mouse. Most notebooks or laptops have built-in trackballs. Some notebooks, however, are equipped with a skin membrane called a touchpad. The touchpad generally is made of a thin plastic membrane and is coated with a conducting material and spread over a printed circuit board. Navigation is performed by applying pressure on the surface of the membrane similar to the movement of a trackball.

## D. Optical Character Recognition Devices

An optical character recognition device, often abbreviated as OCR, is able to recognize text that is printed in a specific type font. Early OCR equipment could only read one type face (`like this one`) in dot matrix form. Scanners are a form of OCR family that can read almost any type font and their accuracy depends in large part on the text or recognition software used. The device converts light—an analog continuous wave form—into digital binary bits of zero and one [0,1] which is a discrete wave form. To accomplish this, scanners use electronic components such as charge-coupled devices (CCD), a diode that is light sensitive when electrically charged, or photomultiplier tubes (PMT), a light sensitive tube that detects light at any intensity by amplifying it. PMTs are usually associated with drum scanners. Some examples of scanners are as follows.

## E. Flatbed Scanners

These are the most common desktop scanners that resemble copy machines because the item being scanned rests on a glass plate while the scanning head moves underneath it. The flatbed scanners are able to capture images from source material as well as three-dimensional objects. A transparency adapter is required in order to scan slides, x-rays, and other transparent originals. Because flatbed scanners allow one to feed in sheets continuously, an automatic document feeder is needed to handle large numbers of documents (see Fig. 5).

## F. Sheetfed Scanners

The sheetfed scanners act more like fax machines rather than copiers; they move the page being scanned past the scanning head, rather than the other way around. These scanners can only scan a single sheet at a time unless equipped with built-in document feeders for multiple-page scanning. Sheetfed scanners are less exact than their flatbed counterparts. The difficulty of moving a sheet of paper without introducing distortion makes them less precise than flatbed scanners.

## G. Hand-Held Scanners

Over the last few years, hand-held scanners have grown in popularity. These units are able to capture images from labels on cans or containers that can't be conveniently fed through flatbed or sheetfed scanners (see Fig. 6).

## H. Drum Scanners

Before the advent of desktop scanning, most images were loaded into computers through drum scanners. These units were used primarily in color prepress companies. The originals were mounted on a glass cylinder, which would then be rotated at high speeds around a sensor located in the center for image processing.

Today, drum scanners have been enhanced by using PMTs. These sensors are more advanced than the CCDs and contact image sensors (CISs) used in other kinds of scanners. With these advanced sensors and the process of repeated rotating of the original past the PMTs at high speed, the drum scanners are important professional tools.

## I. Scanner Specifications

A majority of scanners employ the "bit map" method of storing graphic images. This means the information is stored in a digital memory as a rectangular array of bits. In order to compare the performance of multiple scanners, the criteria listed below are typically used:

- Resolution
- Bit-depth
- Dynamic range
- Scanning method
- Scanning area
- Speed

In addition to the above criteria, there are other issues to consider such as the aesthetics of case design

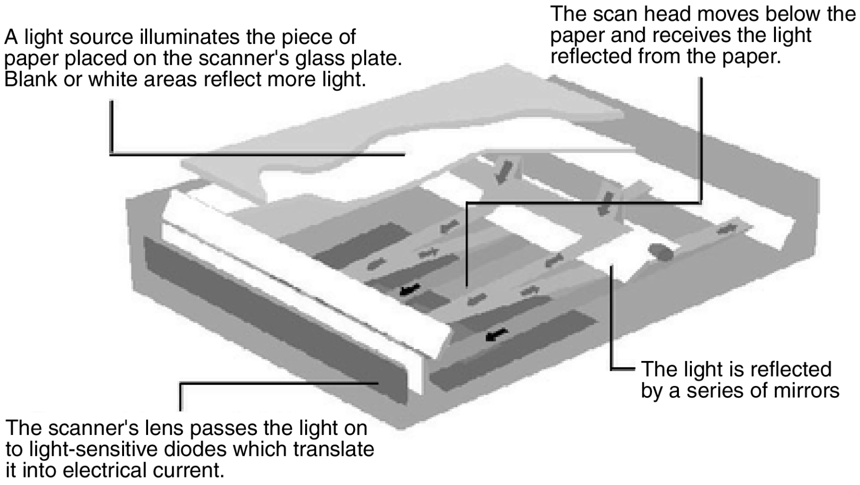

A light source illuminates the piece of paper placed on the scanner's glass plate. Blank or white areas reflect more light.
The scan head moves below the paper and receives the light reflected from the paper.
The light is reflected by a series of mirrors
The scanner's lens passes the light on to light-sensitive diodes which translate it into electrical current.

Figure 5

Figure 6

and the quality of construction, footprint (how much desk space is taken up), connection type (SCSI, parallel, USB, FireWire), bundled software, and price.

## J. Bar Codes

Bar code is an automatic identification technology that allows rapid data collection and does it with extreme accuracy (see Fig. 7). It provides a simple and easy method to encode information which is easily read by inexpensive electronic devices. Bar code is a defined pattern of alternating parallel bars and spaces, representing numbers and other characters that are machine readable. Predetermined width patterns are used to code data into a printable symbol. Bar codes can be thought of as a printed version of the Morse code in that the narrow bars represent dots and the wide bars represent dashes. The bar code reader decodes a bar code by scanning a light source across the bar code and measuring the intensity of light reflected back to the device. The pattern of reflected light produces an electronic signal that exactly matches the printed bar code pattern and is easily decoded into the original data by simple electronic circuits. There are a variety of different types of bar code encoding schemes called symbologies, each developed to fulfill a specific need in a specific industry. Bar code technology is designed to function best with a red light as a scanning spot and is designed to be bidirectional to increase performance. The bidirectionality remains the same if bar codes are read from top to bottom/bottom to top, or left to right/right to left. Currently over 40 bar code symbologies have been developed with their unique scanning device.

Since bar code symbologies are like languages that encode information differently, a scanner programmed to read a particular code cannot read another. Some of the commonly encoding schemes or "symbologies" are code 39 (Normal and Full ASCII), Universal Product Code (UPC-A, UPC-E), and European Article Numbering system (EAN-8, EAN-13). The EAN symbologies adhere to the same size requirement.

In summary, the use of bar coding systems assists in the collection of data for production control, automatic sorting systems, or monitoring work-in-progress. As such, bar coding provides accurate, immediate information; thereby, it facilitates the decision making process (see Figs. 8 and 9).

## K. Automatic Teller Machine

The backbone to any bank system and the one we most often interact with is the ATM, or automatic teller machine. The system acts as a data terminal and supports two input and four output devices.

### 1. The Input Devices

- Card reader. The card reader captures the account information stored on the magnetic strip on the back of an ATM/debit or credit card. The host processor uses this information to route the transaction to the cardholder's bank.
- Keypad. The keypad lets the cardholder tell the bank what kind of transaction is required (cash withdrawal, balance inquiry, etc.) and for what amount. Also, the bank requires the cardholder's personal identification number (PIN) for verification. Federal law requires that the PIN block be sent to the host processor in encrypted form (see Fig. 10).

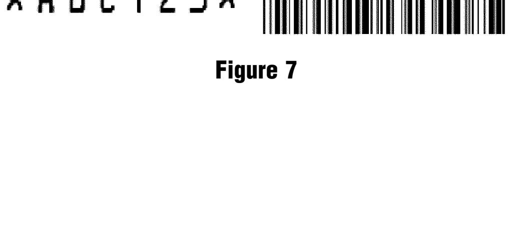


Figure 7

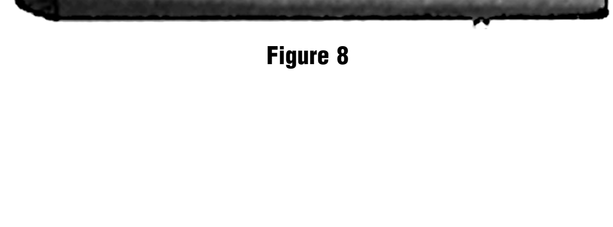


Figure 8

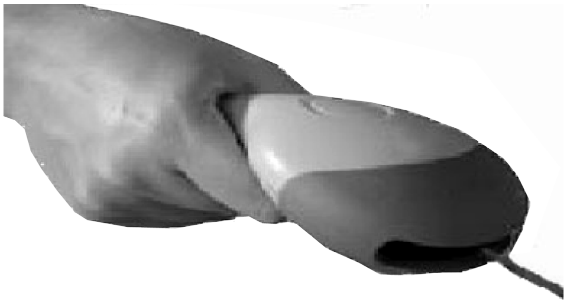


Figure 9

## 2. The Output Devices

- Speaker. The speaker provides the cardholder with tactile feedback when a key is pressed.
- Display screen. The display screen prompts the cardholder through each step of the transaction process. Leased-line machines commonly use a monochrome or color cathode ray tube (CRT) display. Dial-up machines commonly use a monochrome or color liquid crystal display (LCD).
- Receipt printer. The receipt printer provides the cardholder with a paper receipt of the transaction.
- Cash dispenser. The heart of an ATM is the safe and cash-dispensing mechanism. The entire bottom portion of most small ATMs is a safe that contains the cash.

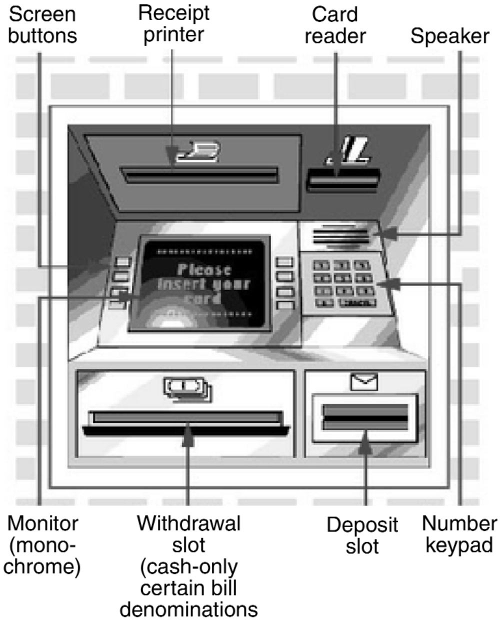


Figure 10

The cash-dispensing mechanism is equipped with photoelectric sensors that count each bill and measure the thickness of each bill as it exits the dispenser. If two bills are stuck together, excessively worn, torn or folded, the dispensing mechanism diverts it to a reject bin. The number of reject bills is also recorded so that the machine owner can be aware of the quality of bills that are being loaded into the machine. A high reject rate would indicate a problem with the bills or with the dispenser mechanism.

The bill count and all of the information pertaining to a particular transaction are recorded in a journal. The journal information is printed out periodically and a hard copy is maintained by the machine owner for 2 years for transaction dispute resolutions.

Like most input devices, the ATM is connected to a front-end or host processor for its feedback. The host processor is analogous to an Internet service provider (ISP) in that it is the gateway between the bank's system and the ATM machine as shown in Fig. 11.

Most host processors can support either leased-line or dial-up machines. Leased-line machines connect directly to the host processor through a four-wire, point-to-point, dedicated telephone line. Dial-up ATMs connect to the host processor through a normal phone line using a modem and a toll-free number, or through an Internet service provider using a local access number via a modem.

Leased-line ATMs are preferred for very high-volume locations because of their throughput capability, and dial-up ATMs are preferred for retail merchant locations where cost is a greater factor than throughput. The initial cost for a dial-up machine is less than half that for a leased-line machine. The monthly operating costs for dial-up are only a fraction of the costs for leased line.

The host processor may be owned by a bank or financial institution, or it may be owned by an independent service provider. Bank-owned processors normally support only bank-owned machines, where as the independent processors support merchant-owned machines.

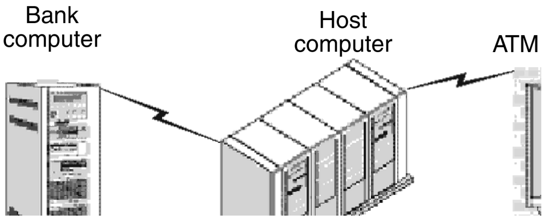


Figure 11

## L. Readers

Serial, or serial port readers, utilize the assigned channels of a computer to pass data for processing from a source such as card-reading devices, magnetic strip readers, or optical character recognition sensors. Serial readers pass data, 1 bit at a time. Parallel readers pass 8 bits at a time. Today, people have come to know the term "smart card" related to parallel and serial readers. Smart cards, such as credit cards, may be swiped into the system for identification. What makes a smart card different from any old piece of plastic and from magnetic-strip cards is an embedded microchip.

These smart cards are more prevalent for use in security areas, or employment verification. They also can be connected to cameras and photo scanners. Light wands have both a light source, and a light detector in a pen-like container which must stay in contact with the source of the data.

Card systems developed in the 1950s were data driven by punch cards used to create and update data. The early card systems were either brush type or photoelectric. The brush type sensed electrically the presence or absence of a hole in each position of a card to form electrical pulses. These pulses were then detected by card-reader circuitry and stored accordingly as data. In contrast, the photoelectric readers sensed each hole by means of light passing through the hole; this process activated a photoelectric cell to cause an electrical pulse to be formed.

Today, the Universal Serial Bus (USB) readers have now replaced many of the parallel and serial readers of the past. The USB peripheral bus standard was developed by Compaq, IBM, DEC, Intel, Microsoft, NEC, and Northern Telecom and the technology is available without charge for all computer and device vendors (see Fig. 12).

Figure 12

## M. Radio Frequency Identification Devices (RFID)

Other than bar codes, products can be identified by a system known as radio frequency devices. Products are given a chip or tag that is read by a radio frequency identification device (RFID). The RFID can detect the tag and send that information to be processed. Some examples of this technology are used for tracking equipment at warehouses and for automatically charging users of toll roads, without them having to stop to pay at a booth.

The most popular formats are used in homes in America and abroad. The radio and television industry uses the same technology that all radio frequency devices employ to deliver programs to a global audience. RF transmissions are possible in nearly every frequency of the electromagnetic spectrum. Most wireless net working systems transmit in gigahertz band. This is a relatively clear frequency range that offers large amounts of frequency bandwidth in which to transmit data. While the technology is quick, reliable, and secure, it does carry a high cost.

## III. OUTPUT DEVICES

Output is associated with processed data. It may be in a readable form that can be understood, or retained in a machine-readable form to serve as input to another machine. Today, the majority of the output received is generally through display screens and printers. Each type of printer uses a different technology to operate as can be observed from the following descriptions.

## A. IMPACT Printers

IMPACT printers refer to a class of printers that work by banging a head or needle against an ink ribbon to make a mark on the paper. This includes dot-matrix printers, daisy-wheel printers, and line printers. In contrast, laser and ink-jet printers are nonimpact printers. The distinction is important because impact printers tend to be considerably nosier than nonimpact printers but are useful for multipart forms such as invoices (see Fig. 13).

Figure 13

## B. Daisy Wheel Printers

This printer produces letter-quality type. A daisy wheel printer works on the same principle as a ball-head typewriter. The daisy wheel is a disk made of plastic or metal on which characters stand out in relief along the outer edge. To print a character, the printer rotates the disk until the desired letter is facing the paper. Then a hammer strikes the disk, forcing the character to hit an ink ribbon, leaving an impression of the character on the paper. You can change the daisy wheel to print different fonts. Daisy wheel printers cannot print graphics, and in general they are noisy and slow, printing from 10 to 75 characters per second. As the price of laser and inkjet printers has declined and the quality of dot-matrix printers has improved, daisy wheel printers have become obsolete (see Fig. 14).

Figure 14

## C. Dot Matrix

Dot matrix was the dominant print technology in the home computing market in the days before the inkjet. Dot matrix printers produce characters and illustrations by striking pins against an ink ribbon to print closely spaced dots in the appropriate shape. They are relatively expensive and do not produce high-quality output. However, they can print to continuous stationery multi-page forms, something laser and inkjet printers cannot do. Print speeds, specified in characters per second (cps), vary from about 50 to over 500 cps. Most dot matrix printers offer different speeds depending on the quality of print desired. Print quality is determined by the number of pins (the mechanisms that print the dots). Typically, this varies from 9 to 24. The best dot matrix printers (24 pins) are capable of near letter-quality type.

## D. Thermal Wax

Thermal wax is another specialist technology well suited for printing on transparencies (see Fig. 15). It uses CMY or CMYK (cyan, magenta, yellow, black) rolls containing page-sized panels of plastic film coated with wax-based colorants. It works by melting ink dots—generally binary, although some higher-end models are capable of producing multi-level dots—on to special thermal paper. Resolution and print speeds are low, typically 300 dpi and around one page per minute (1 ppm). This technology is suitable for applications by specialists.

## E. Solid Ink Printers

Marketed almost exclusively by Tektronix, solid ink printers are page printers that use solid wax ink sticks in a "phase-change" process (see Fig. 16). They work by

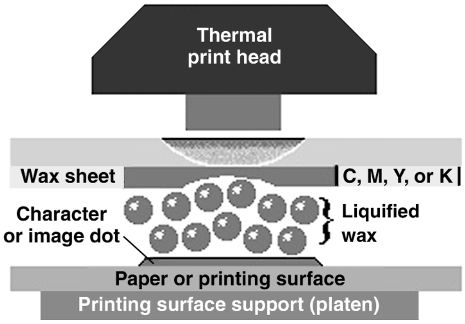


Figure 15

Figure 16

liquefying wax ink sticks into reservoirs, and then squirting the ink onto a transfer drum, from where it is cold-fused onto the paper in a single pass. Once warmed up, thermal wax devices should not be moved to avoid damage. They are an excellent form of data sharing over a network. To this end, they come with Ethernet, parallel, and SCSI ports allowing for comprehensive connectivity.

Solid ink printers are generally cheaper to purchase than a similarly specified color laser and economical to run, owing to a low compartment count. Output quality is good, with multilevel dots being supported by high-end models, but generally not as good as the best color lasers for text and graphics, or the best inkjets for photographs. Resolutions start at 300 dpi, to a maximum of 850 by 450 dpi. Color print speed is typically 4 ppm in standard mode, rising to 6 ppm in a reduced resolution mode.

## F. Inkjet Printers

Although inkjets were available in the 1980s, it was not until the 1990s that prices dropped enough to make them more affordable to the masses (see Fig. 17). Canon claims to have invented what it terms “bubble jet” technology in 1977, when a researcher accidentally touched an ink-filled syringe with a hot soldering iron. The heat forced a drop of ink out of the needle and so began the development of a new printing method.

Inkjet printers have made rapid technological advances in recent years. The three-color printer has been around for several years and has succeeded in making color inkjet printing an affordable option; but as the superior four-color model became cheaper to produce, the swappable cartridge model was gradually phased out. Inkjet printers of high quality use six inks—photo magenta, magenta, photo cyan, cyan, yellow, and black. They come with six separate tank units. The latter design was intended for controlling cost in the event one color is depleted faster than the others due to high usage. This way, the unit used most frequently can be replaced for that color. They also have high picoliter count, a measurement used to determine how finely ink is sprayed onto the paper.

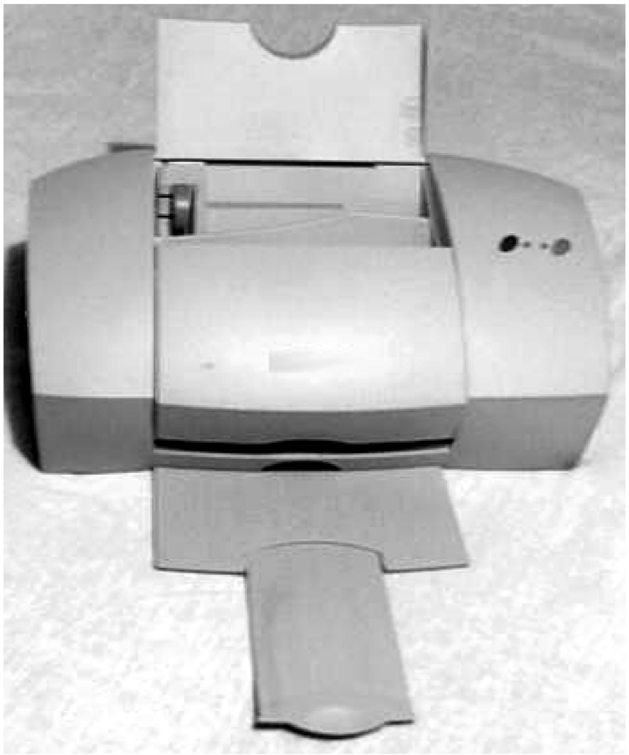


Figure 17

## G. Laser Printers

Hewlett–Packard introduced the laser printer in 1984, based on technology developed by Cannon. It works in a similar way to a photocopier, the difference being the light source. With a photocopier, a page is scanned with a bright light, while with a laser printer the light source is, not surprisingly, a laser. After that, the process is much the same, with the light creating an electrostatic image of the page onto a charged photoreceptor, which in turn attracts toner in the shape of an electrostatic charge. Laser printers quickly became

Figure 18

popular due to the high quality of their print and their relatively low running costs. As the market for lasers developed, competition between manufacturers became increasingly fierce, especially in the production of low-end models. Prices have gone down and down as manufacturers have found new ways of cutting costs.

Output quality has improved over the years, with 600-dpi resolution becoming more standard. Their size has become smaller, making them more suited for home use (see Fig. 18).

## IV. AUXILLARY DEVICES

Some devices act as input and output devices. Storage devices are considered I/O devices because we can send information to them and similarly retrieve that information from them. A POS (point-of-sale) system is also considered as in input output system, as it takes in information like credit card information and produces a screen output and paper output. Some examples of auxiliary devices are as follows.

### A. Magnetic Tape Storage Devices: 7-Track and 9-Track

Seven- and nine-track tapes were magnetic tapes used for storage in the 1970s, 1980s, and well into the 1990s. Almost every kind of minicomputer and mainframe used them for backup, data archiving, and data interchange. Physically at least, they were well standardized for interchange. They use various logical formats such as ASCII, EBCIDIC, OS, ANSI. The tapes could be made "read only" so that the data remain safe.

The 9-track tapes could also be recorded in different densities: 800, 1800, and 6250 bpi (bytes per inch); the higher the density, the newer the technology. The 1600- and 6250-bpi tape drives could recognize lower densities (e.g., a 1600-bpi drive could read an 800-bpi tape). The "9-track" refers to recording 8 data bits plus one parity bit across the tape (edge to edge). Thus 1600 bpi is the same as 1600 cpi (characters per inch). At 1600 bpi, a 2400-foot 9-track tape could hold about 50 MB (depending on blocking, etc).

In earlier times, there were 4-track and 7-track drives at lower densities such as 200 and 556 bpi. IBM was the main vendor.

### B. Magnetic Strip Reader

Magnetic strip readers are convenient for retailers who want to accept credit or debit cards. They are usually attached as a keyboard wedge or directly to a serial port and are most often configured to read tracks one and two or one, two, and three. They can also read driver's licenses, ID cards, and security badges (see Fig. 19).

### C. Paper Tapes

In 1857, Sir Charles Wheatstone introduced the first application of paper tapes as a medium for the preparation, storage, and transmission of data. He used the paper tape to punch in the Morse code with dots or holes. IBM improved this design with standard one-inch wide tapes to represent data in five-, seven-, or eight-channels, and numbered from 0 to 7 with 0.1 in.

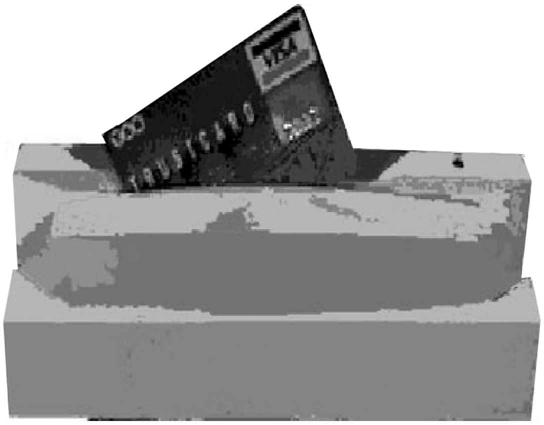


Figure 19

between the punched holes. Each row represents a character. This invention produced the paper used in ticker tape (stock market reporting) machines. The paper-tape device can be configured to be a paper-tape reader by having it combined into one device.

## D. Cartridge Tapes

Magnetic cartridge tapes were a significant improvement over the reel tapes used predominately in the 1970s and 1980s. Their advantage comes from their small size (usually 4 or 5 in.). They are enclosed and therefore protected from dust and dirt. The cartridge consists of a small reel of chromium dioxide tape that is enclosed in a compact plastic housing. The chromium dioxide coating of a cartridge tape permits high density and improved data reliability. A standard 3480-tape cartridge, with only 537 feet of tape, has a data capacity greater than that of a standard 2400-foot tape reel when it is used with a block size larger than 4 kilobytes. The approximate capacity of the tape cartridge, written in 24-KB blocks, is 200 megabytes, while the capacity of a reel tape at the same block size is about 165 MB. Another advantage of magnetic cartridge tapes is the ability to store large amounts of data at a very affordable price. However, one disadvantage is the access time. Data must be restored to the hard disk before the data can be accessed.

## E. Magnetic Disk Drives

In a hard disk, the magnetic recording material is layered onto a high-precision aluminum or glass disk (see Fig. 20). The hard disk platter is then polished to mirror smoothness. There are arms that read the information off these platters. The platters spin anywhere from 170 mph to almost double the speed. Data are stored on the surface of a platter in sectors and tracks. Tracks are bent looking circles, and sectors are pie-shaped wedges (see Fig. 21).

## F. Magnetic Drum Units

Magnetic drum units are a predecessor to modern hard drives. They consist of a metal cylinder coated with magnetic iron-oxide material. A drum can have up to 200 bands around it, which are logical tracks in which data are stored. One problem is that the drum is permanently mounted in the device.

Magnetic drums are able to retrieve data at a quicker rate than tape or disk devices but are not able to store as much data as either of them.

## G. Fault Tolerant Systems

These systems create a map that can rebuild data if a disk crash occurs. An example of this type of system is the redundant array of independent/inexpensive disks (RAID). The most popular types of RAID are RAID 1 and RAID 5. RAID 1 is mirroring, which makes a direct copy of a drive onto an adjacent drive. RAID 5 partly duplicates the data with a minimum of 3 drives with an option for online spares. RAID improves performance and reliability.

## H. Mass Storage

Mass storage devices refer to large-capacity auxiliary storage in an IT-system environment. The components

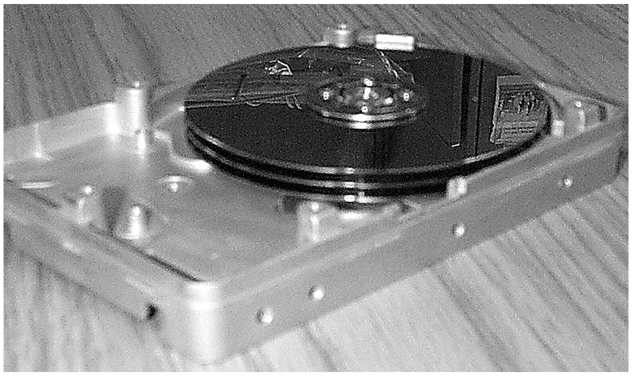


Figure 20

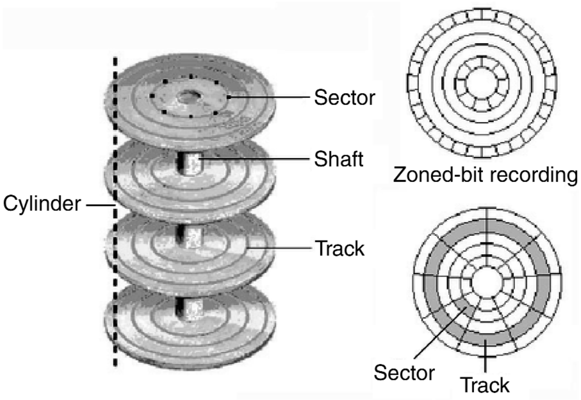


Figure 21

may be high-end storage devices like small computer systems interface (SCSI, pronounced "skuzzy") hard disks, tape drives, tape autoloaders, compact disk (CD), digital video disk (DVD), and storage area networks (SANs). The two most popular mass storage devices are SAN (storage area networks) and drive array subsystems. SAN usually use fiber optic channels to connect to servers in order to increase speed and storage capacity. SAN storage usually ranges from several hundred gigabytes to the Terabytes (see Fig. 22).

The drive array subsystem is typically connected to servers using SCSI technology. Each intelligent controller of SCSI manages the flow of information, and the devices it supports may be made up of several disks or different implementations of RAID that are directly attached to their particular dedicated host servers. The drive arrays, when equipped with RAID 5 disks, strip data across several drives. One drive has the stripping bit and often there are online spare drives in case of drive failure.

The server—a combination of hardware and software—functions in managing shared resources and provides services to computer programs in the same computer, or to computers on another network. Many servers are configured with a RAID controller. When used for the World Wide Web, the server may be called a virtual server. In this mode, the server is at someone else's premises and is shared by multiple Web-site owners with each owner having complete control of the server.

A network is a group of computers connected to each other by means of adapters and cables. It has the ability to share devices and data among the group of computers or other physical devices such as printers or fixed disks, or a directory on a fixed disk that contains information.

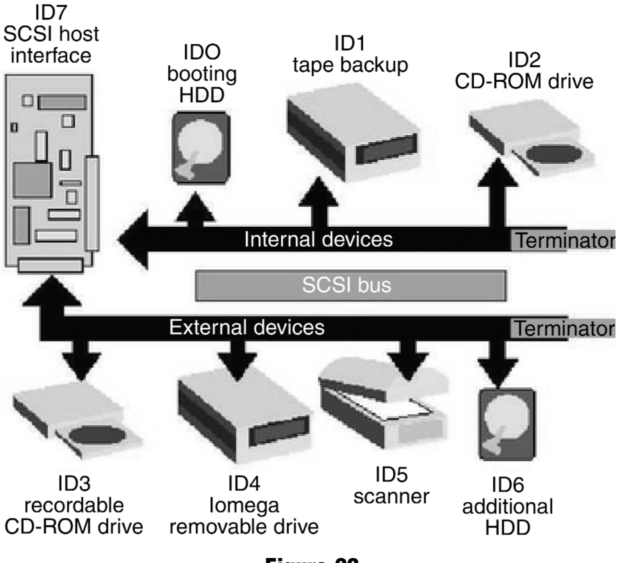


**Figure 22**

## I. Direct Access Devices

Direct access devices consist of magnetic disk drives, optical drives, and many other types of devices. They all access media directly or without the need to pass over other data. Tape drives read data sequentially until they read the information asked for. Because of this, direct access devices are usually faster than sequential devices.

## J. WORM Compact Disks and CD-RW Devices

WORM stands for "write once read many." This is the case of all CD media except CD-RW. CDs are types of optical disks that have burned pits in the surface that represent the binary digit 1. Light is reflected off of the pits by the use of a laser. A CD typically holds 740 MB of data, but recently they have been able to hold up to 900 MB of data by using the CD's outer tracks (see Fig. 23).

## K. Disk Digital Video Units

DVD stands for digital video disk. The media is similar in size to a (CD). The media has a capacity of 4.7 GB of storage for data per side and can be double layered to create a total size of up to 17 GB. This device is usually used to store video. It stores up to 1355 minutes of video in 5.1 Dolby digital sound and has the highest quality picture associated with it (see Fig. 24).

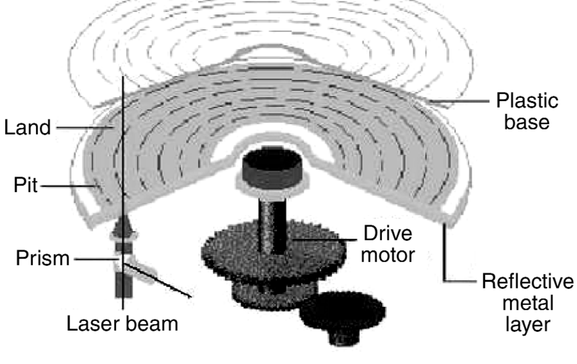


**Figure 23**

## L. Flash Memory

Flash memory refers to memory chips that can be rewritten and hold their content without power. It is widely used for digital camera film and as storage for many consumer and industrial applications. Flash memory chips earlier replaced read only memory basic input/output system (ROM BIOS) chips in a PC so that the BIOS could be updated in place instead of being replaced. Flash memory chips generally have life spans from 100 to 300K write cycles.

### 1. Other Optical Devices

On the market today are many devices in use because of their durability, and because they do not need to be cleaned as often. There are no moving parts, thereby less chance of breakage. These devices include light pens, touch screens, bar code readers, and hand print readers. For example, a kiosk is comprised of a screen and may have several Input and Output [I/O] devices. Most kiosks are of LCD color touch-screen type. The user simply touches the area on the screen for the desired inquiry. Some kiosks are designed to include an internet connection and may be used as an e-mail kiosk.

## V. OTHER IMAGE PROCESSING SYSTEMS

Image processing systems use a sophisticated mathematical algorithm for image recognition. There are two different types of image processing systems that are currently used: fuzzy logic and neural networking. Fuzzy logic image processing is best described by scanning a picture into a computer. When scanned, the picture transmits black and white shades for its input. This gives just two colors for the object to use. However, a picture can be scanned by using grayscales, a system that uses both black and white, but also uses different mixes of the two to get different shades of gray to make the picture look as close to the original as possible. This system of using grayscales is like that of using fuzzy logic. It gives a computer the ability to make approximations about conditions of the image as opposed to only being able to report on the information at hand. This can be compared to a weather forecaster who reviews data on weather conditions and can predict storm patterns.

A second type of image processing system is a neural network system. In such a system, several processors and software are connected and are set to work together. This set up is able to simulate the operation of the human brain. For this reason, neural networks are good at pattern recognition and can solve complex problems, even for cases where all data are not present. Many of the uses for neural network systems lead us into the next topic, biometric devices. Many, but not all, biometric devices use a neural network to identify images.

## A. Biometric Devices

In computer security, biometrics refers to authentication techniques that rely on measurable physical char-

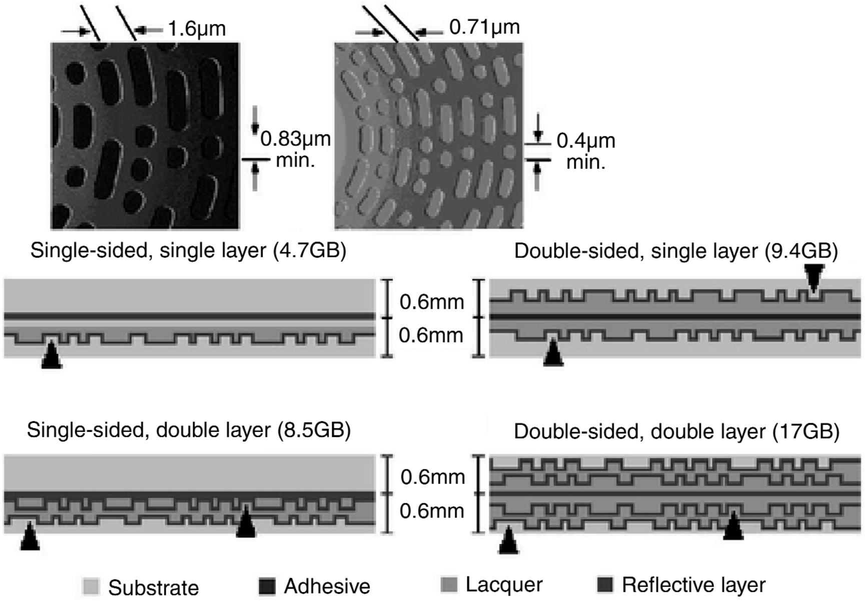

1.6μm
0.71μm
0.83μm min.
0.4μm min.
Single-sided, single layer (4.7GB)
Double-sided, single layer (9.4GB)
0.6mm
0.6mm
Single-sided, double layer (8.5GB)
Double-sided, double layer (17GB)
0.6mm
0.6mm
Substrate
Adhesive
Lacquer
Reflective layer

Figure 24

acteristics that can be automatically checked. In general they are special types of readers or sensors for data acquisition and processing. Examples include computer analysis of fingerprints, speech, or using the iris of the eye by digitally creating about 255 data points from the iris's unique pattern.

Proven application of speech-recognition systems currently in use include material handling applications, quality control, numerical machine-tooling parts programming, source data collections, military command/control, and applications for the physically challenged. The handprint or finger print application is used for door locks or a fingerprint identification mouse to secure drug closets in hospitals, health care facilities, airport security checks, law enforcement agencies, or office and home security. The eye scan is currently used as a passport for frequent travelers at airports such as Charlotte, North Carolina, to London's Heathrow Airport. The future for biometric application is vast. The reader may refer to the Disaster Recovery Planning section of the EIS for in-depth coverage of this topic.

## VI. APPLICATIONS

### A. Case 1. Automatic Teller Machines (ATM) Environment—Banks

In doing an ATM transaction, the user initiates the process by means of the card reader and keypad. The ATM forwards this information to the host processor, which routes the transaction request to the cardholder's bank or institution that issued the card. If the cardholder is requesting cash, the host processor causes an electronic funds transfer to take place from the customer's checking account to the host processor's account. Once the funds are transferred to the host processor's bank account, the processor sends an approval code to the ATM authorizing the machine to dispense the cash. If used as a debit card, the processor then executes an "automated clearing house" (ACH) notice for the cardholder's funds into the merchant's bank account, usually the next bank business day. In this way, the merchant is reimbursed for all funds dispensed by the ATM (see Fig. 25).

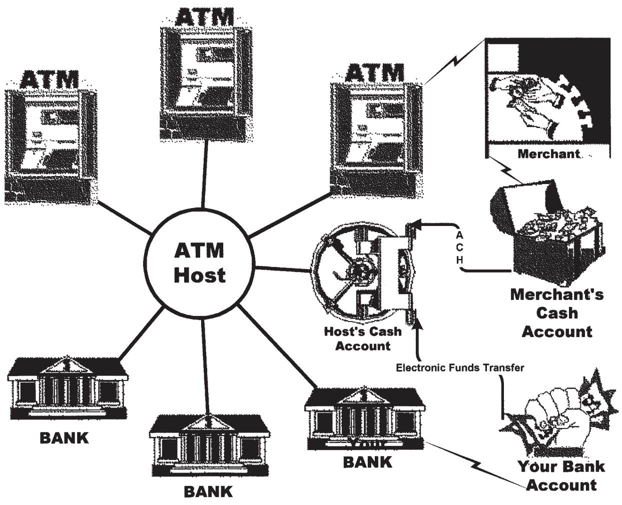


**Figure 25**

### B. Case 2. I/O Environment—Luxury Hotels

Technology plays an important role in the successful operation of any typical large-scale hotel or a convention center. They must rely heavily on how the needed data are collected, supported, and processed. Thus the computer-based information systems designed may include electronic devices, computers, software, networks, and operating systems. In that realm, its operation may be organized to have three major administrative functions: front office, marketing and sales, and back office.

The I/O devices that operate in tandem for the success of the operation are:

- The input devices, such as keyboards, mice, point-of-sale, touch screens, floppy and disk drives, bar code readers, door-lock reader
- The output devices, such as 17″ Dell monitors, Hewlett–Packard laser jets model 5si, door-lock writers
- The storage devices, such as, HS 1120 magnetic tapes (tow digital data storage), RAID 5 hard drive equipped servers, smart-card (magnetic strip) as in room keys
- The communication devices, such as fiber optic wires, Cat 5 (twisted pair), router, a hub/switch, firewall for security, fax machines, phones, wave LAN antennas for the wireless equipment, processing devices—including servers such as an EMC/Compaq servers, Pentium III Dell computers (700 MHz), and the Micros terminals.

#### 1. Servers

The servers are cluster-based, that is, if one of the servers goes down the rest of the servers are not affected. The first server supports the front office database. The second server supports the marketing and sales database. The third server, possibly the SAP system or PeopleSoft, supports the back office or the accounting database. The fourth server is the Micros system that supports the kitchen and restaurant database.

RAID 5 (Distributed Data Guarding) is also called "stripe sets with parity." This level of RAID actually breaks data up into blocks, calculates parity, and then writes the data blocks in "stripes" to the disk drives, saving one stripe on each drive for the parity data. In RAID 5 configuration, if a drive fails, the controller uses the data on the parity drive and the data on the remaining drives to reconstruct data from the failed drive. This allows the system to continue operating with slightly reduced performance until the failed drive is replaced.

### 2. The Front Office

The Micros system has two forms of input devices: touch screen point-of-sale terminal and credit card readers. A touch screen monitor does nothing more than act like a mouse. POS software uses the inventory items themselves to track the sales.

### 3. The Marketing and Sales Office

Marketing and sales is responsible for booking banquets, conventions, and conferences and for preselling rooms. The input devices are keyboards, mice, floppies, and diskettes. The output devices are display and laser jet printers, and a web-based architecture that allows for plenty of booking on the Internet.

### 4. The Back Office

The back office is responsible for accounting such as accounts receivable, accounts payable, debits, and credits; preparing financial and managerial statements; and payroll. They are also responsible for the human resources and data bank. This office is usually supported by a SAP or PeopleSoft software which have a comprehensive range of business applications and support. The input devices which play a major role for the accounting activities are the keyboard, mouse, floppy disk, and barcode readers.

Window-based architecture and spreadsheet software such as Lotus 123 or Excel complete the back office functions.

For recovery purposes of accounting functions, backup and storage are done in the back office using magnetic tapes as the storage device to save all daily transactions.

## C. Case 3. I/O Environment— Food Chain Stores

Perhaps one of the most significant turning points for food-chain enterprises is the introduction of a smart-card. Customers using the smart-card at the check out receive immediate discounts. The customer's card usage also initiates the process of receiving future discount coupons or free products by mail.

It is interesting to note that food-chain store customers do not mind leaving behind a bit of their privacy for discounts and convenience, although some advocates of privacy have questioned the use of these types of cards.

In general, the principle reader of each food-chain store relies heavily on the POS terminals for its business transactions. The system is thus designed to function with the following input/output devices for the successful POS.

### 1. Input/Output

- Point-of-sale data starts at register
- Utilize IBM "Supermarket" package
- Registers are IBM 4690 POS terminals with Telxon barcode scanners
- Ethernet relay to in-store server(s), also running IBM 4690 OS

The checkout lane in each store is equipped with IBM 4690 POS (Point-of-Sale) terminals and Telxon bar code scanners, or their compatibles. The IBM 4690 systems software runs on all these terminals as well as the IBM database server in the "systems" room at each store, or a server in the front of the store. This may be a twisted pair (TP) Ethernet, coaxial, or fiber optics design. Data are stored within the store and are sent by "frame relay" to a server farm that feeds a data warehouse maintained at the corporate office. The transaction data, as well as the customer data, are stored using an Oracle data warehouse coupled with a RAID 5 type SAN storage system. This is where data integration, reporting, forecasting, and marketing is evaluated for regional competitiveness. To remain competitive, outputs from the following reports are further analyzed for efficiency and effectiveness:

- Automated sales checkout via bar code reader—% error and speed of customer service at check out time
- Sales and profitability reports by item, by department, and by vendor
- Customer history—who are the best customers

- Salesperson performance—both sales volume and profitability on those sales
- Sales tax reports—total sales, taxable sales, and sales tax collected for the period chosen
- Perpetual inventory—all transactions at the point-of-sale automatically update the inventory levels so inventory is current at all times.
- Accounts receivable—sales charged on a house account automatically update the customer's account.

## SEE ALSO THE FOLLOWING ARTICLE

Operating Systems

## BIBLIOGRAPHY

Baum, E., and Haussler, D. (1989). What size net gives valid generalization? *Neural Computations*, 1.1. (Technical Report by the International Biometric Consulting Group). Santa Cruz, CA: Santa Cruz University, Computer Research Laboratory.

Durbeck, R. C. (1988). *Output hardcopy devices*. Boston: Academic Press.

El-Bakry, H. M. (2001). Neural networks. *Proceedings, IJCNN '01, International Joint Conference,* Vol. 1, 577–582.

Fleck, R. A., and Honess, C. B. (1978). *Data processing and computers: An introduction*. Columbus, OH: Merrill.

Jain, A.K, Prabhaker, S., Hong, L., and Pankanti, S. (2000). Image Processing, *IEEE Transactions,* Vol. 9, No. 5, 846–859.

Kuroki, M., Yoneoka, T., Satou, T., Takagi, Y., Kitamura, T., and Kayamori, N. (1997). Emerging technologies and factory automation. *Proceedings, ETFA '97, 1997 6th International Conference,* 568–572.

Nutt, G. (2002). *Operating systems, a modern perspective,* 2nd ed. New York: Addison–Wesley Longman.

Pugh, E., Johnson, L., and Palner, J. (1991). *IBM's and early 370 systems.* Cambridge, MA: MIT Press.

Ranade, S. (1991). *Mass storage technologies,* Westport, CT: Meckler Co.

Raza, N., Bradshaw, V., and Hague, M. (1999). RFID technology. *IEE Colloquium* of Oct. 25, 1999, Vol. 1, No. 1, 1–5.

Roach, M. J., Brand, J. D., and Mason, J. S. D. (2000). Pattern recognition. *Proceedings, 15th International Conference,* Vol. 3, 258–261.

Schellenberg, K. (Ed.) (1998). *Computer in society,* 7th ed. Guilford, CT: Dushkin/McGraw–Hill.

Sriran, T., Vishwanatha Rao, K., Biswas, S., and Ahmed, B. (1996). Industrial electronics, control, and instrumentation. *Proceedings of the 1996 IEEE IECON 22nd International Conference,* Vol. 1, 641–646.

Stair, R., M., and Reynolds, G. W. (2001). *Principles of Information Systems,* 5th ed. Boston, MA: Course Technologies.

Sukthankar, R., and Stockton, R. (2001). *IEEE Intelligent Systems,* Vol. 16, No. 2, 14–19.

Zucker, C., and Rourke, J. (2001). *PC hardware: The complete reference.* New York: Osborne/McGraw–Hill.

# Computer-Integrated Manufacturing

**Asghar Sabbaghi and Ali R. Montazemi**
*Indiana University, South Bend*

## GLOSSARY

**bill of material (BOM)** A listing of all the raw materials, parts, subassemblies, and assemblies needed to produce one unit of finished product. The listing in the BOM file is heirarchical and it shows the quantity of each item needed to complete one unit of the following level of assembly.

**computer-aided design (CAD)** This is defined as a computer-based system that provides interactive graphic facilities to assist engineering in product and tool planning, design, drafting, design revisions, process planning, facilities planning, and design optimization. CAD is also used to maintain BOM and engineering revision information and software system.

**computer-aided manufacturing (CAM)** A computer-based system to program, direct, and control manufacturing process and handling equipment. These programs are based on the geometry of the parts and toll paths, both of which are captured by the CAD system. CAM serves as the primary automated communication link between the engineering and shop floor functions of a company, thus integrating design and manufacturing activities.

**computer-integrated manufacturing (CIM)** This is viewed as an integrated information system that links a broad range of manufacturing activities, including engineering design, flexible manufacturing systems, production planning and control, and all the business functions in any production/operation organization. The term was first used by Harrington in 1973 to envision the integration of computerized manufacturing systems at the shop floor level and management information systems at the corporate level. Since then, the concept has been expanded, both technologically and philosophically, to integrate information technologies into the entire organizational functions and units, including engineering, manufacturing, and business functions as well as all managerial level and operational functions.

**enterprise resource planning (ERP)** The ERP system, adopted in recent years by large and medium-size firms, is defined as strategic business solution that integrates all the business functions, including manufacturing, finance, and distribution. ERP systems encompass traditional transaction processing systems as well as model-based DSS such as data warehouse, supply chain optimization, planning, and scheduling systems. Such integrated systems improve management of information resources and enable decisionmakers to better access required information across the organization. Software vendors such as SAP AG, Baan, PeopleSoft, and Oracle provide a host of integrated ERP products.

**flexible manufacturing system (FMS)** This a group of machines designed to handle intermittent processing requirements and produce a variety of similar products. The system includes supervisory computer control, automatic material handling, and robots or other automated processing equipments.

**manufacturing resource planning (MRP II)** This is an expanded scope of MRP and a broader approach to production resource planning that involves not only those in MRP but also other areas, such as marketing and finance, of the firm in the planning process. A major purpose of MRP II is to integrate primary functions and other functions such as personnel, engineering, and purchasing in the planning process. The MRP II systems have also the capability of performing simulation, enabling managers to answer a variety of "what if" questions so that they can gain a better appreciation of available options and their consequences.

**master production schedule (MPS)** This is a timetable that specifies what to be made and when. The schedule must be in accordance with a production plan that sets the overall level of output in broad terms (e.g., product families, standard hours, or dollar volumes). The plan also includes a variety of inputs, including financial plans, customer demand, engineering capabilities, labor availability, inventory fluctuations, supplier performance, and other considerations.

**material requirement planning (MRP)** This is an information system designed to handle ordering and scheduling of dependent-demand inventories, e.g., raw materials, component parts, and subassemblies. A production plan for a specific number of finished products is translated into requirements for component parts and raw materials working backward from the due date, using lead times and other information to determine when and how much to order. Hence, requirements for end items generate requirements for low-level components, which are broken down by planning periods (e.g., weeks) so that ordering, fabrication, and assembly can be scheduled for timely completion of end items while inventory levels are kept reasonably low.

**model-based management system (MBMS)** A model-based management system makes use of quantitative models (i.e., Operations Research and Management Science techniques) to assist the decisionmaker in improving effectiveness of his/her decision processes.

**transmission control protocol–internet protocol (TCP–IP)** Two elements of a suite of internet protocols which include transport and application protocols. The combined acronym TCP–IP is commonly used for the whole suite. While the TCP element divides a message up into packets (call datagrams), recombining the message at the receiver, the IP element routs the package across the network.

## I. INTRODUCTION

The focus of Information Technology (IT) applications has shifted dramatically over the past two decades from efficiency/process improvement to strategic/competitive advantages. This change of focus has inspired organizations to take a business vision, rather than a technological focus, toward IT deployment and management. While computer-integrated manufacturing (CIM) has been recognized as one of the manifestations of IT evolution with much potential for strategic opportunities and competitive advantages, the traditional approach in management of technology, particularly focusing on manufacturing and engineering aspects of the system, in many corporations has remained as an obstacle to the successful implementation and management of CIM.

In practice, CIM has emerged as a new framework in the evolutionary process of development of new approaches and techniques in production/operations management such as just-in-time ( JIT), total quality management (TQM), flexible manufacturing systems (FMS), computer-aided design (CAD), computer-aided manufacturing (CAM), material requirement planning (MRP), and manufacturing resource planning (MRP II), among others. In this context, CIM provides a framework to integrate a variety of technologies in support of information requirements of the company's engineering, manufacturing, business, and management functions. This integration facilitates data communication and data sharing among various functional areas within an organization and thus supports decision-making processes, reduces lead time, and improves communication among decision makers within and outside of the organization.

Since the early 1970s, computers have been increasingly utilized in various operations of the production floor. CAD has extended design and drawing capabilities; MRP has reduced lead time, cut waste, and reduced down time; and software to support Master Production Scheduling (MPS) has increased efficiency and cut production costs. Production floor management of automated machines has been integrated with CAM to form a direct line between the drawings as conceived by the design engineer and the actual production of parts and products. Computer-aided engineering (CAE) has been integrated with CAD to allow an immediate analysis of designs as they take shape with CAD graphics. JIT inventory control, as a stockless or zero-inventory concept, is an integration of production schedules, warehouse management data, and material handling data, both within and between company files and those of subcontractors. While these advances have resulted in significant short- and long-term benefits, however, information technology in all these situations has been developed and adopted to achieve functional efficiency and measurable benefits. In many cases the sole objective of automation has been toward downsizing. Because of technological factors as well as organizational infrastructure, the managerial attitude toward exploring these technologies has been an engineering approach. Thus, making it often functional or departmental rather than company-wide strategy. The result has been "islands of automation" in which individual processes have been automated without much concern for integration and compatibility among them. These processes have had limited flexibility because of their lack of integration and the limited information-sharing capability.

The purpose of this study is to provide an overview of CIM and its component technologies, in a conceptual and managerial rather than a technical context, and to analyze the organizational environment and managerial approaches required for successful implementation and applications of CIM. In particular, CIM is viewed as the integration of computerized manufacturing systems at the shop floor (CMS) and Management Information Systems (MIS). This approach implies an integrated, company-wide, and MIS approach toward CIM implementation and management rather than traditionally functional and technological attitude. This study, therefore, emphasizes the key role of senior management and top MIS professionals in planning, selecting, justifying, implementing, and managing a CIM system. Responses to a survey targeted at various professionals and personnel involved in managing and using CIM will be used to identify and analyze some managerial perceptions that are critical to successful planning and implementation of CIM.

In Section II, we provide various views on the concept of CIM. Next, we present a conceptual schema of a CIM system and its major components. It is our contention that CIM should provide information to different organizational units. This must be achieved through well-integrated database and model-base systems, such as described in Section IV. In Section V, we elaborate on management issues and strategic dimension CIM initiatives. An integrated CIM system requires champions from all organizational units. However, recent literature reflects CIM as an engineering initiative, and, thus avoiding input from senior executives for linking CIM to the organizational strategy. Obviously, if this is found to be true, then organizations do not realize the full potential of CIM. Therefore, this conjecture was the basis of an empirical investigation that we conducted. The nature of this study and findings is reported in Section VI. Section VII provides a summary and concluding comments.

## II. CONCEPTION OF CIM AND LITERATURE REVIEW

Over the past two decades, information technology has significantly changed the way organizations compete, and has also changed the structure of the entire industries. Organizations that have used their business vision, experience, and their expertise to effectively integrate IT into their corporate strategy and organizational plans have gained much competitive advantage and success in the marketplace. In this context, CIM has surfaced as a concept that integrates information technologies into the entire organizational functions and units, including engineering, manufacturing, and business functions as well as all managerial level and operational functions. Since its first use in the term CIM has increasingly received a great deal of attention from production and inventory control managers, consultants, and scholars. Along with a JIT approach in manufacturing, TQM, and world class manufacturing, CIM has increasingly represented an important path that manufacturing companies could follow to improve their competitiveness in domestic and global markets.

CIM has been viewed and interpreted differently in the literature. There has been a tendency in the past to consider CIM as a purely technological endeavor in meeting short-term goals and as a quick payback solution to manufacturing problems. In this context, CIM has been viewed as the most advanced manufacturing technology known, or as the use of database and communication technologies to integrate the design, manufacturing, and business functions that comprise the automated segment of the facilities. It has also been defined as a complete application of computer-aided technology, and as the integration of CAD/CAM and production management (encompassing all activities, from planning and design of a product through its manufacturing and shipping. CIM has been viewed as a global approach which aims at improving industrial performance. This approach is applied in an integrated way to all activities from designing to delivery and after sale. The objectives of the CIM in this approach are to simultaneously improve productivity, decrease costs, meet due dates, increase product quality, and secure flexibility at local and global levels in a manufacturing system. In such an approach, economic, social, and human aspects are at least as important as technical aspects. On the one hand, CIM is viewed as almost synonymous with the use of machine vision, automated handling systems, robotics, and flexible manufacturing systems. On the other hand, CIM is primarily considered an information-based system to manage and structure databases. However, recent trends in the literature acknowledge CIM to be primarily a strategic system that draws on IT to help the firm better meet the needs of its market place. Thus, much of the rapidly proliferating literature on CIM has dwelt on its technological aspects and its potential.

Industry-specific definitions of CIM also are prevalent in the literature. The print industry, for example has adopted certain CIM principles such as electronic data interchange (EDI), digital linkage of users and suppliers, and automation in the production and binding steps to move the industry from a labor-intensive position to one of greater automation. The machine tool industry is another sector frequently cited as practicing certain CIM concepts such as CAD, computer-aided process planning (CAPP), CAM, and MRP II. For industry-specific definitions, it appears that CIM provides a vast menu of possibilities for an organization. From this list of possibilities, decisionmakers select various processes, hardware and software which are then adapted to specific industry needs.

Another emerging trend in CIM literature involves Product Data Management (PDM) systems. In today's manufacturing environments, the need for PDM is paramount because of rapidly changing product technology. In addition to PDM, the DBMS (for example, an Oracle system) must be capable of integrating with a particular network protocol (e.g., TCP-IP). This integration is essential for the development, delivery, and support of object-oriented client/server documents that are critical to business functions. For successful CIM installation, PDM cannot be overlooked because it is the backbone of data management in a CIM environment.

In practice, there is a wide array of CIM systems implemented at varying levels of sophistication with varying degrees of success. Particularly, in gaining considerable increases in productivity, reduction in lead time, and unit cost as well as in better machine utilization. For instance, GM redesigned its full-size pickup track operations at the Pontiac East Assembly Plant, Michigan, its largest manufacturing automation protocol (MAP) installation worldwide and used CIM to reduce redundancy and improve quality and customer service. This system handles almost the entire operation (99%) of truck cab and box welds, application of vehicle base paint by robots, and inspects truck boxes and doors by vision systems to ensure the accuracy of parts going through the welding process. Over 300 programmable controllers direct robots to different weld times to regulate the time and the pressure of welds.

Ford Motor Company plastic plant, which manufactures 10,000 bumper fascias a day, installed its CIM in 1986, expanding since then. The automated system coordinates batch runs of numerous styles and colors of fascias through the plant. After molding parts are placed by robots onto automated guided vehicles (AGVS), which bring the parts over to an automated storage and retrieval system (AS/RS). When needed to fill an order, fascias are picked out of the automated warehouse and placed on an automated electronic monorail (AEM) conveyer, which carries the parts to assembly and painting operations. Once a customer places an order, the system automatically uploads the customer order into Milan's scheduling system, which will alert the AS/RS to find the parts and deliver them to assembly and painting station. One software package provides a graphical overview of the entire (AEM) conveyors. It alerts the operator to a fault in the track, and switches and dispatches maintenance to fix problems. A second software package does the same for AGVS.

GW Plastics in Bethel, VT, one of the CIM pioneers and CIM award winners, has seen its custom injection molding business soar by about 20% a year since its first installation of CIM in 1986, and has greatly expanded it to embrace its capacity; including new molding plants in San Antonio, TX, and Tucson, AZ. As business grew, GW replaced its original mid-range computers with an IBM AS400 mini-mainframe for greater processing power and installed a wide-area network for managing data in support of financial and manufacturing information, engineering files, sales, shipping, backlog, and forecast data. GW's barcode system is integrated into AS400 to provide real-time verification of shipments to customers who demand JIT delivery.

Much of the literature on CIM suggests considerable improvements in productivity, lead time, and unit costs. However, the same studies also suggest that CIM systems appear not to have fully paid off in enhancing company-wide integration (technologically and organizationally), responsiveness, and business flexibility, as well as in supporting strategic decision-making processes. In some instances, the planning and implementation issues have led to disappointing results. Many firms do not achieve their objectives from CIM investments largely because the analytical methods used to justify these projects often do not capture the richness and underlying flexibility of new technologies. Swamidass (1994) has surveyed United States manufacturers and found that even though manufacturing technologies such as CAD, CAM, TQM, JIT, and computer numerical control (CNC) have been widely used at most plants, skilled use is relatively uncommon among them. Although factory automation seems to be critical to the survival of manufacturing firms, it is not clear from current MIS or operations management literature whether CIM has been used mainly as a ubiquitous support function or as an important element in achieving a sustainable competitive advantage in products or markets. CIM as a technology is giving way to the idea of CIM as a philosophy. This includes "the highest level of integrated automation in a manufacturing plant" and increasingly broader conceptions of enterprise management. According to Forrester and Hassard (1992), part of the problem is that what counts as the effective integration of systems can be company specific. Hill (1994) contends that the unwillingness of firms to undergo the necessary organizational changes is one reason for their inability to gain the full benefits of CIM. Organizational integration also has to match and support the technological integration in the move toward CIM. De Meyer (1990) investigated the computerized integration of several business functions, such as inventory planning, accounting, and product design in European firms. In a series of cross-sectional studies, he shows that the most often integrated functions include sales planning with master production scheduling and process control with quality reporting. His study does not address the technological impediments to integration or the relative contributions of the various enabling technologies to strategic management. Over time the decline in the cost of automation technologies has enabled manufacturers to adopt sophisticated production systems. Initial adoption of these systems has forced competitors to follow the industry leaders in the implementation effort or exit their industries.

## III. CONCEPTUAL SCHEMA OF CIM AND ITS MAJOR COMPONENTS

Conceptually, CIM can be viewed as an integrated system shown in Fig. 1. Applications of CIM are in support of effective decision making processes in various functional areas, such as engineering, manufacturing, marketing, and accounting. In this holistic view, functional areas are considered as subsystems and evaluated against the company's goals and strategic objectives, namely, optimization of total company-wide business, rather than suboptimization. CIM is an interactive system in support of various types of decision problems (i.e., unstructured, semi-structured, and structured decision problems).

From technical standpoint, CIM requires integration of information through common databases, hardware/software integration, and compatible software. In particular, model bases and databases are

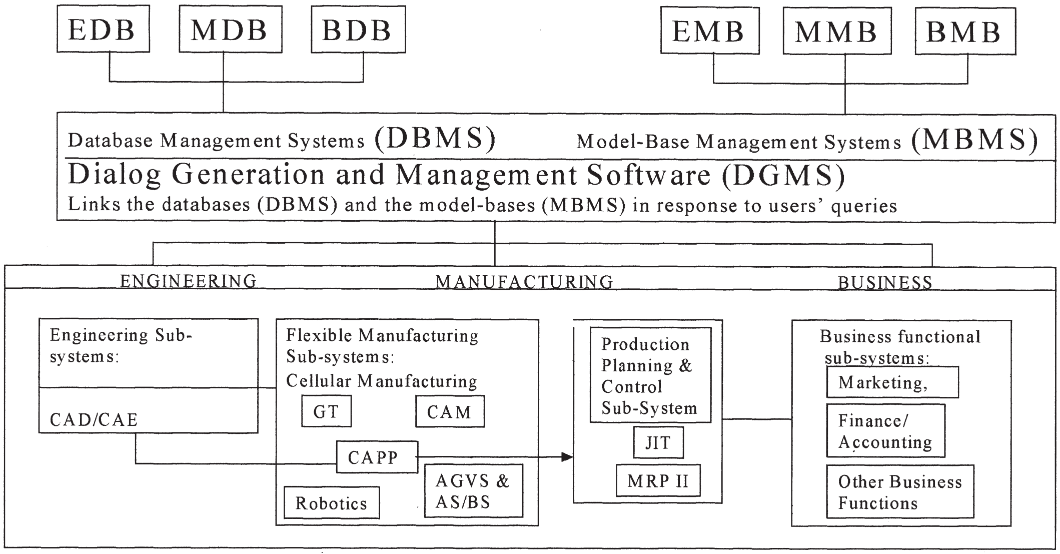

EDB
MDB
BDB
EMB
MMB
BMB
Database Management Systems (DBMS)
Model-Base Management Systems (MBMS)
Dialog Generation and Management Software (DGMS)
Links the databases (DBMS) and the model-bases (MBMS) in response to users' queries
ENGINEERING
MANUFACTURING
BUSINESS
Engineering Sub-systems:
CAD/CAE
Flexible Manufacturing Sub-systems:
Cellular Manufacturing
GT
CAM
CAPP
AGVS & AS/BS
Robotics
Production Planning & Control Sub-System
JIT
MRP II
Business functional sub-systems:
Marketing,
Finance/ Accounting
Other Business Functions

**Figure 1** A conceptual schema of CIM.

linked through model based management systems (MBMS) and database management systems (DBMS). These software modules coordinate the flow of information manipulation and transmission in response to queries by users in various functional areas through dialog generation and management software (DGMS). The model-base component consists of interrelated and integrated models based on methodologies gleaned from engineering, operations research, and management science in support of decisionmakers from various functional areas. The abstraction of the models and their linkages are constructed from numerous variables common to various functional areas at different levels of aggregation. Based on a large number of internal linkages in the model-base and the database, once the optimum design is determined, manufacturing process would define the BOM. Other subsequent functional models are deployed to support the production planning and financial and accounting analysis.

Fixture planning and programming is an important function of CAPP, which is the link between design and manufacturing (Fig. 1). This generates instructions for the production parts and translates design data into process plan to produce a part or a product. CAPP has several layers which must be integrated with other functions within the process planning and other CIM activities. The CAPP subsystem must provide information and programs for machine tools, robots, and fixturing subsystems to the production department. Furthermore, the production planning and control (PPC) subsystem requires information about the resources necessary to manufacture the product. This information may include the required machine tools, fixturing subsystems, and process time, among others, which would allow PPC to plan and schedule the required resources.

Another technical challenge in CIM development is to manage the huge amount of rapidly changing heterogeneous data, which may range from graphic and geometric data in engineering to textual and numeric data in manufacturing, sales, financial, accounting, and other business areas. These various types of data are to be integrated and shared through an underlying integrated database system.

## IV. INTEGRATION OF DATABASES AS THE CORE COMPONENT

Integration is the technical hub of CIM. One of the major requirements for a manufacturing system is the ability to store, manipulate, transmit, and display graphical data integrated with text. For example, CAD/CAM must be capable of supporting engineering analysis with heavy loads of numerical computation, as well as manufacturing and business analysis. For instance, CNC machines, as the advanced manufacturing technologies of CIM, provide the data communication networks and thus define the level of data integration in the system. Manufacturing automation protocol and standard for the exchange of product data (STEP) provide common specifications hardware integration and data formats to facilitate communication between heterogeneous systems. Data from different subsystems such as CAD and CAM are shared through an underlying CIM database system, in a coordinated and adaptable format. Islam (1997) presented an analysis of IDEF, an enterprise modeling technique and data flow diagram (DFD), to identify some of the requirements for CIM database systems. He has identified some of the main characteristics of data model to support a generic CIM. In particular, the data from various subsystems like engineering, manufacturing, marketing, and other functional areas are different in nature—accounting data and other business systems are numeric and alphanumeric, while the data for CAD subsystems are geometric and graphic. Therefore, a common database should integrate the heterogeneous data type. Since design data have to be shared by subsystems like CAPP, MRP II, CAM, and FMS, the data need to be converted into a neutral recognizable format. The common database requires integrity constraints, such as a consistency checking routine that checks the validity between two design elements for their assembly. The CAPP subsystem generates a process plan based on the geometric and engineering data of the CAD subsystem as well as data from MRP II subsystems. The CAD data for BOM are part input data for MRP II subsystems, where geometric data are converted to alphanumeric data. The MRP II provides work center data, and purchase order and material plan which are used by other subsystems. The output from MRP II includes various data such as manufacturing order and inventory status, which are inputs to a CAM subsystem. CAM also uses design data from CAD and routine from CAPP subsystems. The FMS subsystem is a combination of heterogenous hardware and software monitoring the manufacturing operations and thus should have updated information for CAM, the production plan, and other necessary changes in the system. Another issue in this context is the integration of the product data management (PDM) and enterprise resource planning (ERP). Companies often struggle in determining how these two technologies, PDM and ERP, can best work together, what tasks each should handle, who should control information, and how they should be linked.

PDM evolved from engineering efforts in the 1980s and was first used primarily to manage engineering drawings and related CAD files (Miller, 1999). The technology eventually expanded to include management of data from a variety of separate applications including not only mechanical drafting, solid modelers, and structural analysis, but also electronic CAD (ECAD), NC programming, and others such as technical publications and office applications. In this capacity, PDM systems manage a variety of engineering data and processes including design geometry, project plans, part files, assembly diagrams, analysis results, correspondence, BOMs, specifications, engineering changes, approval processes, product structure, parts classification and retrieval, configuration management, program management, authorizations, workflow, and others.

Enterprise resource planning systems trace their roots back to manufacturing initiatives in the 1960s, evolving from MRP systems for inventory planning and control and later MRP II technology which expanded into shop floor scheduling and coordination. These systems broadened into what is now termed ERP, which encompasses numerous aspects of production such as inventory control, shop scheduling, capacity planning, and master scheduling as well as purchasing, sales, accounting, finance, and even human resources. By coordinating production operations for peak efficiency, ERP has become virtually indispensable for manufacturers to reduce manufacturing lead time and cost as well as to facilitate teamwork and collaboration on the factory floor. As PDM and ERP have both expanded, their increased functionality has resulted in overlap of some functions such as BOMs, parts classification, component information, configuration management, process workflow, and program management.

Transferring information quickly and accurately between engineering and manufacturing speeds workflow, improves communication throughout the organization, and avoids redundant efforts along with reducing associated errors and delays in recreating data. This provides an ability to leverage efforts so that data already entered in one area do not have to be recreated in another. In many cases, product information is transferred when design engineering releases a PDM product definition to industrial engineers, who then convert it into an assembly view for ERP. Often, a more direct exchange is being used where the PDM product structure is sent via a translator directly into ERP. A few systems even provide two-way translation between PDM and ERP, giving engineering and manufacturing managers a direct view of one another's database. Still, in some other cases, companies strive for the most tightly integrated system where overlapping portions of PDM and ERP information are stored in the same database and shared by both systems.

Companies must identify a solution that can effectively integrate PDM and ERP systems and can closely meet their individual needs. Often, this means carefully evaluating company operations, goals, and procedures to identify how data such as BOMs can be shared, exchanged, and to ensure that the right information in the right format is available to the right person on a timely manner. As an example, fuel gauging and proximity sensing system manufacturer Eldec (a division of the Crane Co., Lynnwood, WA) masters all parts and BOMs in its PDM system and transfers information to its ERP system automatically at predefined times. Only six attributes are mastered in PDM, which establishes information such as purchase or fabricated part type and possible alternates. The data is sent to ERP, where an integration tool and template help create the 140 attributes needed for manufacturing. Production personnel fill in the remaining data and maintain control of information throughout manufacturing. To expedite the transfer of information from PDM to ERP, Eldec developed programs to automatically extract BOM data from schematics in PDM and put it into an Excel spreadsheet in a format that can be loaded directly into the ERP system.

ERP denotes control and management of the entire manufacturing facility in areas including not only production but also purchasing, finance, and engineering. It has evolved from earlier MRP systems for inventory control and later MRP II technology for shop floor scheduling and coordination. By coordinating the manufacturing operations for peak efficiency, ERP has become virtually indispensable for manufacturers to reduce manufacturing time and cost as well as facilitate teamwork and coordination. Similarly, PDM systems manage product-related information throughout the enterprise including design geometry, engineering drawings, project plans, part files, assembly diagrams, and product specifications. Typical users have traditionally included designers and engineers, but PDM system usage is being expanded to include other areas including manufacturing, sales, marketing, purchasing, shipping, and finance. Because of significant overlaps in data and functionality between PDM and ERP, most companies view integration between the two systems appealing.

The scope of integration in CIM is far beyond merely the hardware/software and data integration and networking: the human and organizational aspects of CIM play a critical role in well-deliberated planning, successful implementation, and effective management of CIM.

## V. STRATEGIC PLANNING, IMPLEMENTATION, AND MANAGEMENT ISSUES OF CIM

As we discussed earlier, CIM consists of overall integration of various functional areas from engineering to manufacturing, inventory, sales, marketing, etc., as well as integration of databases that support various levels of managerial decision problems. Since CIM requires such fundamental changes in organizational communication, both technologically and functionally, as well as massive capital investment, it is the senior management responsibility, with its long-range, company-wide view to integrate CIM strategy into the corporate strategy, to evaluate all possible strategies and intangible benefits, to judge its feasibility, and to decide on its adoption.

It has been reported that the failure of CIM to live up to its promise of a fully integrated information system has been due to a lack of commitment from senior management. While senior managers have taken full advantage of such innovative concepts as TQM and JIT scheduling, they have not fully utilized the economic benefits of CIM. Lack of understanding at the corporate level about technology management and its potential have been cited for the failure. More specifically, for nontechnical managers, it has been easier to envision the applications of the TQM and JIT concepts, and to measure their benefits. However, CIM has been perceived as a technological initiative that requires highly sophisticated hardware and software integration. Furthermore, the characterization of CIM technology has been depended on by the manufacturing industry and thus viewed as a highly technological issue rather than management issue.

Due to the technology orientation of CIM, it has been predominantly implemented by engineers and line managers. Consequently, the objective has been to improve efficiency and reduce costs rather than on improving corporate-wide integration of information systems. The engineering approach in development of CIM has also been implemented from bottom up where individual functional areas and lines have undertaken CIM projects based on their local needs and on a piecemeal approach, and thus has led to islands of automation, focusing on localized benefits. In particular, the lack of integration among functional areas such as marketing, purchasing, and production planning, and across key technologies such as MRP, CAD, and robotics, has been a stumbling block in the successful implementation and management of CIM processes. This is partly due to the lack of leadership from MIS professionals.

MIS professionals have traditionally focused on business functional areas such as accounting, finance, and planning data, and have shied away from robots and programmable controllers. Thus, MIS departments are lagging behind other departments in their contribution to CIM development.

To ameliorate this problem, senior management, particularly chief information officers, must approach CIM as a tool for competitive advantage in the market place, and integrate CIM strategy into corporate strategy and consider all possible strategic and intangible benefits as well as tangible ones. CIM technology contributes to competitive advantage by being responsive to the market changes and being flexible to redesign and manufacture according to market conditions, as well as in identifying the needs of a specialized market and responding effectively and efficiently to those needs.

Given the dimension and the characteristics of CIM strategy, it embraces considerations for the office and the business of the future as well as the factory of the future. The CIM strategy involves a dramatic change in manufacturing and business philosophy, as these changes will affect the entire corporation. Therefore, the need for a strategic plan detailing how a manufacturing and business concern can be addressed, via CIM, involves similar problems and issues inherent in information technology and MIS planning. The CIM strategy has to be viewed from an information technology management perspective and must be integrated into long-range business strategy by senior management. Senior managers must be involved in the CIM strategy development as the participants rather than spectators. Their roles should be shifted from being represented by subordinates to ones who create business strategy with CIM as an integral component. In this context, senior management can drive CIM strategy as an inseparable component of the overall business strategy.

## VI. EMPIRICAL STUDY

The integration of various functional departments and managerial levels remains as a critical factor in design, development, implementation, and management of CIM systems. Managers, particularly those in small to mid-size companies, may assume CIM to represent a potential loss of management control, and a major challenge to their position and authority within the company. In their view, for instance, CIM may change the way the company plans, schedules, executes, and tracks production decisions. Much of the skills and knowledge possessed by the current management may no longer be pertinent to this new CIM-run environment; moreover, they may feel that they no longer understand the operations in CIM environment. Other employees may also be adversely disposed toward CIM. For example, they may view CIM aimed at narrowly automating much of the functions and reducing labor force rather than on developing knowledge workers and on improving the productivity and innovation. As a result, they may experience fear, uncertainty, and dread.

In order to examine the managerial issues that were previously discussed, a number of manufacturing companies in St. Joseph County, IN, were surveyed with regard to their adoption and implementation of CIM. The survey was carried out through a structured questionnaire format. A contact person in each company was identified for distribution of the forms among employees and collecting them after completion. An effort was made to include a wide variety of functional areas, types of positions, and managerial levels in the survey. The questionnaire included two different parts: (1) interviewer and organization identification: company size, position of interviewee, gender, the interviewers' level of education, and their perception of CIM impact on employment and on the stress to be placed on CIM initiatives; and (2) a set of open-ended questions. From a total of 211 respondents, 1.2% were employed in small companies (companies with less than 100 employees), 33.5% were employed in medium-size companies (with 100–500 employees), and 64.5% were from large companies (having more than 500 employees). The data in this survey thus represent mostly large and mid-size companies.

The survey consists of various employees: 23.9% of the respondents were from the management level, 51.2% professionals, and 24.9% clerical and other support personnel. With regard to the level of education, 12.2% of respondents had high school degrees or less, 13.8% had completed some college work, 22.8% had a bachelor's degree, 25.4% had carried out some graduate studies, 23.8% had completed a graduate degree, and 2.1% had professional training. Hence all educational levels except professional training are well represented in the survey. Various types of professionals were represented: 23.4% representing management of CIM, IS professionals made 23.9%, CIM professionals 27.9%, and clerical operators represented 24.9% of the respondents. Most companies that were visited during the data collection had computer systems of some sort. Often stand-alone accounting and inventory systems were in use, but these could not be easily interfaced with other manufacturing applications.

With regard to the computerization of the functional departments in their firms of those surveyed, 74.7% came from those with a fully computerized accounting department, 64.2% from firms with computerized MRP/MAP, 60.5% from those with computerized inventory, 54.7% from companies with computerized sales planning, 67.9% from computerized shop floor control, 76.3% from those with computerized process control, 71.1% from those with computerized quality reporting, and 16.3% from firms with other computerized departments. Thus, firms with computerized departments are well represented in the survey.

We have examined the perceptions of the respondents, through the analysis of variance (ANOVA), regarding their expected impact of CIM on the number of employees and their expected stress on CIM over the next two years. Table I reports the results of the statistical analysis when we focused on comparison of the perception of various groups of respondents. We examined the perceptions of various groups designated by the size of the companies, the gender of the respondents, the employment position, and more importantly the managerial positions of the respondents in the survey to the impact of CIM on employees and the stress to be placed on CIM initiatives.

We have used one-way ANOVA to analyze the perceptions of various respondents on the impact of CIM on employment and the level of stress to be placed on CIM initiative and management during the next two years. More specifically, we have examined the respondents perceptions by a number of factors: the respondent's company size, gender, education, and job title. In order to test some hypothesis in this context, we have formulated the following null hypotheses:

$H_1$—The perception of the users regarding the impact of CIM on employment does not depend on the gender.

$H_2$—The perception of the users regarding the stress to be placed on CIM initiatives does not depend on the gender.

$H_3$—The perception of the users regarding the impact of CIM on employment does not depend on their company's size.

$H_4$—The perception of the users regarding the stress to be placed on CIM initiatives does not depend on their company's size.

**Table I** One-way ANOVA: Perception of Various Groups of Employees toward the Impact of CIM on Employment and Stress on CIM

| By gender | | Sum of squares | d.f. | Mean square | F | Sig. |
|---|---|---|---|---|---|---|
| CIM-IMP | Between groups | 1.435 | 1 | 1.435 | 2.669 | 0.104 |
| | Within groups | 112.394 | 209 | 0.538 | | |
| | Total | 113.829 | 210 | | | |
| Stress | Between groups | .557 | 1 | 0.557 | 0.465 | 0.496 |
| | Within groups | 249.974 | 209 | 1.196 | | |
| | Total | 250.531 | 210 | | | |
| **By the size of the companies** | | **Sum of squares** | **d.f.** | **Mean square** | **F** | **Sig.** |
| CIM-IMP | Between groups | 1.254 | 3 | 0.418 | 0.769 | 0.513 |
| | Within groups | 112.575 | 207 | 0.544 | | |
| | Total | 113.829 | 210 | | | |
| Stress | Between groups | 1.904 | 3 | 0.635 | 0.528 | 0.663 |
| | Within groups | 248.627 | 207 | 1.201 | | |
| | Total | 250.531 | 210 | | | |
| **By education** | | **Sum of squares** | **d.f.** | **Mean square** | **F** | **Sig.** |
| CIM-IMP | Between groups | 1.904 | 6 | 0.317 | 0.578 | 0.747 |
| | Within groups | 111.925 | 204 | 0.549 | | |
| | Total | 113.829 | 210 | | | |
| Stress | Between groups | 13.809 | 6 | 2.301 | 1.983 | 0.070 |
| | Within groups | 236.722 | 204 | 1.160 | | |
| | Total | 250.531 | 210 | | | |
| **By the title of respondents** | | **Sum of squares** | **d.f.** | **Mean square** | **F** | **Sig.** |
| CIM-IMP | Between groups | 1.580 | 3 | 0.527 | 0.971 | 0.407 |
| | Within groups | 112.250 | 207 | 0.542 | | |
| | Total | 113.829 | 210 | | | |
| Stress | Between groups | 62.563 | 3 | 20.854 | 22.966 | 0.000 |
| | Within groups | 187.967 | 207 | 0.908 | | |
| | Total | 250.531 | 210 | | | |

$H_5$—The perception of the users regarding the impact of CIM on employment does not depend on the managerial and functional title of the respondents.

$H_6$—The perception of the users regarding the stress to be placed on CIM initiatives does not depend on the managerial and functional title of the respondents.

$H_7$—The perception of the users regarding the impact of CIM on employment does not depend on the education of the user.

$H_8$—The perception of the users regarding the stress to be placed on CIM initiatives does not depend on the user's education.

These hypotheses can be formulated as follows. We use $AP_i$ (X) to represent the average perception of group i of respondents toward the specific dimension of CIM that is under examination, where for gender: i = M (male), and i = F (female); for size: i = s (small), m (mid-size), and l (large); for education: i = 1 (high school or less), 2 (some college), 3 (bachelor's degree), 4 (some graduate study), 5 (graduate degree), and 6 (professional training); and for job title dimension: i = a (CIM managers), b (IS professionals), c (CIM professionals), and d (clerical and operating personnel). For instance, $AP_M$(CIM-IMP) denotes the average perception of males toward the CIM impact on employment, $AP_M$(Stress) represents the average perception of males toward stress to be placed on CIM initiatives and management, and $AP_b$(Stress) denotes the average perception of IS professionals toward stress to be placed on CIM initiatives and management.

Using these definitions, we can formulate the above hypotheses, respectively, in the following mathematical forms. For the gender factor:

$H_1$: $AP_M(\text{CIM-IMP}) = AP_F(\text{CIM-IMP})$
$H_2$: $AP_M(\text{Stress}) = AP_F(\text{Stress})$

For the company size factor:

$H_3$: $AP_s(\text{CIM-IMP}) = AP_m(\text{CIM-IMP}) = AP_1(\text{CIM-IMP})$
$H_4$: $Ap_s(\text{Stress}) = AP_m(\text{Stress}) = Ap_1(\text{Stress})$

For the education factor:

$H_5$: $AP_1(\text{CIM-IMP}) = AP_2(\text{CIM-IMP}) = AP_3(\text{CIM-IMP}) = AP_4(\text{CIM-IMP}) = AP_5(\text{CIM-IMP}) = AP_6(\text{CIM-IMP})$
$H_6$: $AP_1(\text{Stress}) = AP_2(\text{Stress}) = AP_3(\text{Stress}) = AP_4(\text{Stress}) = AP_5(\text{Stress}) = AP_6(\text{Stress})$

For the job responsibility and title factor:

$H_7$: $AP_a(\text{CIM-IMP}) = AP_b(\text{CIM-IMP}) = AP_c(\text{CIM-IMP}) = AP_d(\text{CIM-IMP})$
$H_8$: $AP_a(\text{Stress}) = AP_b(\text{Stress}) = Ap_c(\text{Stress}) = AP_d(\text{Stress})$

One-way ANOVA, as shown in Table I, suggests that the first three null hypotheses can be accepted, i.e., there were no significant differences among various groups of respondents by the size of companies, by the gender, and by education of respondents, as they perceive the impact of CIM on employment and the stress to be placed on CIM. However, the data reported two different results for the fourth null hypothesis ($H_4$): While CIM management, CIM and IS professionals, and operating personnel have the same perception toward the impact of CIM, they show significantly different perceptions toward the stress to be placed on CIM initiatives and management.

The result of ANOVA rejects the fifth hypothesis ($H_5$) that the various employees have similar perceptions of stress on CIM. We extend our analysis to pinpoint whether the observed differences in the survey could be attributed to just the natural variability among the sample averages or whether there is reason to believe that some of the four groups have significantly different perceptions for the stress to be placed on CIM over the next two years. By extending our analysis to a multiple comparison procedure, using Bonferroni, as shown in Table II, one can see no significant difference between the perception of CIM managers and CIM professionals on the one hand and between IS professionals and clerical/operating personnel on the other hand. However, the first two groups, CIM managers and CIM professionals, have significantly different perceptions regarding the stress to be placed on CIM initiatives and management. In particular, IS professionals show significantly lower stress on CIM projects.

We further expanded our analysis of the data to include more than one grouping or factor variable. In particular, by reviewing the one-factor analysis, we combined employment title and the educational background of the respondents in our two-way analysis, as shown in Table III. There was no significant evidence to conclude that there exists interaction between the job

**Table II** Multiple Comparisons (Bonferroni) of Respondents' Perceptions by the Employment Position

**Dependent Variable: STRESS**

| **By the employment position** | | | | | **95% Confidence interval** | |
|---|---|---|---|---|---|---|
| **(I) TITLE** | **(J) TITLE** | **Mean difference (I-J)** | **Std. error** | **Sig.** | **Lower bound** | **Upper bound** |
| CIM managers | IS professionals | 1.17[a] | 0.19 | 0.000 | 0.67 | 1.68 |
| | CIM professionals | −8.67E-02 | 0.18 | 1.000 | −0.57 | 0.40 |
| | Clerical and operation | 0.88[a] | 0.19 | 0.000 | 0.39 | 1.38 |
| IS professionals | CIM managers | −1.17[a] | 0.19 | 0.000 | −1.68 | −0.67 |
| | CIM professionals | −1.26[a] | 0.19 | 0.000 | −1.76 | −0.77 |
| | Clerical and operation | −0.29 | 0.19 | 0.749 | −0.80 | 0.21 |
| CIM professionals | CIM managers | 8.67E-02 | 0.18 | 1.000 | −0.40 | 0.57 |
| | IS professionals | 1.26[a] | 0.19 | 0.000 | 0.77 | 1.76 |
| | Clerical and operation | 0.97[a] | 0.18 | 0.000 | 0.48 | 1.46 |
| Clerical and operation | CIM managers | −0.88[a] | 0.19 | 0.000 | −1.38 | −0.39 |
| | IS professionals | 0.29 | 0.19 | 0.749 | −0.21 | 0.80 |
| | CIM professionals | −0.97[a] | 0.18 | 0.000 | −1.46 | −0.48 |

[a] The mean difference is significant at the 0.05 level.

**Table III** Tests of Interaction between Factors: Impact of CIM and Stress on CIM

**Dependent Variable: CIM Impact on the Number of Employees**

**Job title & the company size**

| Source | Type III sum of squares | d.f. | Mean square | F | Sig. |
|---|---|---|---|---|---|
| Corrected model | 9.226 | 10 | 0.923 | 1.764 | 0.069 |
| Intercept | 56.705 | 1 | 56.705 | 108.420 | 0.000 |
| Title | 6.874 | 3 | 2.291 | 4.381 | 0.005 |
| Size | 1.432 | 3 | 0.477 | 0.913 | 0.436 |
| Title * Size | 6.606 | 4 | 1.652 | 3.158 | 0.015 |
| Error | 104.603 | 200 | 0.523 | | |
| Total | 934.000 | 211 | | | |
| Corrected total | 113.829 | 210 | Squared = 0.081 (adjusted R squared = 0.035) | | |

| Job title and gender | Type III sum of squares | d.f. | Mean square | F | Sig. |
|---|---|---|---|---|---|
| Corrected Model | 5.642 | 7 | 0.806 | 1.512 | 0.165 |
| Intercept | 599.524 | 1 | 599.524 | 1124.932 | 0.000 |
| Title | 1.682 | 3 | 0.561 | 1.052 | 0.371 |
| Gender | 0.879 | 1 | 0.879 | 1.650 | 0.200 |
| Title * Gender | 2.893 | 3 | 0.964 | 1.809 | 0.147 |
| Error | 108.187 | 203 | 0.533 | | |
| Total | 934.000 | 211 | | | |
| Corrected total | 113.829 | 210 | a. R squared = 0.050 (adjusted R squared = 0.017) | | |

| Job title and education | Type III sum of squares | d.f. | Mean square | F | Sig. |
|---|---|---|---|---|---|
| Corrected model | 16.658 | 22 | 0.757 | 1.465 | 0.090 |
| Intercept | 232.932 | 1 | 232.932 | 450.660 | 0.000 |
| Title | 1.943 | 3 | 0.648 | 1.253 | 0.292 |
| Education | 2.564 | 6 | 0.427 | 0.827 | 0.551 |
| Title * Education | 13.277 | 13 | 1.021 | 1.976 | 0.025 |
| Error | 97.171 | 188 | 0.517 | | |
| Total | 934.000 | 211 | | | |
| Corrected total | 113.829 | 210 | a. R squared = 0.146 (adjusted R squared = 0.046) | | |

**Dependent Variable: STRESS**

| Job title and company size | Type III sum of squares | d.f. | Mean square | F | Sig. |
|---|---|---|---|---|---|
| Corrected Model | 68.092 | 10 | 6.809 | 7.465 | 0.000 |
| Intercept | 226.731 | 1 | 226.731 | 248.556 | 0.000 |
| Title | 43.736 | 3 | 14.579 | 15.982 | 0.000 |
| Size | 1.018 | 3 | 0.339 | 0.372 | 0.773 |
| Title * Size | 5.076 | 4 | 1.269 | 1.391 | 0.238 |
| Error | 182.439 | 200 | 0.912 | | |
| Total | 2442.000 | 211 | | | |
| Corrected total | 250.531 | 210 | a. R squared = 0.272 (adjusted R squared = 0.235) | | |

| Job title and gender | Type III sum of squares | d.f. | Mean square | F | Sig. |
|---|---|---|---|---|---|
| Corrected Model | 64.646 | 7 | 9.235 | 10.085 | 0.000 |
| Intercept | 1542.226 | 1 | 1542.226 | 1684.226 | 0.000 |
| Title | 48.096 | 3 | 16.032 | 17.508 | 0.000 |
| Gender | 0.421 | 1 | 0.421 | 0.460 | 0.499 |
| Title * Gender | 1.665 | 3 | 0.555 | 0.606 | 0.612 |
| Error | 185.885 | 203 | 0.916 | | |
| Total | 2442.000 | 211 | | | |
| Corrected total | 250.531 | 210 | a. R squared = 0.258 (adjusted R squared = 0.232) | | |

*(continues)*

**Table III** *(continued)*

| Job title and education | Type III sum of squares | d.f. | Mean square | F | Sig. |
|---|---|---|---|---|---|
| Corrected Model | 95.594 | 22 | 4.345 | 5.272 | 0.000 |
| Intercept | 793.582 | 1 | 793.582 | 962.930 | 0.000 |
| Title | 11.842 | 3 | 3.947 | 4.790 | 0.003 |
| Education | 2.872 | 6 | 0.479 | 0.581 | 0.745 |
| Title * Education | 25.306 | 13 | 1.947 | 2.362 | 0.006 |
| Error | 154.937 | 188 | 0.824 | | |
| Total | 2442.000 | 211 | | | |
| Corrected total | 250.531 | 210 | a. R squared = 0.382 (adjusted R squared = 0.309) | | |

title of respondents and their gender, and the size of their companies. Therefore, one cannot reject the sixth null hypothesis ($H_6$), that there is no interaction between the employment title and the gender as well as between the employment title and the size of the company with regard to stress on CIM. In other words, the effect of job title and responsibility on placing stress on CIM seem to be similar for males and females and for respondents from mid-size and large companies. However, there was a significant interaction between the job title and the education of employees. The significant interaction between employment title and educational background of respondents tells us that it is reasonable to believe that the difference in perception among respondents with different education is significantly different for any specific job title of the employees. Thus the relationship between job title and perception of employees is different with educational background, i.e., the eighth null hypothesis ($H_8$) is not supported.

## VII. CONCLUDING COMMENTS

CIM in this study was considered as the (vertical) integration of the flow of information from factory to the board room with its focus on business and also it is the (horizontal) integration of functional areas from design and manufacturing, to marketing and other business functions. It is, thus, a strategic challenge for senior management, particularly for CIOs, to integrate CIM strategy into corporate strategy to ensure the company's competitive effectiveness.

Our empirical study in this investigation was conducted to analyze the perception of various users, including IS professionals and CIM managers, with regard to the impact of CIM on employment and the stress to be placed on CIM projects. According to the results of this empirical study, summarized in Table IV, two types of perceptions were recognized: one perception, shared by CIM managers and professionals,

**Table IV** Summary Results of the Statistical Analysis

| Null hypothesis | Statement of the hypothesis | Empirical result |
|---|---|---|
| $H_1$ | The perception of the users regarding the impact of CIM on employment does not depend on the gender | Supported |
| $H_2$ | The perception of the users regarding the stress to be placed on CIM initiatives does not depend on the gender | Supported |
| $H_3$ | The perception of the users regarding the impact of CIM on employment does not depend on their company's size | Supported |
| $H_4$ | The perception of the users regarding the stress to be placed on CIM initiatives does not depend on their company's size | Not supported |
| $H_5$ | The perception of the users regarding the impact of CIM on employment does not depend on the managerial and functional title of the respondents | Not supported |
| $H_6$ | The perception of the users regarding the stress to be placed on CIM initiatives does not depend on the managerial and functional title of the respondents | Supported |
| $H_7$ | The perception of the users regarding the impact of CIM on employment does not depend on the education of the user | Not supported |
| $H_8$ | The perception of the users regarding the stress to be placed on CIM initiatives does not depend on the user's education | Not supported |

judged the impact of CIM insignificant, while another, expressed by IS professionals and clerical operators, viewed the impact of CIM as labor replacement—i.e., decreasing the number of employees. The latter judgement was also displayed by the majority of employees from the mid-size companies.

According to the data, the perceived stress on CIM did not depend on the organizational size, but rather upon the type of responding employee. Two groups with two different perceptions were identified: CIM managers and professionals viewed that there would be significant stress on CIM, whereas IS professionals and clerical operators were not as optimistic. Our analysis suggests that IS professionals perceive the stress on CIM to be significantly different from CIM managers and CIM professionals. In particular, IS professionals have placed less emphasis on CIM undertakings and their management, and thus lagged behind other departments in contributing to the CIM development and its strategic applications for competitive edge. These data also suggest that training and education among CIM managers and CIM professionals has played a positive role in placing higher stress on CIM technology.

Successful implementation and management of CIM initiative depend not only upon expertise in technology assessment and deployment, but more importantly, on the business vision and a company-wide management approach. Effective assessment in deployment of CIM technology, in particular, should take into consideration its potential values and nontangible as well as tangible costs and benefits. Given the scope of CIM technology, senior IS management with its long-range and company-wide view is responsible for playing a leading role in CIM undertakings, for estimating all possible strategic and intangible benefits resulting from CIM, and for leading the feasibility analysis and the adoption process. The lack of leadership by senior IS management in CIM initiatives and management will inevitably lead to the automation of functional tasks without achieving the integration that is vital to the survival and functioning of the company as a whole.

## SEE ALSO THE FOLLOWING ARTICLES

Computer-Aided Design • Computer-Aided Manufacturing • Enterprise Resource Planning • Operations Management • Productivity • Supply Chain Management • Total Quality Management and Quality Control

## BIBLIOGRAPHY

Aronson, R. (1995). Lead winners find CIM is key to improvement. *Manufacturing Engineering,* Vol. 115, No. 5; 3–69.

Davenport, T. H. (2000). *Mission Critical: Realizing the Promise of Enterprise Systems.* Boston: Harvard Business School Press.

DeGaspari, J. (1995). Ten years after. *Plastics Technology,* Vol. 41, No. 11, 46–48.

De Meyer, A. (1990). How to arrive at computer integrated manufacturing: A 3-year survey. *European Journal of Operational Research,* 47, 29–247.

Doumeints, G., Vallespir, B., and Chen, D. (1995). Methodologies for designing CIM systems: A survey. *Computer in Industry,* No. 25, 263–280.

Fjemestand, J., and Charabarti, A. (1993). A survey of computer-integrated manufacturing literature: A framework of strategy, implementation, and innovation. *Technology Analysis and Strategic Management,* Vol. 5, No. 3, 251–71.

Forrester, P., and Hassard, J. (1992). The CAPM/CAE interface within CIM system approaches in medium-sized companies. *Computing and Control Engineering Journal,* Vol. 3, No. 2, 75–78.

Fraser, J. (March 1998). 300-mm: How will production software keep up? *Solid State Technology,* Vol. 41, No. 31, 30–33.

Harrington, J., Jr. (1973). *Computer Integrated Manufacturing.* New York: Industrial Press.

Hill, M. (1994). Computer-integrated manufacturing: Elements and totality. *New Wave Manufacturing Strategy* ( J. Storey, ed.), pp. 122–150, London: Paul Chapman.

Islam, A. (1997). Deviation of selection criteria of CIM database using IDEF. *Computer and Industrial Engineering,* Vol. 33, Nos. 1–2, 31–34.

Johanson, J., Karmarkar, U. S. (1995). Computer integrated manufacturing: Empirical implications for industrial information systems. *Journal of Management Information Systems,* Vol. 12, No. 2, 59–83.

King, W. R., and Ramamurthy, K. R. (1992). Do organizations achieve their objectives from computer-based manufacturing technologies? *IEEE Transaction on Engineering Management,* Vol. 39, No. 2, 129–141.

Lamparter, W. (April 1996). The next revolution? *American Printer,* Vol. 217, Issue 1, 34–41.

Lamparter, W. (September 1997). CIM for print. *American Printer,* Vol. 219, Issue 6, 48–53.

Miller, E. (March 1999). Integrating PDM and ERP. *Computer-Aided Engineering,* Vol. 18, No. 3, 69–74.

Newman, S. (1994). *Strategic Information Systems.* Toronto: Macmillan.

Noitove, M. H. (November 1995). CIM stars in tooling development. *Plastics Technology,* Vol. 41, No. 11, 44–46.

Ogando, J. (March 1998). New entry in CIM systems, let's start small and expand later. *Plastics Technology,* Vol. 44, No. 3, 18–20.

Procter, S., and Brown, A. D. (1997). Computer-integrated operations: The introduction of hospital information support systems. *International Journal of Operations and Production Management,* Vol. 17, No. 7/8, 746–755.

Zambelli, J., and Kelley, W. ( July 1998). U.S. steel: Re-engineering for the future. *Steel-Times International,* Vol. 22, No. 3, 24–26.

# Computer-Supported Cooperative Work

**Judith S. Olson and Gary M. Olson**
*University of Michigan*

## GLOSSARY

**asynchronous** Turn taking in conversation that is delayed. E-mail and conversation databases (like Lotus Notes) facilitate conversation without requiring the participants to be available to each other at the same time.

**backchannel** Responses that a listener utters during the speaker's speech to signal to the speaker that he or she understands or not, agrees or not, etc. They consist of short grunts and "uh huhs" or head nods and furled brows.

**collaboratory** A laboratory without walls, the collection of technologies to connect scientists to each other, to remote instruments, and to digital libraries.

**groupware** The collection of technologies that are intended to support groups, including e-mail, NetMeeting, Lotus Notes, etc.

**MUDs and MOOs** Specially built technologies to support real-time communication with others remotely. They originated in the game world, where MUD stands for Multi-User Dungeons and Dragons, and the OO in MOO is object oriented, referring to the underlying programming language type of MUD.

**Picturephone** A commercial product that allowed regular telephone users to see the called party as well as be seen by them.

**real-time** Without delay, usually referring to synchronous communication either conversing with another person or with simultaneous access to work materials.

**workflow** Applications that allow people to coordinate a series of tasks on related documents in a paperless manner, often passing responsibility for reading/ writing/ approval of documents electronically,

## I. DEFINITION

Computer-supported cooperative work (CSCW) is the study of how people work together using computing and communication technologies. The term applies whether the people are collocated or remote, working at the same time or asynchronously, or trying to make a transition from one situation to the other (getting a meeting scheduled, assigning work tasks to people to do in parallel). The collection of technologies available for this is called Groupware. The name CSCW emerged in the mid-1980s as the name of a biannual conference, but has since grown to be the name of the field. CSCW is broadly interdisciplinary, drawing from computer science, management information systems, information science, psychology, sociology, and anthropology.

## II. THINKING ABOUT GROUP WORK: A FRAMEWORK

Many of our waking hours are spent in groups. We live in communities and work in teams in organizations. We use a variety of technologies to do this, many of them old. We hold regular meetings face-to-face, we write on whiteboards, we pass out copies of things to discuss or consult this evening's program. Recently, however, the pace of invention and use of the technologies to help us work in groups has accelerated, first with personal computers (projected Powerpoint slides, later projected notetaking on line) and then with simple networked computing (attachments on e-mail, Lotus Notes shared databases) and the internet itself (instant messaging, discussion groups, and shared repositories of news and information). These technologies have fundamentally changed how we collaborate. Today people can plan a successful global conference without ever having met; large corporations form worldwide teams, called "virtual collocation," and some mobile consultants take their work on the road, never requiring the use of an office.

So are these good technologies? Do they empower us or do they alienate us from each other? What do we learn from the use of these technologies about human behavior itself? The answer, of course, is very complex. Not all groups are the same, they do not do the same kind of activities, and the technologies include a huge variety. It helps to begin with a framework—a way to talk about the various findings and sort things out. Figure 1 presents such a framework, showing key elements in both the players and settings of work, its process and outcomes.

## A. The Group

The same technology can have a very different effect on groups that are made up of different kinds of people, relationships, organizations, and contexts of time and location. Heterogeneous groups behave differently than homogenous ones. How these differences play out is affected by technologies. For example, strangers who are speaking their non-native language benefit from video conferencing much more than established groups speaking their mother tongue. Similarly, groups that have established trust can function cooperatively over simple email, whereas those who do not trust each other quickly dissipate into self-serving behaviors. Furthermore, technologies that fit with a group's reward structure are more likely to be adopted than those that require cultural change.

## B. The Task

If we look at the microstructure of group interactions, they are made up of a host of building-block activities. They consist of the exchange of information, planning, gathering or generating information, discussing to come to agreement, and planning and producing a product. Each of these subtasks is likely to be supported best by different technical support. Furthermore, there are different dependencies among group members in getting the task done; some tasks can be done by a divide-and-conquer strategy, others require close coordination. And, tasks differ in their difficulty, requiring coordinated expertise (e.g., in designing a bridge) or merely joining similar low-level activities (e.g., laying bricks for a building in parallel). And sometimes the group activity is not work related, but intended to build relationships, as in social gatherings and support sessions.

## C. The Technologies

Technologies can be simply categorized in two ways. We can look at the situation that the technology is intended to support (see Groupware), and we can look at what it

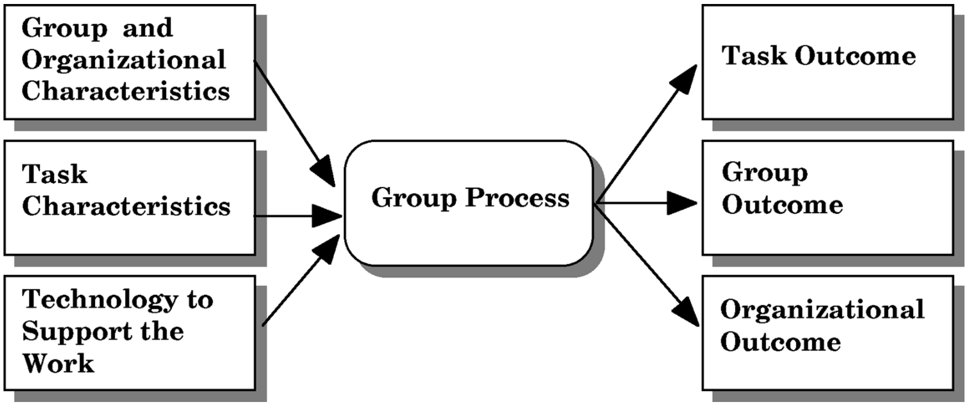


**Figure 1** A simplified relationship between technology and group work.

supports: the peoples' conversations or a shared object. First, technologies focused on several settings: support for real-time collocated work (like a meeting), for remote real-time work (like a video conference), for asynchronous work in the same place (like hospital or factory floor shift handoffs), and for asynchronous work regardless of location (like around the world software development). In addition to these distinctions of delay and location, we can also look at whether the technologies support conversation (e.g., the telephone) or the sharing of objects (e.g., a project plan, engineering documents, the next version of code). Table I organizes some of today's groupware on these dimensions.

## D. The Process

The composition of the group, the task they set out to do, and the technologies they employ determine the process they engage in. One could describe the content of these conversations, the gestures used, the timing, and participation.

## E. Outcomes

The success of a new technology's introduction can be measured in a variety of ways. The quality of the

**Table I** **Groupware to Support Various Time/Places of Groupwork and Focusing on Support of the Conversation or the Shared Objects**

| Work situation | Example products |
|---|---|
| Real time | |
| Support for face-to face conversations | Vantana, Group Systems V |
| Support for remote conversation | |
| Chat boxes, MUDs, and MOOs | Unix Talk, Lambda MOO |
| Videoconferencing | PictureTel, Vtel |
| Desktop audio | Placeware, NetMeeting |
| Desktop video | Proshare, CU-SeeMe, NetMeeting |
| Support for shared work objects | |
| Camera | PictureTel Object Camera |
| Computer whiteboards | SmartBoard, SoftBoard, NetMeeting |
| Shared editors | Aspects, ShrEdit |
| Application sharing | ProShare, NetMeeting, Timbuktu, Point-to-Point, Shared-X. |
| Asynchronous work | |
| Support of conversation | |
| E-mail | Outlook, Mulberry, Eudora, and many more |
| Filters for e-mail | Outlook, Mulberry, and many more |
| Conversation databases | Lotus Notes, Confer, Netnews |
| Revision control systems | RCS |
| Recommender systems | Book recommendations on amazon.com |
| Support of objects | |
| Shared file servers | FTP, Fetch, servers on the network |
| Group authoring | Microsoft Word revision mode, For Comment |
| Project management | MS Project, MacProject |
| Workflow systems | Lotus Notes |
| Transitions between modes of work | |
| Electronic calendars | Meeting Maker, PROFs |
| Awareness servers | ICQ, Instant Messenger, CU-SeeMe |

product, the individual and collective learning, and the group's feelings for each other (affecting their willingness to work together in the future) are all affected by the previous factors. What's important here is to assess not only the immediate outcomes (did they get the work done, were they satisfied), but also those that wouldn't appear until much later (like the growth of expertise in the community, the feelings of isolation, or mistrust).

## III. FINDINGS ABOUT HOW GROUP WORK CHANGES WITH TECHNOLOGIES

In the following, we outline some of the results from the past 20 years of research about the variety of situations in which groups find themselves, working on different tasks and with different technological support. Where appropriate we will refer to the details of the process they undergo to get their tasks done and the consequent outcomes, both related to the task and their group. The sections are ordered to fit the time/place distinctions set out in Table I.

### A. Support for Face-to-Face Conversations

Face-to-face meetings are supported by a variety of technologies. Embedded in many technologies are procedures for structuring the discussions. For example, some embody the "nominal group technique" or the rules of "brainstorming." Others support various kinds of voting mechanisms, such as anonymous ranking algorithms. This set of technologies typically supports the conversation of the meeting. Other technologies used in face-to-face meetings are more free form, allowing people to collaboratively create and change an object under discussion, such as a presentation, proposal, or engineering diagram. In the latter case, the conversation takes place with people's voices as in a normal meeting, but technology supports the object. We will discuss these in Section III.C.

One form of technology support that structures the meetings is called Group Decision Support Systems (GDSS). Earlier work on individual decision support systems (DSSs) grew to involve groups doing similar work. The GDSSs are typically employed by large heterogeneous groups engaged in the tasks of generating a number of ideas (brainstorming) and deciding among them. Because the processes dictated are strict and the technology nonintuitive, these systems typically require a facilitator. Experimental evaluations of GDSSs, reviewed by Fjermestad and others in 1998, have shown that the *process* is indeed more structured, and the outcomes judged to be better. Because these systems typically allow anonymity of the ideas generated, they produce more equal participation among group members. Although decision quality is higher, they require more time and the participants are less satisfied than they are with traditional meetings.

### B. Support for Remote Real-Time Conversation

The telephone is the most popular technology to support people conversing at a distance. Audio conferences using speakerphones support the groups larger than two. More recently, however, people are using a less-rich, less expensive medium, conversing using text-based chats, in either MUDs or MOOs, or via Instant Messaging capabilities. Or, when wishing to get closer to "being there," they opt for video conferencing. Each of these is reviewed in turn below.

#### 1. Text-Based Conversation

In MUDs, MOOs, and instant messaging systems, participants type their contributions in a small window on the computer screen. As soon as they finish, they send the burst of text to all others currently participating, or to a subset they designate, with the utterance tagged with their names. The MUDs and MOOs, in addition, allow people to describe other actions as well, e.g., "Gary enters the room;" "Judy smiles." Participants in MOOs and MUDs also create descriptions of places and objects, sometimes giving the objects actions that are triggered when others act on them, e.g., "powers" that emerge when an object is "picked up." Although these were invented to support fantasy games, today they are used both for education and productive meetings as well as game playing.

Descriptions of the activity in MOOs and MUDs center around the issue of identity. Since participants can describe themselves as they wish, including gender, they often experience reactions to their described selves different from their true selves. For example, those that describe themselves in terms of lower power status are reacted to in a more friendly way than those who describe themselves as powerful. Different MOO communities develop distinctly different cultures depending on their own developed rules of etiquette.

Chat systems have been used for serious purposes, such as the one in the Upper Atmospheric Research Collaboratory (UARC). Here, scientists chat while they collectively view shared displays of data from instruments around the world. As many as 15 scientists have been on at once, discussing the phenomena, with another 30 signed on but not directly conversing, called "lurkers." When the content of the chat dialog is categorized, it turns out that nearly eight separate threads of conversations are carried on in parallel, many more than in a face-to-face meeting, with no more confusion about what is being talked about. Because the participants can scroll back through the previous dialog, they can keep track of the threads and keep up.

### 2. Audio- and Videoconferencing

It has been known for 30 years now that audio conversation helps comprehension much more than reading text, but that adding video adds nothing further. Yet, we continue to develop video connections, first with Picturephone, large-scale videoconferencing suites, and later with video on the desktop, as in NetMeeting. More recent research shows that people consistently *like* the video connection better than audio only. Perhaps this is due to the ease with which they can tell things about the context—who is speaking, how others are reacting to the things being said, and what else is going on in the room.

More recent studies of the value of video show a more complicated picture. Small established groups doing a design task produces better output quality when they are face-to-face than when they communicate by audio only, and that with video the quality is as good as face-to-face. But, a detailed analysis of the process shows that both remote groups grapple with organizing themselves more than the face-to-face groups do. There is more overhead to doing the work. In a second study, examining groups that did not know each other well, video provided a large, significant advantage in both the output quality as well as their satisfaction with the work. In this case, video was important in the strangers' ability to gesture when they had to convey a difficult concept, and to read and adjust to the expressions of understanding or confusion on the other person's face.

These effects make sense when we examine all the things in communication that are conveyed visually. They determine whose turn it is to talk next and what people are referring to (especially when they say shorthand words like "this" and "that"). They can gauge the level of understanding or agreement of the listeners and whether people are paying attention. When the conversation is only supported by audio, these cues are lacking, leading to confusion and disruption.

Modern video communication technology is imperfect, however, disrupting conversation and comprehension. Because of bandwidth limits, most video systems today delay the signal by about 1 second, and even delay the audio in order to keep it synchronized with the video. Unfortunately, this delay has a well-known effect on conversation. Since people pause about 1 second to allow someone else to take a turn, if they do not hear anything in 1 second (now caused by the technical delay, not because someone did not take the turn), they will continue to speak. This causes all kinds of disruption as then two people are speaking at the same time, and more difficult yet, have to explicitly negotiate who will proceed. When given a choice, experienced people will ask for a good audio system without delay (which is possible over regular telephone lines) and dismiss the potentially odd look of the out-of-synch video.

Video is also used for remote presentations, not just for small group meetings. In some distance-learning technologies, the presenter broadcasts by video and audio, gesturing with a telepointer or annotating the presentation with digital ink. In some of these systems, like in Placeware, the audience can send in chat-like questions to the presenter or vote on questions asked by the presenter, and even chat among themselves without disrupting the speaker. Audiences using these systems like them because they can multi-task (e.g., read e-mail while listening to the dull parts); speakers like them less because they cannot react to the normal visual cues about whether the audience understands the material or not.

## C. Support for Face-to-Face or Remote Real-Time Sharing of Objects

The kinds of systems built for sharing the object of the meeting support both face-to-face and remote meetings. When we talk about the object of the meeting, we mean things like the proposal under discussion, the agenda and/or minutes, an engineering drawing or a to-do list. In using these systems, it is assumed that the conversation proceeds in parallel, usually by voice in a meeting room, or through audio conferencing when participants are remote. There are two classes of system that support the sharing of the object: one that allows deep shared editing of the material (where several people can edit at once) and one that shares the entire screen or window and allows only one person to edit the material at a time. Although to date there is only one commercial product that allows simultaneous in-document editing (Aspects), such systems have existed for 10 years in the research world. Systems in this class include Cognoter in the Colab at Xerox PARC and ShrEdit from Michigan. Application sharing through the use of ProShare and NetMeeting, and screen-sharing offered by Shared-X, Timbuktu, or Point-to-Point are simple but powerful collaborative tools. Like flip charts and whiteboards, these tools do not dictate the group process but rather provide editable, visible support for whatever the group dictates is useful at the time.

Research shows that these systems improve small, established groups doing design tasks in the quality of their work. Groups liked it slightly less than working in a traditional room, with only paper and whiteboards. But, interestingly, the process by which they worked differed as well—those using the shared object explored fewer alternative designs, while achieving higher quality. Perhaps the system kept them focused on the task at hand, helping them evaluate as they went whether or not they were making progress. They wasted less time summarizing the current state of the design because it grew in front of them, and participants could work in parallel while seeing what the others were doing.

In a very different setting, another group editor, called Aspects, was used by groups of four 6th grade students engaged in the task of writing articles for their class magazine over the course of 12 weeks. Here, students fought over ownership and control and worried about whether they had permission to change something that another had created. In this case, they could not see exactly what the other person was doing, and so were confused about whether what they did would fit. However, the students reported liking the experience, in particular because it allowed everyone to participate in a joint creation.

Most of the early shared object technologies were embodied in normal desktop computers, with one person per computer. More recently, such technologies are housed in electronic whiteboards, like the LiveBoard, SmartBoard, or SoftBoard. The LiveBoard, a large electronic whiteboard with pen/gesture input, was evaluated in a long-term case by a group doing the task of regular patent reviews for a company. The fact that the group persisted in using it effectively for over two years attests to its value. Comments and suggestions along the way were incorporated into evolutionary design changes, including the system recognizing the structure of a “list” and “outline,” and separating various regions of the work on the board. The board had enough intelligence to “do what I want it to” for the kinds of objects the group was using frequently in their work.

Ordinary video connectivity is also used to share work objects. Engineers share the results of a manufacturing defect remotely by putting the damaged part on the “object camera” in a videoconference, and zooming in on the defect. Discussion is much more effective when the part can been seen than when it is merely described. In another situation, video cameras were used in surgery, focused on the detailed work but projected to the team members in the operating room. By seeing the progress of the surgery, supporting team members did not need verbal instructions about what to do; they could see when the next process was about to commence and prepared for it.

## D. Asynchronous Support of Conversation

### 1. E-Mail

E-mail is the one “killer app” of CSCW. The fact that messages can be exchanged across platforms and networks and that there is a standard, has made it almost as easy and ubiquitous as using a telephone. And, with the multipurpose Internet mail extensions (MIME) standard for attachments and common representation formats like Postscript, it has become easy to transfer full documents and other media to widespread communities.

But with the spread of email have come some powerful effects on human behavior. It has changed who talks to whom, giving power to some people who, because of more social reasons, were not heard from before. They are no longer inhibited from participating because of shyness or articulateness; they can speak without seeing other people. Unfortunately, this invisibility, the fact that the sender is not seeing the reaction of the recipient while typing, has created the negative phenomenon of “flaming.” In the absence of social cues, people tend to write asocial emotive messages that are either shocking, upsetting or offensive to the reader.

E-mail also has profound effects on the conduct of work, as well, depending on how people use it. Not only are messages sent and received independent of time and place, but people also use their “inbox” as a reminder for things they have to do. Time management is supported poorly this way, but it speaks to the context in which email is used and points to future needs.

E-mail comes from a variety of sources with a variety of purposes. Much of today’s e-mail is generated by individuals broadcasting notifications to a list. This can appear in the context of news servers or merely from people in the department notifying everyone of an upcoming event. Other e-mails are targeted directly to the individual recipient and imply an action to be taken by that person. The flood of e-mail is often overwhelming. To help the person cope, some e-mail systems offer ways to sort the e-mail manually by having the recipient shunt it to a folder for future use, or offer programmable filters that automatically sort things into folders or order them on implied priority. In other systems, like Lotus Notes, messages are not displayed in order of arrival, but organized around various topics with responses. This helps the reader keep track of where they are in a particular conversation or issue and give context to their reply.

The fact that these conversations in general are asynchronous and carry few social cues has far-reaching consequences. In particular, after long-term use of e-mail without any face-to-face contact, people begin to distrust each other. Recent studies have measured this distrust and then explored various conditions that may help. Groups who were remote but able to meet face-to-face periodically developed and maintained trust. People who met socially before working remotely developed and maintained trust. Meetings over video were almost as effective as face-to-face, those with audio a bit less. Interestingly, a "pre-meeting" over text-based chat was effective in engendering initial trust *if* the participants talked of social things, the topics they would cover if "getting acquainted."

### 2. Recommender Systems

Recently, the pooling of people's opinions about various things like books or movies has been enabled through technology called recommender systems. This is a form of very asynchronous and anonymous "conversation." By matching a single person's preferences and purchases with others like him or her, the system can suggest new things that the person might enjoy. The interesting part of this kind of system is that the entry of data requires no effort on the part of the recommender; the people who "recommend" are not consciously doing so; they are merely behaving normally and the system combs the information for use.

## E. Support for Asynchronous Sharing of Objects

A variety of objects are shared in the conduct of work. Documents such as work plans, proposals, requirements, etc., are often authored by many over time. People store finished documents like the quarter-end financial statement for others to access. Some people are experimenting with storing what are called design rationales to help people who come later on a project to understand earlier thinking. And project management systems and workflow systems support the coordination of various stages of work.

### 1. Collaborative Authoring of Documents

Much of group work currently consists of individuals writing documents (e.g., system requirements, policy proposals, project reports) and then soliciting comments from many different people and making changes, iterating several times. Today this activity involves a lot of paper drafts and a great deal of time simply entering edits. The standard word processors (e.g., Microsoft Word) now have revision features that make edits visible (crossouts of previous words, new additions marked differently) and allow easy acceptance of the edits and production of a clean copy. There have been extensive prototype systems with ideas about how to support more of this process, reviewed by Michailidis and Rada in 1996.

Authors using these tools intermingle their talk and writing when they develop the ideas for their text, implying a need for informal support at this stage of writing. A study of the use of the PREP editor also supported the need for flexibility in the technology to support the difference phases and preferences of collaborating authors. At some points in the text, authors chose to attach voice commentary, and at others they wanted to explain their ideas by rewriting the text.

### 2. Design Rationale

A number of systems were built to support groups of designers in the complex task of designing an object like an automobile or airplane. Most of these systems aim to capture the argumentation that occurs during the design process, linking the questions of consideration with the alternative solutions that were proposed as well as the evaluative discussion that accompanied it. There is a strong belief that this kind of system would help designers in two ways: (1) it would help designers generate more alternatives and therefore consider more and consider them more fully and (2) it would also be a memory source to help those other team members that have to maintain and/or alter the system. By retrieving the rationale behind an earlier decision, the maintainers might be expected to spend less time rediscovering why some unused alternatives would not work. The most well-known of these systems is gIBIS, which uses a hypertext linking structure to organize the various issues, alternatives, and criteria in the design rationale.

The capture of design rationale has not been totally successful, likely because it is a classic case of misaligned benefits. The person benefiting from the information is not the person who has to invest time and effort by entering it. Also, the representation does not always fit the discussion. Discussions do not always follow one-issue-at-a-time, and alternatives and criteria sometimes relate in braided ways. But, when the diagramming/procedure of design rationale is followed, indeed the designers that come on the project later use it with some success. An analysis of their discussions showed that half their questions are design rationale questions, and half of these are answered by the design rationale documentation. Difficulties arise in that the originator does not always anticipate what later designers might need (and therefore ignores rationales actually discussed) and some design decisions are made without rationale and are accepted without discussion.

## 3. Repositories of Shared Knowledge

Other types of coordination are possible with applications like Lotus Notes. A group can keep open issues lists in a form accessible to all interested parties, and construct workflow systems that automatically route information to the right people for additions and approvals. Some organizations are viewing Lotus Notes as repositories of corporate knowledge, capturing people's experience on previous projects, their heuristics for decisionmaking (e.g., pricing policies and exception handling), boilerplate for various kinds of proposals, etc.

Three case studies have shown the organizational consequences of introducing these kinds of technologies. In the first, consultants were asked to share their knowledge about various clients and engagements in a large Lotus Notes database so that others could benefit from their experience and insights. Two key issues prevented successful adoption. First, although consultants had to bill all their working hours to various clients, there was no account for them to use to bill the time devoted to data entry and learning of Lotus Notes.

Second, consultants were promoted on the basis of their skill advantage over their co-workers, discouraging them from sharing their knowledge. So in this case the accounting/billing of time and the assessment of credit was misaligned with the capability the technology afforded, the objective of its introduction, and the goals of its use. In a successful case in our experience, sales people shared their client contacts with each other. This sharing prevented the embarrassing occasions when a single client was being told different stories by two different sales people. This use fit the incentive scheme in which sales people received commissions on total sales as well as their individual sales.

In the third case study, software designers used Lotus Notes to keep their open issues list and to share information about future features or potential solutions to bugs. Their use of the system initially rose and then declined over 12 months. Interviews of the group members revealed that the team members were less and less inclined to use the application because they thought the manager was not participating. They saw no activity on his part and assumed he did not value their use of the system. In truth, the manager was participating regularly; he read the material but did not write. Unfortunately, Lotus Notes does not make reading activity visible in the interface.

## 4. Workflow Applications

Workflow applications allow people to design, execute, and manage coordinated activities over a network. A process involving several people, like the reporting, approval, and payment of a travel expenses, would be supported electronically. Initial reporting would be done in an electronic document, transferred to another, signatures obtained for approval, and records kept of not only where a particular document is in the process, but who is responsible for it and when was it received or completed.

Workflow applications have often resulted from business process reengineering efforts, where teams examine a business activity and find ways to make it more efficient. Often, efficiency comes with some technology to either store documents for many people to access (eliminating paper), or some stages of processing being eliminated, automated, or supported. Not only does workflow have a bad reputation among workers because it displaces workers, but it also is often conceived as the ideal work process, both rigid and dictatorial. Many of the systems do not support the flexibility and judgment that often accompanies real coordinated work and therefore fall into disuse.

One other aspect of workflow applications has also generated user resistance. Because applications can track the status of various documents and procedures, management can monitor the work of employees. In many countries, such monitoring is disallowed by powerful worker organizations. In the United States, it is allowed, but often unwelcome. In a team where workflow was a new concept, the team members misunderstood how important it was to be accurate in assigning responsibility to various subtasks (like who will handle a particular bug fix). They did not know the full range of views available to the manager to monitor work. One team member, the one most often assigned the first task in a series, came to realize that there was a potential for managerial monitoring. The management report would look as if he alone were responsible for delays, when in fact the real work was being done by others down the line. This feature made him less and less eager to use the application. In general, managerial monitoring is a feature that is known to disincline people from using Groupware.

## F. Support for the Transition between Asynchronous and Real-Time Work: Awareness Support and Calendars

Group work is a mixture of synchronous and asynchronous activities. People meet to plan the work and assign individuals to do various subtasks. These people coordinate and clarify as they go, and periodically meet to align goals and plan next steps. They move often between individual subtasks and coordination or clarification in real time. The following technologies would support these transitions.

First, project management software captures decisions made in meetings about who is doing what, and what the linkages or dependencies are between subtasks. These technologies help calculate the consequences of changes to the plan (by calculating the critical path) and indicate to team members who is waiting for work. Open issues lists and project management software are the tools to support the transition from real-time meetings to parallel, more independent work. These technologies suffer only from the time and effort involved in keeping them up to date. Like writing and distributing meeting minutes, it requires someone to do it.

More difficult is the transition from asynchronous work to synchronous, both accessing an individual so that one can converse or negotiate in real time and calling meetings. Recognizing how difficult this is, some organizations expect workers to be at their desks at all times (and thus reachable at all times). Others schedule standing meetings, expecting full attendance whether one's expertise is needed or not. In lieu of these rash solutions, some have adopted some technologies to help people locate others or to assess when they can reach them so they can make contact.

Two of the most comprehensive of these systems are Montage and Cruiser, which allow video "glances" into team members' offices so one can assess whether they are available for a phone or video conversation. If the glance instead reveals that the intended person is not there or not available, the seeker has several options. The seeker can leave an e-mail message, or can view the person's calendar to see when he/she might return or where they might be reached. Or the seeker can leave a "sticky note" on the screen of the person being sought, attracting the team member's attention immediately upon their return. In an evaluation of Montage which was deployed in a distributed workgroup, the results showed that people glanced at each other nearly 3 times a day, and 3/4 of those were unacknowledged (people were there but they did not respond to connect in a real-time video link). The connections when made were short (a little over a minute). And, although the additional access to calendars, e-mail and sticky notes were used infrequently, people reported afterward valuing them highly.

Other uses of video to allow awareness of team members' activity have been tried and reviewed in 1999 by Mackay. The VideoWindow at Bellcore was intended to encourage both ordinary meeting and casual interactions from remote sites over coffee. RAVE, a suite of systems at Rank Xerox EuroPARC, was intended to support awareness of global activity, glances into individual's offices, and point-to-point contact for close intense work. Long-term use of video connectivity was analyzed in the Portland Experiment. All of these systems have been studied within research lab settings, where modest amounts of sustained use were found. It would be extremely useful to have studies of these systems carried out in other kinds of organizational settings.

All of the implementations of awareness through video raise issues of privacy. Various solutions have been proposed, including introducing reciprocity (you are only on camera when you can see the person viewing you), warning (the sound of a "squeaky door opening" or footsteps coming serves as a signal of an impending glance), viewee control over camera position, and showing recent snapshots as opposed to live immediate action. One awareness system, called Thunderwire, used open audio instead of video. In use, most difficulties pointed to the interpretation of silence. People realized the need for designing new norms in announcing oneself (because the hearers are blind), and negotiating inattention and withdrawal.

On-line calendars afford awareness as well as ease in scheduling meetings. PROFs calendar and Meeting Maker are two popular implementations; both allow designation of who can write and who can read the calendar, as well as control over private portions of the calendar, where viewers can see that the person is busy, but not what they are doing. This application has been declared the quintessential misalignment of costs and benefits (the individual has to keep the calendar up to date if it is going to be of benefit to others), but many organizations have since adopted it successfully. A culture of sharing and accessibility enables the successful adoption of on-line calendars.

### G. Efforts to Support Communities

A number of recent projects investigated the needs of large-scale communities, both through user-centered design and by merely deploying a flexible technology and watching its use. Two main thrusts are relevant here: the development of collaboratories, and the study of home/community use of the world wide web.

A collaboratory is the ". . . combination of technology, tools and infrastructure that allow scientists to work with remote facilities and each other as if they were colocated." A 1993 National Research Council report defines a collaboratory as a ". . . center without walls, in which the nation's researchers can perform their research without regard to geographical location—interacting with colleagues, accessing instrumentation, sharing data and computational resources [and] accessing information in digital libraries." A simplified form of these definitions describes a collaboratory as the use of computing and communication technology to achieve the enhanced access to colleagues and instruments provided by a shared physical location, but in a domain where potential collaborations are not constrained by temporal or geographic barriers.

One such collaboratory effort is the UARC, a set of technologies that allows space scientists studying the upper atmosphere to view real-time data from various instruments (like incoherent scatter radar) around the world. They can align these views with models of what should be going on, and converse through a chat facility with the other scientists or graduate students, and share their configuration views with others to support their ongoing conversation. Preliminary analysis found that UARC theorists and data analysts are working together where they did not before. Also graduate students have access to remote mentors and can experience real-time data collection, where previously they might get to go to a site once in their graduate training. Their network of colleagues has shifted with use of the collaboratory, and it is expected that publication authorship will shift as well. The issue, really, is whether science is progressing faster or not, and since there is no real control group for this effort, we do not know.

An earlier collaboratory supporting molecular biologists studying the *C. elegans* nematode, affectionately called the Worm Community, illustrated both the difficulty of getting a community started and the important emergent attitudes about various participants' willingness to share. Various disciplines develop their own cultures about joint work, credit, and need for immediate communication, and will be variously successful in adopting this and other new capabilities, like digital libraries. There are now efforts underway to support medical radiology and AIDS clinicians and bench scientists in collaboratories. It is likely that collaboratory interactions in science will become a routine aspect of scientific practice, with important implications not just for the practices of scientists but also for the training of graduate students.

In contrast to this well-planned, user-centered development of technologies evident in collaboratories, there are efforts to install various technologies in designated communities. The intent is to learn by various evaluation strategies what people value and what they might need in the future. Two such efforts are the HomeNet in the Pittsburgh area and the Blacksburg Electronic Village in Virginia. These systems have been installed for a few years and data collection has progressed to a point of revealing trends in behavior. They are finding that teenage males are by far the heaviest users, although there is evidence that access to e-mail for keeping up personal conversations and contacts is valued by all.

## IV. KEEPING UP AS DEVELOPMENTS EMERGE

A number of sources exist that review subsets of the technologies and associated group behavior in greater depth. They are well worth pursuing if one wants more detail and deeper analysis of cognition in CSCW. Six large volumes of anthologies of studies in this area are listed in the Bibliography. This article is a shortened version of a paper published in 1999 by Olson and Olson, which has a more extensive bibliography of 150 articles. These volumes plus the *Proceedings on Computer Supported Cooperative Work (CSCW)* and the *European Computer Supported Cooperative Work (ECSCW),* the conferences that alternate meeting biannually, provide both the basics and the continuing progress in this exciting field. Several journals also publish CSCW work: *Computer Supported Cooperative Work, Human Computer Interaction, ACM Transaction on Information Systems, Communication of the ACM, and ACM Transactions on Computer-Human Interaction.*

## SEE ALSO THE FOLLOWING ARTICLES

Electronic Mail • Group Support Systems and Electronic Meeting Systems • Groupware • Human Side of Information • Knowledge Management • Virtual Organizations • Voice Communications

## BIBLIOGRAPHY

Abbott, K. R., and Sarin, S. K. (1994). Experiences with workflow management: Issues for the next generation. *Proceedings of the Conference on Computer Supported Cooperative Work,* pp. 113–120.

Baecker, R. M. (1993). *Readings in Groupware and Computer-Supported Cooperative Work.* San Mateo, CA: Morgan Kaufman.

Finholt, T. A., and Olson, G. M. (1997). From laboratories to collaboratories: A new organizational form for scientific collaboration. *Psychological Science,* pp. 28–36.

Finn, K., Sellen, A., and Wilbur, S., eds. (1997). *Video-Mediated Communication.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Fjermestad, J., and Hiltz, S. R. (1998–1999). An assessment of group support systems experimental research. *Journal of Management Information Systems.*

Greif, I., ed. (1988). *Computer-Supported Cooperative Work: A Book of Readings.* San Mateo, CA: Morgan Kaufmann Publishers.

Mackay, W. (1999). Media spaces: Environments for informal multimedia interaction. *Computer Supported Cooperative Work,* pp. 55–82.

Marca, D., and Bock, G. (1992). *Groupware: Software for Computer Supported Cooperative Work.* Los Alamitos, CA: IEEE Computer Society Press.

McGrath, J. E. (1984). *Groups: Interaction and performance.* Englewood Cliffs, NJ: Prentice Hall.

McLeod, P. L. (1992). An assessment of the experimental literature on electronic group support: Results of a meta-analysis. *Human-Computer Interaction,* 7, 257–280

Michailidis, A., and Rada, R. (1996). A review of collaborative authoring tools. *Groupware and Authoring* (R. Rada, ed.), pp. 9–44. New York: Academic Press.

Olson, G. M., and Olson, J. S. (1999). Computer supported cooperative work. *Handbook of Applied Cognition* (F. Durso, ed.). pp. 409–442. New York: John Wiley & Sons.

Olson, G. M., and Olson, J. S. (2000). Distance matters. *Human Computer Interaction.* 15, 139–179.

Rada, R. (1996). *Groupware and Authoring.* New York: Academic Press.

Sproull, L., and Kiesler, S. (1991). *Connections: New Ways of Working in the Networked Organization.* Cambridge, MA: MIT Press.

# Computer Viruses

**Robert M. Slade**
*Vancouver Institute for Research into User Security*

## GLOSSARY

**activity monitor** A type of antiviral software that checks for signs of suspicious activity, such as attempts to rewrite program files, format disks, and so forth. Some versions of activity monitors will generate an alert for such operations, while others will block the behavior.

**BSI** A boot sector infector; a virus that replaces the original boot sector on a disk, which normally contains executable code.

**change detection** Antiviral software that looks for changes in the computer system. A virus must change something, and it is assumed that program files, disk system areas, and certain areas of memory should not change. This software is very often referred to as *integrity-checking software,* but it does not necessarily protect the integrity of data, nor does it always assess the reasons for a possibly valid change. Change detection using strong encryption is sometimes also known as *authentication software.*

**companion virus** A type of viral program that does not actually attach to another program, but which interposes itself into the chain of command, so that the virus is executed before the "infected" program. Most often, this is done by using a similar name and the rules of program precedence to associate itself with a regular program. Also referred to as a *spawning virus.*

**false negative** A false negative report occurs when an antiviral reports no viral activity or presence, when there is a virus present. References to false negatives are usually only made in technical reports. Most people simply refer to an antiviral "missing" a virus. In general security terms, a false negative is called a false acceptance, or Type II, error.

**false positive** A false positive report occurs when the activity or presence of a virus is reported when there is, in fact, no virus. False positive reports have come to be very widely used among those who know about viral and antiviral programs. Very few use the analogous term *false alarm.* In general security terms, a false positive is known as a false rejection, or Type I, error.

**file infector** A virus that attaches itself to, or associates itself with, a file, usually a program file. File infectors most often append or prepend themselves to regular program files, or overwrite program code. The file infector class is often also used to refer to programs that do not physically attach to files but associate themselves with program file names.

**heuristics** In general, heuristics refer to trial-and-error or seat-of-the-pants thinking rather than formal rules. In antiviral jargon, however, the term has developed a specific meaning with regard to the examination of program code for functions or opcode strings known to be associated with viral activity. In most cases this is similar to activity monitoring but without actually executing the program; in other cases, code is run under some type of emulation. Recently the meaning has expanded to include generic signature scanning meant to catch a group of viruses without making definite identifications.

**macro virus** A macro is a small piece of programming in a simple language, used to perform a simple, repetitive function. Microsoft's Word Basic and VBA macro languages can include macros in data files, and have sufficient functionality to write complete viruses.

**malware** A general term used to refer to all forms of malicious or damaging software, including viral programs, trojans, logic bombs, and the like.

**multipartite** Formerly a viral program that will infect both boot sector/MBRs and files. Possibly now a virus that will infect multiple types of objects or reproduces in multiple ways.

**payload** Used to describe the code in a viral program that is not concerned with reproduction or detection avoidance. The payload is often a message but is sometimes code to corrupt or erase data.

**polymorphism** Techniques that use some system of changing the "form" of the virus on each infection to try and avoid detection by signature scanning software. Less sophisticated systems are referred to as *self-encrypting*.

**scanner** A program that reads the contents of a file looking for code known to exist in specific viral programs.

**stealth** Various technologies used by viral programs to avoid detection on disk. The term properly refers to the technology, not to a particular virus.

**system infector** A virus that redirects system pointers and information in order to "infect" a file without actually changing the infected program file. This is a type of stealth technology.

**trojan horse** A program that either pretends to have, or is described as having, a (beneficial) set of features but which, either instead, or in addition, contains a damaging payload. Most frequently the usage is shortened to *trojan*.

**virus** A final definition has not yet been agreed on by all researchers. A common definition is "a program that modifies other programs to contain a possibly altered version of itself." This definition is generally attributed to Fred Cohen, although Dr. Cohen's actual definition is in mathematical form. Another possible definition is "an entity that uses the resources of the host (system or computer) to reproduce itself and spread, without informed operator action."

**wild, in the** A jargon reference to those viral programs that have been released into, and successfully spread in, the normal computer user community and environment. It is used to distinguish those viral programs that are written and tested in a controlled research environment, without escaping, from those that are uncontrolled "in the wild."

**worm** A self-reproducing program that is distinguished from a virus by copying itself without being attached to a program file, or which spreads over computer networks, particularly via e-mail. A recent refinement is the definition of a worm as spreading without user action, for example, by taking advantage of loopholes and trap doors in software.

**zoo** A jargon reference to a set of viral programs of known characteristics used to test antiviral software.

**A COMPUTER VIRUS** is a program written with functions and intended to copy and disperse itself without the knowledge and cooperation of the owner or user of the computer. A final definition has not yet been agreed on by all researchers. A common definition is "a program that modifies other programs to contain a possibly altered version of itself." This definition is generally attributed to Fred Cohen from his seminal research in the mid-1980s, although Dr. Cohen's actual definition is in mathematical form. Another possible definition is an entity that uses the resources of the host (system or computer) to reproduce itself and spread, without informed operator action.

## I. BASIC CHARACTERISTICS

Dr. Fred Cohen is generally held to have defined the term *computer virus* in his thesis (published in 1984). (The suggestion for the use of the term *virus* is credited to Len Adleman, his seminar advisor.) However, his original definition covers only those sections of code that, when active, attach themselves to other programs. This, however, neglects many of the programs that have been most successful "in the wild." Many researchers still insist on Cohen's definition and use other terms such as *worm* and *bacterium* for those viral programs that do not attack programs. Currently, viruses are generally held to attach themselves to some object, although the object may be a program, disk, document, e-mail message, computer system, or other information entity.

Computer viral programs are not a "natural" occurrence. Viruses are programs written by programmers. They do not just appear through some kind of electronic evolution. Viral programs are written, deliberately, by people. However, the definition of *program* may include many items not normally thought of in terms of programming, such as disk boot sectors and Microsoft Office documents or data files that also contain macro programming.

Many people have the impression that anything that goes wrong with a computer is caused by a virus. From hardware failures to errors in use, everything is blamed on a virus. A virus is not just any damaging condition. Similarly, it is now popularly believed that any program that may do damage to your data or inhibit access to computing resources is a virus. Viral programs are not simply programs that do damage. Indeed, viral programs are not always damaging, at least not in the sense of being deliberately designed to erase data or disrupt operations. Most viral programs seem to have been designed to be a kind of electronic graffiti: intended to make the writer's mark in the world, if not his or her name. In some cases a name is displayed, on occasion an address, phone number, company name, or political party.

## II. HISTORY AND TRENDS

Many claims have been made for the existence of viruses prior to the 1980s, but, so far, these claims have not been accompanied by proof. The Core Wars programming contests did involve self-replicating code, but usually within a structured and artificial environment.

At least two Apple II viruses are known to have been created in the early 1980s. There is some evidence that the first viruses were created during the 1980s, and Fred Cohen's work was undertaken during that decade. However, it was not until the end of the decade (in 1987 in particular) that knowledge of real viruses became widespread, even among security experts. For many years boot sector infectors and file infectors were the only types of common viruses. These programs spread relatively slowly, primarily distributed on floppy disks, and were thus slow to disseminate geographically. However, these viruses tended to be very long lived.

During the early 1990s virus writers started experimenting with various functions intended to defeat detection. (Some forms had seen limited trials earlier.) Among these were polymorphism, designed to change form in order to defeat scanners, and stealth, designed to attempt to confound any type of detection. None of these virus technologies had a significant impact. Most viruses using these "advanced" technologies were easier to detect because of a necessary increase in program size.

Although demonstration programs had been created earlier, the mid-1990s saw the introduction of macro and script viruses in the wild. These were initially confined to word processing files, particularly files associated with the Microsoft Office Suite. However, the inclusion of programming capabilities eventually led to script viruses in many objects that would normally be considered to contain data only, such as Excel spreadsheets, PowerPoint presentation files, and e-mail messages. This fact led to greatly increased demands for computer resources among antiviral systems, since many more objects had to be tested, and Windows object linking and embedding (OLE) format data files presented substantial complexity to scanners. Macro viruses also increase new variant forms very quickly, since the virus carries its own source code, and anyone who obtains a copy can generally modify it and create a new member of the virus family.

E-mail viruses became the major new form of virus in the late 1990s and early 2000s. These viruses may use macro capabilities, scripting, or executable attachments to create e-mail messages or attachments sent out to e-mail addresses harvested from the infected machine. E-mail viruses spread with extreme rapidity, distributing themselves worldwide in a matter of hours. Some versions create so many copies of themselves that corporate and even service provider mail servers are flooded and cease to function. E-mail viruses are very visible, and so tend to be identified within a short space of time, but because many are macros or scripts, many variants can be quickly generated.

With the strong integration of the Microsoft Windows operating system with its Internet Explorer browser, Outlook mailer, Office suite, and system scripting, recent viruses have started to blur the normal distinctions. A document sent as an e-mail file attachment can make a call to a Web site that starts active content, which installs a remote access tool acting as a portal for the client portion of a distributed denial-of-service network.

Because the work has had to deal with detailed analysis of low-level code, virus research has led to significant advances in the field of forensic programming. However, to date computer forensic work has concentrated on file recovery and decryption, so the contributions in this area likely still lie in the future.

Many computer pundits, as well as some security experts, have proposed that computer viruses are a result of the fact that currently popular desktop operating systems have only nominal security provisions. They further suggest that viruses will disappear as security functions are added to operating systems. This thesis ignores the fact, well established by Cohen's research and subsequently confirmed, that viruses use the most basic of computer functions, and that a perfect defense against viruses is impossible. This is not to say that an increase in security measures by operating system vendors could not reduce the risk of viruses; the current danger could be drastically reduced with relatively minor modifications to system functions.

It is going too far to say (as some have) that the very existence of viral programs, and the fact that both viral strains and the numbers of individual infections are growing, means that computers are finished. At the present time, the general public is not well informed about the virus threat, and so more copies of viral programs are being produced than are being destroyed. Indeed, no less an authority than Fred Cohen has championed the idea that viral programs can be used to great effect. An application using a viral form can improve performance in the same way that computer hardware benefits from parallel processors. It is, however, unlikely that viral programs can operate effectively and usefully in the current computer environment without substantial protective measures being built into them. A number of virus and worm programs have been written with the obvious intent of proving that viruses could carry a useful payload, and some have even had a payload that could be said to enhance security. Unfortunately, all such viruses have created serious problems themselves.

## III. RELATED PROGRAMS AND TERMS

Computer viruses have many aspects. There are also a number of classes of malware (maliciously programmed software, or programmed security threats) that do not have viral characteristics.

### A. Specific Virus Types

Viruses are generally partly classified by the objects to which they attach. (Worms, discussed in the next section, may be seen as a type of virus that attaches to nothing.)

#### 1. Boot Sector Infector

Most desktop computer operating systems have some form of boot sector, a specific location on disk that contains programming to bootstrap the start-up of a computer. Boot sector infectors (BSIs) replace or redirect this programming in order to have the virus invoked, usually as the first program running on the computer.

BSIs would not appear to fit the definition of a virus infecting another program, because BSIs can be spread by disks that do not contain any program files. However, the boot sector of a normal MS-DOS disk, whether or not it is a "system" or bootable disk, always contains a program (even if it only states that the disk is not bootable), and so it can be said that a BSI is a "true" virus.

The terminology of BSIs comes from MS-DOS systems, and this leads to some additional confusion. The physical "first sector" on a hard drive is not the operating-system boot sector. On a hard drive the boot sector is the first "logical" sector. The number one position on a hard drive is the master boot record (MBR). Some viral programs, such as the Stoned virus, always attack the physical first sector: the boot sector on floppy disks and the master boot record on hard disks. Thus viral programs that always attack the boot sector might be termed "pure" BSIs, whereas programs like Stoned might be referred to as an "MBR type" of BSI. The term *boot sector infector* is used for all of them though, since all of them infect the boot sector on floppy disks.

#### 2. File Infectors

File infecting viral programs link, or attach, to an existing program in many different ways. The largest number will place the bulk of the viral code toward the end of the program file, with a jump sequence at the beginning of the file that points to the main body of the virus. Some viral code attaches to the beginning of the file—simpler in concept, but actually more difficult in execution. These two techniques are known as *appending* and *prepending*, respectively, but the terms are used less than in years past.

Some viral programs do not attach to the beginning or end of the file, but write their code into the target program itself. Most often this is done by simply overwriting whatever is there already. Of course, if a virus has overwritten existing code, the original target program is damaged, and there is little or no possibility of recovery other than by deleting the infected file and restoring from a clean backup copy. However, some overwriting viruses are known to look for strings of null characters. If such can be identified, the viral code can be removed and replaced with nulls again.

#### 3. System or Companion Viruses

Some viral programs do not physically touch the target file at all. There are two ways to infect in this manner. One method is quite simple, and may take advantage of precedence in the system. In MS-DOS, for example, when a command is given, the system checks first for internal commands, then COM, EXE, and

BAT files in that order. EXE files can be "infected" by writing a COM file in the same directory with the same file name. This type of virus is most commonly known as a *companion virus,* although the term *spawning virus* is also used.

The second method is more difficult. System viral programs will not change the target program, but will change the directory entry for the program so as to point to the virus. The original file will not be changed, but when the target program is called, the virus will be run first instead. More recently, other ways to have the system call the virus have been found. The registry, in recent Microsoft Windows versions, has enormous control over the operation of the computer and can be used to modify many operations in order to have viral programs run in place of, or before, many normal functions.

### 4. Macro, Script, or Interpreted Viruses

Early viruses were written as object or machine executable code. There were some experiments involving MS-DOS batch files, or programs that contained macro languages, such as Lotus 1-2-3. Macro, batch, or scripting languages are not directly executable by the computer, but must be interpreted by the operating system or an application program. Macro or script programs are written in a simple source code and are thus comprehensible to an educated user.

With the introduction of more functional macro, script, and batch languages, plus the ability to attach these programs to data objects in such way that they are easily executable, interpreted viruses have become a serious problem. Viral macros can be included with ordinary Microsoft Word document files, and text attachments or e-mail inclusions can contain viruses written in the Microsoft VBScript language. With the prevalence of the Microsoft operating system, Office application, and e-mail software, these types of viruses have become the most common form of viral program seen today.

Interpreted viruses have another feature that increases their numbers. Each macro or script virus carries its own source code, so virus writers are easily able to use existing viruses as templates to create others.

### 5. E-mail Viruses

Most high-profile viruses in recent years have involved the use of e-mail systems. Technically speaking, e-mail viruses are no different than other viruses or worms, aside from the obvious distinction of the specific use of e-mail or network systems, and may be either object code programs, script attachments, or Microsoft Office document attachments with macro viruses.

The major characteristic of an e-mail virus is the more extensive use of social engineering to try to get the user to activate the virus. This includes vague but attractive subject lines and message text, and the use of address book data to generate the appearance that the message is from someone the user knows and trusts.

The infection pattern of e-mail viruses is also significantly different from that for more traditional forms. Older viruses took months to spread, but stayed in the computing environment for years. E-mail viruses and worms often spread worldwide within a matter of hours, but, because of the attendant publicity, seldom are a problem for more than a few days.

### 6. Multipartite Viruses

Multipartite viral programs were originally also known as dual infection, since they had the potential to infect both program files and boot sectors. This expands the range of possible vectors. Multipartite infections can theoretically travel on any disk, and multiple copies may travel on a disk if program files are present. Dual infectors can also travel on networks, which pure BSIs cannot, and via files passed over bulletin board systems and other communications channels.

Multipartite viruses have traditionally been seen as a combination of the two earliest types, file and boot sector infectors, hence the alias of dual infection. However, the term may now be more generally used for any virus that can infect multiple objects or infect in multiple ways.

## B. Worms

Worm programs also reproduce and are seen by many as simply a special case of computer virus. The distinction is said to be either that worms are viruses that travel across networks, or that worms spread by themselves, without attaching to an infected object.

The derivation of the term *worm* is given by the experiments in distributed computing by John Shoch and Jon Hupp. They wrote programs that would transfer copies of themselves to other machines on a network while remaining under the control of the original program. They saw the entire matrix of copied programs as a single "worm": a single entity with many program segments. There are also references to the network "tapeworm" in the fictional work *Shockwave*

*Rider* by John Brunner, although the program described in Brunner's novel neither reproduced nor had segments.

Two examples of the usage of the term *worm* are the famous Morris/Internet/UNIX worm of late 1988, and the lesser known CHRISTMA EXEC mail worm of December 1987. Many recent e-mail viruses are also seen as examples of worms.

## C. Hoaxes

Hoax warnings are specifically related to viruses. These are false warnings about nonexistent computer viruses, generally spread as chain letters. Characteristics of hoaxes are that they give almost no technical details of the supposed virus, warn of terrible damage that the virus will do, and state that the virus is impossible to detect or eradicate. The final point is that hoaxes ask the reader to forward the message to all friends and contacts. The chain letter aspect of hoaxes is another association with viruses: the hoax uses the user to reproduce, rather than copying itself with computer functions.

## D. Trojans

Trojans, or trojan horse programs, are the largest class of malware. However, the term is subject to much confusion, particularly in relation to computer viruses. A *trojan* is a program that pretends to do one thing while performing another, unwanted action. The extent of the "pretense" may vary greatly. Many of the early PC trojans relied merely on the file name and a description on a bulletin board. "Log-in" trojans, popular among university student mainframe users, mimicked the screen display and the prompts of the normal log-in program and could, in fact, pass the username and password along to the valid log-in program at the same time as they stole the user data. Some trojans may contain actual code that does what it is supposed to be doing while performing additional nasty acts that it does not tell you about.

An additional confusion with viruses involves trojan horse programs that may be spread by e-mail. In years past, a trojan program had to be posted on an electronic bulletin board system or a file archive site. Because of the static posting, a malicious program would soon be identified and eliminated. More recently, trojan programs have been distributed by mass e-mail campaigns, by posting on Usenet newsgroup discussion groups, or through automated distribution agents (bots) on Internet relay chat (IRC) channels. Since source identification in these communications channels can be easily hidden, trojan programs can be redistributed in a number of disguises, and specific identification of a malicious program has become much more difficult.

Some data security writers consider that a virus is simply a specific example of the class of trojan horse programs. There is some validity to this usage since a virus is an unknown quantity that is hidden and transmitted along with a legitimate disk or program, and any program can be turned into a trojan by infecting it with a virus. However, the term *virus* more properly refers to the added, infectious code rather than the virus/target combination. Therefore, the term *trojan* refers to a deliberately misleading or modified program that does not reproduce itself.

## E. Remote Access Tools

All networking software can, in a sense be considered remote access tools (RATs): We have file transfer sites and clients, World Wide Web servers and browsers, and terminal emulation software that allows a microcomputer user to log on to a distant computer and use it as if he or she were on site. The RATs considered to be in the malware camp tend to fall somewhere in the middle of the spectrum. Once a client, such as Back Orifice or SubSe7en, is installed on the target computer, the controlling computer is able to obtain information about the target computer. The master computer will be able to download files from, and upload files to, the target. The control computer will also be able to submit commands to the victim, which basically allows the distant operator to do pretty much anything to the prey. One other function is quite important: All of this activity goes on without any alert being given to the owner or operator of the targeted computer.

When a RAT program has been run on a computer, it will install itself in such a way as to be active every time the computer is turned on after that. Information is sent back to the controlling computer noting that the system is active. The user of the command computer is now able to explore the target, escalate access to other resources, and install other software, such as DDoS zombies, if so desired.

Once more, note that remote access tools are not viral. When the software is active, though, the master computer can submit commands to have the installation program sent on, via network transfer or e-mail, to other machines.

## F. DDoS Agents

Distributed denial-of-service (DDoS) is a modified denial-of-service (DoS) attack. DoS attacks do not attempt to destroy or corrupt data, but attempt to use up a computing resource to the point where normal work cannot proceed. The structure of a DDoS attack requires a master computer to control the attack, a target of the attack, and a number of computers in the middle that the master computer uses to generate the attack. These computers between the master and the target are variously called agents or clients, but are usually referred to as running "zombie" programs.

Again, note that DDoS programs are not viral, but checking for zombie software protects not only you and your system, but prevents attacks on others as well. It is, however, still in your best interest to ensure that no zombie programs are active on any of your machines. If your computers are used to launch an assault on some other system, you could be liable for damages.

## G. Jokes or Pranks

Pranks are very much a part of the computer culture. So much so that you can now buy commercially produced joke packages that allow you to perform "Stupid Mac (or PC, or Windows) Tricks." Numberless pranks are available as shareware. Some make the computer appear to insult the user; some use sound effects or voices; some use special visual effects. A fairly common thread running through most pranks is that the computer is, in some way, nonfunctional. Many pretend to have detected some kind of fault in the computer (and some pretend to rectify such faults, of course making things worse). One entry in the virus field is PARASCAN, the paranoid scanner. It pretends to find large numbers of infected files, although it does not actually check for any infections.

Generally speaking, pranks that create some kind of announcement are not viral, and viruses that generate a screen or audio display are rare. The distinction between jokes and trojans is harder to make, but pranks are intended for amusement. Joke programs may, of course, result in a denial of service if people find the prank message frightening.

# IV. TRIPARTITE VIRUS STRUCTURE

Malicious software has six basic elements, although not all may be present in each specific program. Insertion is the method used to become resident in the target system. Avoidance, otherwise known as stealth, is the method used to evade detection. Eradication is the means by which the malware removes traces of itself following a trigger. Propagation or replication is considered the province of viruses and worms only and is what makes a program a virus. The trigger is the event or condition that initiates a payload. The payload is the additional coding carried by the program. In considering computer viruses, three structural parts are considered important, the replication or infection mechanism, the trigger, and the payload.

## A. Infection Mechanism

The first, and only necessary, part of the structure is the infection mechanism. This is the code that allows the virus to reproduce, and thus to be a virus. The infection mechanism itself has a number of parts to it.

The first function is to search for, or detect, an appropriate object to infect. The search may be active, as in the case of some file infectors that take directory listings in order to find appropriate programs, of appropriate sizes, or it may be passive, in the case of macro viruses that infect every document as it is saved. Some additional decisions may be made once an object is found. Some viruses may try to actually slow the rate of infection in order to avoid detection. Most will check to see if the object has already been infected.

The next action will be the infection itself. This may entail the writing of a new section of code to the boot sector, the addition of code to a program file, the addition of macro code to the Microsoft Word NORMAL.DOT file, the sending of a file attachment to harvested e-mail addresses, or a number of other operations. Additional subfunctions exist at this step as well, such as the movement of the original boot sector to a new location, or the addition of jump codes in an infected program file to point to the virus code. There may also be changes to system files, to try and ensure that the virus will be run every time the computer is turned on. This can be considered the insertion portion of the virus.

At the time of infection, a number of steps may be taken to try to keep the virus safe from detection. The original file creation date may be conserved and used to reset the directory listing in order to avoid a change in date. The virus may have its form changed, in some kind of polymorphism. The active portion of the virus may take charge of certain system interrupts, in order to make false reports when someone tries to look for a change to the system. Certain prompts or alerts may also be generated in an attempt to make any odd behavior noticed by the user appear to be part of a normal, or at least innocent, computer error.

## B. Trigger

The second major component of a virus is the payload trigger. The virus may look for a certain number of infections, a certain date and/or time, or a certain piece of text, or it may simply blow up the first time it is used. As noted, a virus does not actually have to have either a trigger or a payload.

## C. Payload

If a virus does have a trigger, then it usually does have a payload. The payload can be pretty much anything, from a simple one-time message, to a complicated display, to a program that reformats the hard disk. However, the bigger the payload, the more likely it is that the virus will get noticed. Therefore, while you may have seen lists of payload symptoms to watch for—such as text messages, ambulances running across the screen, letters falling down, and the like—checking for these payloads is not a very good way to keep free of viruses. The successful ones keep quiet. However, a virus carrying a very destructive payload will also eradicate itself when it wipes out its target.

# V. DAMAGE

There is much discussion of "damaging payloads" in viruses, and the topic should be addressed more carefully. Viruses can do any kind of damage that software can do. This includes corrupting data, erasing files, corrupting system data, reformatting disks, corrupting security systems, corrupting software, or killing program processes.

The myth of viral programs physically damaging hardware seems to be one of the more enduring. No viral program yet found has been designed to damage hardware, and there has never been any confirmed case of a viral program directly causing physical damage to computer hardware. There is at least one virus that does attempt to correct the BIOS (basic input/output system) programming if the computer uses a Flash memory EEPROM. This does prevent the computer from starting correctly, and does require the physical removal and reprogramming or replacement of the BIOS chip. However, no hardware is actually damaged.

However, deliberate damage is not the only kind that viruses can cause. Viruses tend to be poorly programmed and tested, and they carry many bugs and sloppy programming practices. A great many viruses have caused data corruption that was probably never intended, simply because the programmer did not know enough about disk structures or memory management.

Viruses can also cause various kinds of denial of service, because they take up CPU time, disk space, space in mail queues, and other resources.

# VI. ANTIVIRAL TECHNOLOGIES

All antiviral technologies are based on the three classes outlined by Fred Cohen in his early research. The first type performs an ongoing assessment of the functions taking place in the computer, looking for operations known to be dangerous. The second checks regularly for changes in the computer system where changes should occur only infrequently. The third examines files for known code found in previous viruses.

Within these three basic types of antiviral software, implementation details vary greatly. Some systems are meant only for use on stand-alone systems, while others provide support for centralized operation on a network. With Internet connections being so important now, many packages can be run in conjunction with content scanning gateways or firewalls.

## A. Activity Monitors

An activity monitor watches for suspicious activity. It may, for example, check for any calls to format a disk or attempts to alter or delete a program file while a program other than the operating system is in control. It may be more sophisticated, and check for any program that performs "direct" activities with hardware, without using the standard system calls.

Activity monitors represent some of the oldest examples of antiviral software. Generally speaking, such programs followed in the footsteps of the earlier antitrojan software, such as BOMBSQAD and WORMCHEK in the MS-DOS arena, which used the same "check what the program tries to do" approach. This tactic can be amazingly effective, particularly given the fact that so much malware is slavishly derivative and tends to use the same functions over and over again.

It is, however, very hard to tell the difference between a word processor updating a file and a virus infecting a file. Activity monitoring programs may be more trouble than they are worth because they can continually ask for confirmation of valid activities. The annals of computer virus research are littered with suggestions for virus-proof computers and systems that basically all boil down to the same thing: If you restrict the operations that a computer can perform, you can eliminate viral programs. Unfortunately, you also can eliminate most of the usefulness of the computer.

## B. Change Detection Software

Change detection software, also often referred to as integrity- checking software, examines system and/or program files and configurations, stores the information, and compares it against the actual configuration at a later time. Most of these programs perform a checksum or cyclic redundancy check (CRC) that will detect changes to a file even if the length is unchanged. Some programs will even use sophisticated encryption techniques to generate a signature that is, if not absolutely immune to malicious attack, prohibitively expensive, in processing terms, from the point of view of a virus.

A sufficiently advanced change-detection system, which takes all factors including system areas of the disk and the computer memory into account, has the best chance of detecting all current and future viral strains. However, change detection also has the highest probability of false alarms, since it will not know whether a change is viral or valid. The addition of intelligent analysis of the changes detected may assist with this failing.

## C. Scanners

Scanners examine files, boot sectors, and/or memory for evidence of viral infection. They generally look for viral signatures, sections of program code that are known to be in specific viral programs but not in most other programs. Because of this, scanning software will generally detect only known viruses and must be updated regularly. Some scanning software has resident versions that check each file as it is run.

Scanners have generally been the most popular form of antiviral software, probably because they make a specific identification. In fact, scanners offer somewhat weak protection, since they require regular updating. Scanner identification of a virus may not always be dependable: A number of scanner products have been known to identify viruses based on common families rather than definitive signatures.

## D. Heuristic Scanners

A recent addition to scanners is intelligent analysis of unknown code, currently referred to as *heuristic scanning*. Note that heuristic scanning does not represent a new type of antiviral software. More closely akin to activity monitoring functions than traditional signature scanning, this looks for “suspicious” sections of code that are generally found in viral programs. While it is possible for normal programs to want to “go resident,” look for other program files, or even modify their own code, such activities are telltale signs that can help an informed user come to some decision about the advisability of running or installing a given new and unknown program. Heuristics, however, may generate a lot of false alarms, and may either scare novice users, or give them a false sense of security after “wolf” has been cried too often.

## E. Disinfection Software

Disinfection software is not a type of antiviral software as such, but is generally a feature added to normal signature scanners. Disinfection is by no means the optimal way to deal with viral infections. The best solution is to delete (and, preferably, overwrite) the affected file or area, and restore programs from original sources. BSIs affect a whole disk, and therefore present greater problems, but in most cases material can be recovered from infected disks, and the disks themselves “cleansed” in various ways. There comes a point at which the trade-off between security and convenience tips the scales in favor of disinfection, but be aware of the dangers.

In many cases, disinfection is simply not possible. An overwriting virus, for example, will not keep any track of the material it destroys when it dumps itself into a file. Many viruses contain bugs that prevent the recovery of the original file. Also, sadly, disinfection software has been known to contain bugs that left the situation worse after the attempted cleanup than after the infection.

Generally speaking, disinfecting software will contain a description of the specific viral operation of a given viral program, so that the infection process can be reversed. However, virus removal is no longer the exclusive province of scanning software. Two types of generic disinfection now exist. Some change-detection programs store sufficient information about the file to make an attempt to restore it if the damage is not too severe or complicated. Also, heuristic scanning is being used to trace and remove viral infections. So far testing has revealed serious drawbacks to both of these applications, but the technology is still in its infancy and shows promise for the future.

## VII. ANTIDETECTION MECHANISMS

Viruses are seen as predators in the software jungle, but actually behave more like prey. Most viruses can be killed easily, once detected, and so rely on prolific reproduction or technologies to avoid detection.

### A. Stealth

In a sense all concealment mechanisms used in viruses are "stealth" technologies, but this term is most often used for programs that attempt to avoid detection by trapping calls to read the disk and "lying" to the interrogating program. By so doing, they avoid any kind of detection that relies on perusal of the disk. The disk gives back only that information regarding file dates, sizes, and makeup appropriate to the original situation, providing of course, that the virus is active at the time of checking. Although this stealth method avoids any kind of "disk" detection, including checksumming and signature scanning, it leaves traces in the computer's memory that can be detected.

### B. Hiding Places

Consider the common boot sector. To the vast majority of users, the fact that a program can be located at a physical position on the disk but not be referenced by the file directory list is a foreign concept. This confusion may contribute to the enormous success of boot sector viral programs.

The sector and even the partition boot record on a hard disk are accessible to dedicated amateurs armed with utility software. However, there are other places to hide on a disk that are not as easily examined. It is quite possible to format an additional track outside the normal range. To avoid problems between drives with variations in tolerance, the software does not "push" the limits of the hardware. Special programs for the Apple II computer provided 38 tracks rather than the normal 35. Various programs are available for MS-DOS as well that provide greater storage on the same-sized disks.

In addition to tracks outside of and between normal formats, there is substantial space between the sectors on a disk. Some programs can increase the number of sectors so as to increase the space on disk. However, it is also possible to use the additional space without formatting additional sectors, simply by writing information to the space between. This fact has occasionally been used by commercial software manufacturers for the purposes of copy protection.

### C. Polymorphism

The "shape" of a virus can be changed in a number of ways. One way is to get a simple "random" number, such as the value of the seconds field of the system time when the infection occurs, and to perform a simple encryption on the value of each byte in the viral code. Only a short chunk is left at the beginning to decrypt the rest of the virus when the time comes to activate it. Encryption could be used in other ways: encrypting a regular, but arbitrary, number of bytes, or encrypting most of the code as a whole rather than on a byte basis.

In programming there are always at least half a dozen means to the same end. Many programming functions are commutative—it does not matter in what order certain operations are performed. This means that very small chunks of code, pieces too small to be of use as signatures, can be rearranged in different orders each time the virus infects a new file. This, as you can imagine, requires a more "intelligent" program than a simple encryption routine.

A distinction tends to be made between the early, and limited, self-encrypting viral programs, and the latter, more sophisticated polymorphs. Earlier self-encrypting viral programs had limited numbers of variants: Even the enormous Whale virus had less than 40 distinct forms. (Some of the earliest were the V2Px family written by Mark Washburn. He stated that he wrote them to prove that scanners were unworkable, and then he wrote his own activity-monitoring program. He is one of the very few people to have written and released a virus who also wrote antiviral software. His release of "live" code into the wild tends to deny him status as an antivirus researcher. Lest some say this is arbitrary bias, please note that his thesis was rather ineffectual: All his variants are fairly easily detectable.) More recent polymorphs are more prolific: Tremor is calculated to have almost 6,000,000,000 forms.

## VIII. CONCLUSION

Viruses are here. Virus numbers are growing. New viruses and variants will continue to grow for the foreseeable future and will become an increasing problem in all areas of computing. We are also seeing a very disturbing trend toward convergence in virus writing. Until now, viruses have been a problem all on their own. Viruses are no longer an isolated threat. They are being used to launch attacks from one operating system platform, aimed at another. Invasions of privacy, with attendant social and legal problems, are being carried as payloads in recent viruses. Viral programs are using a variety of technologies to update themselves on the fly and make their presence known to outsiders once they have invaded a system.

However, training and some basic policies can greatly reduce the danger of viruses to users. Here are a few guidelines that can really help in the current environment:

- Do not double-click on attachments.
- When sending attachments, be really specific.
- Do not blindly use Microsoft products as a company standard.
- Disable Windows Script Host. Disable ActiveX. Disable VBScript. Disable JavaScript.
- Do not send HTML-formatted e-mail.
- Use more than one scanner, and scan everything.

Viruses do not really present a new threat. Still it is true that viruses present a greater risk than many other forms of malicious software. A virus need only be created and released once. If it is successful, it spreads on its own, attacking systems as it goes. Each system compromised becomes another source of infection. If you are not part of the solution, in the viral world, you are most definitely part of the problem.

## SEE ALSO THE FOLLOWING ARTICLES

Crime, Use of Computers in • Disaster Recovery Planning • Encryption • Error Detecting and Correcting Codes • Ethical Issues • Firewalls • Internet, Overview • Network Environments, Managing • Privacy • Security Issues and Measures

## BIBLIOGRAPHY

Cohen, F. (1994). *A short course on computer viruses.* 2nd ed. New York: Wiley.

Ferbrache, D. (1992). *A pathology of computer viruses.* London: Springer-Verlag.

Gattiker, U., Harley, D., and Slade, R. (2001). *Viruses revealed.* New York: McGraw-Hill.

Highland, H. J. (1990). *Computer virus handbook.* New York: Elsevier Advanced Technology.

Hruska, J. (1992). *Computer viruses and anti-virus warfare.* 2nd ed. London: Ellis Horwood.

Kane, P. (1994). *PC security and virus protection handbook.* New York: M&T Books.

Lammer, V. (1993). Survivor's guide to computer viruses. *Virus Bulletin.*

Slade, R. M. (1996). *Robert Slade's guide to computer viruses.* 2nd ed. New York: Springer-Verlag.

Solomon, A. (1991). *PC viruses: Detection, analysis and cure.* London: Springer-Verlag.

Solomon, A. (1995). *Dr. Solomon's virus encyclopedia.* Aylesbury, UK: S&S International.

Vibert, R. S. (2000). *The enterprise anti-virus book.* Braeside, Canada: Segura Solutions.

# Continuous Simulation

**Stanislaw Raczynski**
*Universidad Panamericana, Mexico*

## GLOSSARY

**analog computer** A device that contains several (up to several hundreds) integrators, amplifiers, summation circuits, signal sources, and auxiliary circuits that use the operational amplifier as the basic element. The main application of analog computers is to solve ordinary differential equations and to simulate the behavior of continuous dynamic systems.

**bond graphs (BGs)** A bond graph is a net of links (named *bonds,* represented by a "harpoons") and *nodes.* Each bond is related to two variables: *effort* and *flow.* For example, in electrical models the efforts are voltages and the flows are currents. In a mechanical model efforts are forces and flows are velocities. The nodes represent external excitations or internal balances that the model variables must obey. BGs are useful while modeling physical systems.

**computer simulation** The process of establishing relations between models and computers. As a result we get a computer implementation of the model.

**differential inclusions (DIs)** The difference between a differential equation and a differential inclusion is that the right-hand side of the DI is a set-valued function instead of real- or vector-valued one. The general form of a DI is

$$dx/dt = F(x,t)$$

where $t$ is an independent variable, $x$ is the dependent variable, and $F$ is a set.

**distributed parameter system** A system in which we cannot relate discrete elements like electrical resistances, capacitors, masses, dampers, etc., with discrete system components. This occurs, for example in three-dimensional heat transfer problems, fluid dynamics, or diffusion processes. The main tools used in modeling of such kind of systems are the partial differential equations.

**modeling** A process of creating models, which are abstractions of real or imaginary systems. Models are used to predict what happens if certain actions are taken. Many years ago, A. Rosenblueth and N. Wiener pointed out that modeling is one of the central needs of scientific reasoning. Dealing with models we must take into account many factors, like the level of simplification, experimental frame, model validity, tractability, credibility, and the aim of modeling among others. The difference between modeling and simulation is seen in the fact that modeling means to look for relations between real systems and models, and computer simulation refers to relations between models and computers.

***N*-body problem** The problem of integrating the trajectory of $N$ bodies moving in space, due to gravitational or other forces they produce. The difficulty of the numerical solution is due to the fact that the number of reciprocal interactions grows with the square of the number of bodies. Recently, simulation of the movement of several hundreds of thousands of bodies has been accomplished.

**ordinary differential equation (ODE)** An ordinary differential equation in its general form can be expressed as follows.

$$F(x, x^{(1)}, x^{(2)}, ..., x^{(n)}, t) = 0$$

where $t$ is the independent variable representing the model time, $x$ is the dependent variable and $x^{(1)} = dx/dt$, $x^{(2)} = d^2x/dt^2$, etc. The order of the equation is equal to the order of the highest order derivative of the dependent variable.

**partial differential equation (PDE)** A partial differential equation is a mathematical relation between the independent variable, the dependent variable, and one or more partial derivatives of the dependent variable.

**predictor-corrector method** One of the methods belonging to the group of *multistep* methods for ordinary differential equations. It is faster than the Runge–Kutta methods and widely used in continuous system simulation, in particular in the simulation of physical systems.

**Runge–Kutta method** A family of numerical methods for ordinary differential equations. Used in continuous system simulation.

**signal flow graphs (SFGs)** A graphical representation of causal dynamic systems. A SFG is composed of links and nodes. Links are directed, that is, every link has an input and output, marked with an arrow. Nodes represent signals or variables, and links are operators that transform input signals into output signals. The main field of application is automatic control and instrumentation. SFGs are also useful in models of system dynamics.

**stiff equations** Stiffness occurs when there are two or more very different timescales, for example, in systems that have very fast and very slow parts interacting with each other. In the solution to stiff equations some terms may change rapidly compared to other ones. The integration process becomes unstable even if such rapid terms are negligible. This problem is called *stiffness* and provokes serious difficulties in system simulation.

**systems dynamics** A method for studying the world around us. Unlike other scientists who study the world by breaking it up into smaller and smaller pieces, "system dynamicists" look at things as a whole. The central concept of system dynamics is an understanding of how all objects in a system interact with one another. The objects and people in the simulated system interact through feedback loops.

Some general concepts on **CONTINUOUS SYSTEM SIMULATION** are discussed. *Continuous simulation* here refers to concentrated parameter systems and distributed parameter systems. A classification of dynamic systems is reviewed in a summarized form. The main numerical methods for the concentrated parameter systems described by the ordinary differential equations are described. An example of simulating a simple mechanical system is given. The methods of signal flow graphs and bond graphs are discussed. A new, alternate approach is proposed that uses the differential inclusions instead of ordinary differential equations. Next, we remark on the distributed parameter systems, partial differential equation models, and the finite element method. Finally, a short comment is made about the *N*-body problem and galactic simulations. Simulation software for continuous simulation is not discussed in this article, because of the encyclopedic character of the text. Any software described in publications of such type may be obsolete within a few years, whereas the more general concepts do not change so quickly.

## I. INTRODUCTION

Roughly speaking, *continuous simulation* is one of the two main fields of computer simulation and modeling, the other being *discrete event simulation*. Continuous models include those of *concentrated parameter systems* and *distributed parameter systems*. The former group of models includes those for which the power of the set of all possible states (or, more precisely, the number of classes of equivalence of inputs) is equal to the power of the set of real numbers, and the latter refers to systems for which that set is greater than the set of reals. These classes of dynamic systems are described in more detail in the next section. The most common mathematical tools for continuous modeling and simulation are the ordinary differential equations (ODEs) and the partial differential equations (PDEs).

First of all, we must remember that in the digital computer nothing is continuous, so the process of using continuous simulation with this hardware is an illusion. Historically, the first (and only) devices that did realize continuous simulation were the analog computers. Those machines are able to simulate truly continuous and parallel processes. The development of digital machines made it necessary to look for new numerical methods and their implementations in order to get good approximations for the solution of both ordinary and partial differential equations. This aim has been achieved to some extent, so we have access to quite good software tools for continuous simulation.

In the present article some of the main algorithms are discussed, like the methods of Euler, Runge–Kutta, multistep, predictor-corrector, Richardson extrapolation, midpoint for the ODEs, and the main finite difference and finite element methods for the PDEs.

To illustrate the very elemental reason why continuous simulation on a digital computer is only an imperfect approximation of the real system dynamics, consider a simple model of an integrator. This is a continuous device that receives an input signal and provides the output as the integral of the input. The differential equation that describes the device is

$$dx/dt = u(t) \tag{1}$$

where $u$ is the input and $x$ is the output. The obvious and most simple algorithm that can be applied on a digital computer is to discretize the time variable and advance the time from 0 to the desired final time in small intervals $h$. The iterative formula can be

$$x(t+h) = x(t) + hu(t) \tag{2}$$

given the initial condition $x(0)$. This is a simple "rectangle rule" that approximates the area below the curve $u(t)$ using a series of rectangles. The result is always charged with certain error. From the mathematical point of view this algorithm is quite good for regular input signals, because the error tends to zero when $h$ approaches zero, so we can obtain any required accuracy.

Suppose now that our task is to simulate the integrator over the time interval [0,1] with $u = \text{const} = 1$. We want to implement the above algorithm on a computer on which real numbers are stored to a resolution of eight significant digits. To achieve high accuracy of the simulation we execute the corresponding program of Eq. (2) several times, with $h$ approaching zero. One can expect that the error will also approach zero. Unfortunately, this is not the case. Observe, that if $h < 0.000000001$, the result of the sum operation at the right-hand side of Eq. (2) is equal to $x(t)$ instead of $x(t) + \text{h}u(t)$ because of the arithmetic resolution of the computer. So, the error does not tend to zero when $h$ becomes small, and the final result may be zero instead of one (integral of 1 over [0,1]). This example is rather primitive, but it shows the important fact that we cannot construct a series of digital simulations of a continuous problem that tends to the exact solution—at least theoretically. Of course, we have a huge number of numerical methods that guarantee sufficiently small errors and are used with good results, but we must be careful with any numerical algorithm and be aware of the requirements regarding the simulated signals to avoid serious methodological errors. A simple fact that we always must take into account is that in a digital computer real numbers does not exist, and are always represented as their rough approximations.

## II. DYNAMIC SYSTEMS

Computer simulation is an implementation of a system model on a computer. The aim of modeling and simulation is to observe the changes of the model state over a given time interval. The fundamental concepts of dynamic systems are the model state and causality. Roughly speaking, the system state is a minimal set of data that permits us to calculate the future system trajectory, given the actual state and all system inputs (external excitations) over the time interval under cosideration. A dynamic system is causal if its actual state depends on its previous states and the previous and actual external excitations only. Another notion of causality is the input/output causality relation used in signal processing, instrumentation and automatic control.

Note that these two causality concepts are quite different. An electronic amplifier has its input and output signals well defined and obviously the input is the cause and the output is the result, not vice versa. However, in physical systems this concept does not work. For a moving mass the relevant variables are the dynamic force and the acceleration. But if the mass is the only isolated element of our model, we cannot say which variable is the cause and which one is the result. The same occurs with an electric resistor. Unless you define to what device it is connected (e.g., voltage source or current source), you cannot say if the cause is the current or the voltage.

One well-known classification of dynamic systems is based on the number of *classes of equivalence of input signals*. This is valid only for causal systems that have well-defined input and output signals. A set of inputs forms an equivalence class over a given time interval if the system state and system outputs at the end of the interval are the same for all inputs belonging to the set. Practically, this means that the number of classes of equivalence of inputs is the number of all possible system states at the end of the considered time interval. Let us denote this set of states by $S$. Then we can define the following groups of dynamic systems:

- *Finite automata* is a system for which the number of classes of equivalence of inputs $N$ is finite. For example, an electric switch is a finite automata. A digital computer also belongs to this class, though it may have trillions of possible states.

- *Infinite automata* is a system for which $N$ is infinite, but the set of possible states is enumerable. In other words, for this class of systems the power of the set $S$ is equal to the power of the set of integers.
- *Concentrated parameter systems* are those for which $N$ is infinite, and the elements of the set $S$ are not enumerable. For this class of systems the power of the set $S$ (its cardinal number) is equal to the power of the set of all real numbers. This is a wide class of systems that includes mechanical systems or electric circuits. However, the parameters of the systems of this class must be concentrated in specific discrete elements, like ideal springs, dampers, capacitors, and so on. The most common modeling and simulation tool for concentrated parameter systems are the ODEs.
- *Distributed parameter systems* result when the power of the set $S$ is greater than the power of the set of reals. In other words, for such systems the number of possible states is greater than the number of all reals. A classic example is a guitar string. Its state is a continuous three-dimensional function over the string's initial length. It is known that no one-to-one mapping between real numbers and continuous functions can be established, because the power of the set of all such functions is greater than the power of reals. The parameters of systems of such kind cannot be concentrated in discrete components like ideal capacitors or dampers. Other examples are heat transfer problems, waves on the surface of water, and fluid dynamics. The main mathematical tool for this class of systems are the PDEs.

Continuous simulation is applied to the simulation of the last two classes of dynamic systems, namely, the concentrated and distributed parameter systems.

## III. ORDINARY DIFFERENTIAL EQUATIONS AND MODELS OF CONCENTRATED PARAMETER SYSTEMS

An ordinary differential equation in its general form can be expressed as follows.

$$F(x, x^{(1)}, x^{(2)}, ..., x^{(n)}, t) = 0 \qquad (3)$$

where $t$ is the independent variable representing the model time, $x$ is the dependent variable, and $x^{(1)}=dx/dt$, $x^{(2)}=d^2x/dt^2$, etc. The order of the equation is equal to the order of the highest order derivative of the dependent variable.

ODEs have been used to describe dynamic systems, mainly concentrated parameter systems. This caused the common believe that ODE modeling is the only tool to simulate such systems. In fact, this is not true. First of all, we must be sure that the following conditions are satisfied:

1. There exists a differential equation that represents a valid model of our system.
2. If so, we must know if there exists a (unique) solution to our ODE with given initial conditions.
3. If conditions 1 and 2 are satisfied, we must determine if the ODE model we use can provide solutions that satisfy the aim of the simulation task.

The above conditions seem to be obvious, but in practice few simulationists check them. There is a strange belief that everything can be simulated with ODEs and that solutions provided by simulation software represent or approximate the real system behavior. Unfortunately, this is not the case. In my opinion, ODEs are too primitive to be applied to global models of systems like industrial dynamics, ecology, or microbiology. In Section IX of this article you can find a description of other possible mathematical tools, namely differential inclusions.

If we can resolve Eq. (3) with respect to the highest derivative of the dependent variable, then we can find the equivalent set of the first-order equations. Indeed, from Eq. (1) we have

$$x^{(n)} = f(x, x^{(1)}, x^{(2)}, ..., x^{(n-1)}, t) \qquad (4)$$

Now, let us denote $x = x_1$, $x^{(1)} = x_2$, $x^{(2)} = x_3$, etc. So,

$$\begin{aligned} dx_1/dt &= x_2 \\ dx_2/dt &= x_3 \\ &\vdots \\ dx_n/dt &= f(x, x_1, x_2, ..., x_n, t) \end{aligned} \qquad (5)$$

The last equation can be written in vectorial form as follows:

$$d\mathbf{x}/dt = \boldsymbol{f}(\mathbf{x},t) \qquad (6)$$

where the boldface letters denote vectors. For example, the following equation of an oscillator

$$ax + b\ dx/dt + c\ d^2x/dt^2 = 0 \qquad (7)$$

is equivalent to the following set of two equations of the first order:

$$\begin{aligned} dx_1/dt &= x_2 \\ dx_2/dt &= -(ax_1 + bx_2)/c \end{aligned} \qquad (8)$$

The form of the set of first-order equations is important in computer simulation, because most of the numerical methods and their implementations use this mathematical model. Normally, the user is only asked to give the right-hand sides of the Eqs. (5) and define the initial conditions and other model parameters. The software should do the rest of the simulation task automatically.

The numerical methods used in continuous systems simulation are discussed in Section V. The next section contains some comments on continuous simulation on analog computers.

## IV. CONTINUOUS SIMULATION WITH ANALOG COMPUTERS

A typical example of an analog simulation is a wind-tunnel test on a scaled-down physical model of an airplane. Another way to do an analog simulation is to look for an analogy between one physical model and another, for example, a mechanical system and an electric circuit. A simple electronic circuit with one operational amplifier, one resistor, and one capacitor can realize the operation of mathematical integration. As a consequence, it is possible to solve differential equations using combinations of such circuits. The advanced devices with many such circuits and variable interconnections between them are called *analog computers.*

In the early 1940s and 1950s, analog computers were commonly used to simulate continuous dynamic systems, mainly, automatic control systems, mechanical systems, and others. During the last decades analog computers have been losing importance. However, we should remember that analog computers are truly continuous PDE solvers. As stated in the introduction, the continuous simulation of digital computers is a much more "artificial" approximation of a real continuous problems.

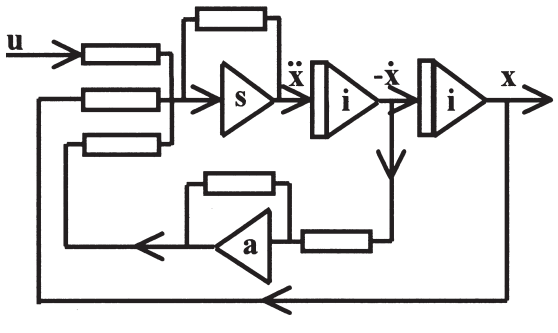


**Figure 1** An analog model with integrators.

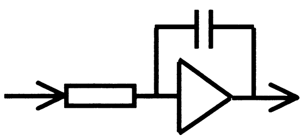


**Figure 2** Analog integrator circuit.

Figure 1 shows an analog circuit that satisfies the equation

$$d^2x/dt^2 = -u - x - adx/dt$$

where $u(t)$ is an external input and $x(t)$ is the model response. In Fig. 1, operational amplifiers marked with an i are integrators, the amplifier marked with with an s is a summator, and the amplifier marked with an a is an inverting amplifier. All amplifiers are supposed to have negative infinite gain. The rectangles represent resistors. The analog integrator is realized by the circuit shown in Fig. 2.

## V. NUMERICAL METHODS FOR ORDINARY DIFFERENTIAL EQUATIONS

Note that this article is on simulation and not on mathematics. This section gives only a very short description of some of the most used methods. See the Bibliography for more details.

As mentioned in Section III, the ODE models in most cases can be expressed in the form of a set of ODEs of the first order, as follows:

$$dx_i/dt = f_i(x_1, x_2, ..., x_n, t)$$

where $x_i$ are dependent variables, $i = 1, 2, 3, ..., n$, and $t$ is the independent variable, representing the model time.

The *boundary conditions* for these variables may consist of requiring that certain variables have certain numerical values in a given time instant. They can be more complicated, given, for example, in the form of a set of nonlinear algebraic equations for the state variables. The nature of the boundary conditions determines which numerical methods will be feasible for solving the problem. Boundary conditions divide into two broad categories:

- *Initial value problems,* in which all the dependent variables are given at some starting time instant and we want to calculate their changes over some time interval or in some fixed time instants.
- *Two-point boundary value problems,* in which the conditions are specified at more than one time

instant. Typically, some of the conditions are specified at a starting point and others at the final time. Problems of such kind arise in optimal control theory rather than simulation and are not discussed here.

The simplest way to solve the initial value problem is to rewrite the *dxs* and *dts* as finite increments *Dx* and *Dt*, and to multiply the equations by *Dt*. This gives an algebraic formula for the change in the dependent variables, when the time is incremented by the step size *Dt*. In the limit case while the step size approaches zero, the solution may be a good approximation of the real one. The implementation of this method is *Euler's method.* Euler's method is simple but not recommended for several reasons, the most important being the fact that approximation errors may accumulate during the iteration process.

In this article, we discuss the most commonly used methods:

- Runge-Kutta methods
- Richardson approximations
- Predictor-corrector methods.

## A. Runge–Kutta Methods

This is one of the most popular families of algorithms that solve ODEs. Recall that the ODE models in most cases can be expressed in the form of a set of ODEs of the first order:

$$dx_i/dt = f_i(x_1, x_2, ..., x_n, t)$$

where $x_i$, $i = 1, 2, 3, ..., n$, are dependent variables, and $t$ is the independent variable, representing the model time in dynamic systems.

The main disadvantage of Euler's method is that the solution advances through the integration step $h$, but uses derivative information only at the beginning of that interval. To avoid this, we can take a "trial" step to the midpoint of the interval. Then we can use the values of $x$ and $t$ at that point to recalculate the derivative and compute the "real" step over the whole interval. This results in the following algorithm:

$$k_1 = hf(x_j, t_j)$$

$$k_2 = hf(x_j + k_1/2, t_j + h/2)$$

$$x_{j+1} = x_j + k_2 + O(h^3)$$

where $x_j$ is the vector $x$ at the time instant $hj$, $j$ being the consecutive step number. The above equations represent the *second-order Runge–Kutta* or *midpoint* method. The term $O(h^3)$ is a function that tends to zero as fast as $h^3$, when $h$ approaches zero.

Runge–Kutta methods propagate a solution over an interval by combining the information from several Euler-style steps, each involving one evaluation of the right-hand sides of the equations. Then, the information obtained from these evaluations is used to match a Taylor series expansion of the solution to some higher order curve.

For example, the *fourth-order Runge–Kutta algorithm* is given by the following formula:

$$k_1 = hf(x_j, t_j)$$

$$k_2 = hf(x_j + k_1/2, t_j + h/2)$$

$$k_3 = hf(x_j + k_2/2, t_j + h/2)$$

$$k_4 = hf(x_j + k_3, t_j + h)$$

$$x_{j+1} = x_j + k_1/6 + k_2/3 + k_3/3 + k_4/6 + O(h^5)$$

where $h$ is the integration step (time increment),

$$x_j = x(t_j), \qquad x_{j+1} = x(t_j + h)$$

where $j = 1, 2, 3, ...$ is the consecutive step number, and $k_1$, $k_2$, $k_3$, and $k_4$ are auxiliary variables. $O(z)$ is a function that tends to zero at least as fast as its argument $z$ does. This means that the above algorithm will approximate well any real solution that is a fourth-order curve.

There are many modifications and higher order versions of the Runge–Kutta method. For example, a widely used fifth-order Runge–Kutta–Fehlberg algorithm provides a highly accurate solution and also an error estimate. Using this estimate the program can repeat the integration step with smaller time increments to achieve the desired accuracy.

One of the disadvantages of the Runge–Kutta methods is the fact that for one step evaluation we need several evaluations (e.g., four for the fourth-order method) of the system's right-hand sides. The regularity assumptions for these methods consist of the continuous differentiability assumption (up to order $N$) for the right-hand sides of the equations. On the other hand, the method has no "memory," that is, no information from previous steps is used. This makes the method somewhat more robust compared to other algorithms, like the multistep predictor-corrector methods.

## B. Richardson Approximations

The Richardson approximations improve the accuracy of an approximated solution to a set of the first-order ordinary differential equations. The true solution is assumed to be an analytic function of the size of the integration step $h$. The function can be evaluated with various values of $h$, none of them small enough to yield the accuracy we desire. Then, we fit the obtained solutions to some analytic function and evaluate it at the point $h = 0$. For example, the points can be the results of evaluating the solution by the Elle's method in 2, 4, and 6 steps. These three points can be used to evaluate the solution with $h = 0$ assuming the solution is a second-order polynomial function of $h$.

## C. Predictor-Corrector Methods

These methods belong to the family of *multistep* methods. They are used primarily in problems with very "smooth" equations and complicated right-hand sides. If this is not the case, the Runge–Kutta or Richardson approximation methods dominate.

In the multistep methods we approximate the equation's right-hand side by a polynomial passing through several previous points and possibly through the actual point, being evaluated at $t = (n+1)h$. If this point is included, the method is said to be *implicit;* otherwise, it is *explicit.* The general formula is as follows.

$$x_{n+1} = x_n + h(\beta_0 X_{n+1} + \beta_1 X_n + \beta_2 X_{n-1} + ...)$$

where $X_k$ denotes $f(x_k, t_k)$. If $\beta_0 \neq 0$ then the method is implicit. The implicit methods must solve an algebraic equation for

$$x_{n+1} = x((n+1)h)$$

because this value has not been calculated yet and appears on both sides of the general formula. Normally, the Newton–Raphson method is used. If the value of $x$ at $(n+1)h$ is not used, the approximation can be calculated in one step, and the method is explicit.

The predictor-corrector method uses both approximations; first, an explicit method is used to estimate the value of $x((n+1)h)$, and then the implicit method calculates (corrects) it using this approximation as the starting point for the Newton–Raphson method. This procedure seems to be complicated, but you should note that all of these operations need only one evaluation of the equation's right-hand sides per integration step. If the model is complicated, the cost of computing (real CPU time) depends mainly on the number of right-hand side evaluations, which makes the multistep methods several times faster than other methods, e.g., Runge–Kutta. On the other hand, any multistep method must use some previously stored values of the dependent variable. If the equations are not regular enough over these consecutive steps, the method fails. In particular, if our model receives discontinuous excitations or changes its structure (for example, switching controllers), the multistep methods are not recommended.

As an example, see the following Adams–Bushworth–Moulton corrector-predictor scheme:

$$x_{n+1} = x_n + \frac{h}{12}(23X_n - 16X_{n-1} + 5X_{n-2}) \qquad \text{(predictor)}$$

$$x_{n+1} = x_n + \frac{h}{12}(5X_{n+1} + 8X_n - X_{n-1}) \qquad \text{(corrector)}$$

Here $X_k$ denotes $f(x_k, t_k)$. Note that in the corrector part the value of $x$ at $(n+1)h$ is taken from the predictor formula, so that the corrector needs no Newton–Raphson iterations to be performed ($x_{n+1}$ does not appear at the right-hand side of the corrector equation).

Using any of the above formulas for the approximation of the solution on the next model time step, we can create our simulation program. Simply the main time loop must be coded, where the model time is advanced by the small integration step $h$ and the resulting model state is used as the initial state for the next time step. In such a way, most of the simulation packages for ODE models work. Many of these programs are able to change the integration step if necessary. A common algorithm that is used is the combination of the Runge–Kutta–Fehlberg method with an appropriate step-change algorithm.

A very important and difficult problem in any numerical method for the ODEs is the numerical stability. The lack of numerical stability may lead to wrong and qualitatively bad solutions. Our model can be described by correct and stable equations, while the simulation results may reveal unstable behavior. Sometimes it is easy to misinterpret this instability as a lack of stability in the original model. The numerical stability depends, of course, on the particular algorithm. For example, the stability condition for the Runge–Kutta algorithm of the fourth order is $2.78 < rh < 0$, where $h$ is the integration step and $r$ depends on the model. If the model is linear and one dimensional, then $r = df/dx$, where $f$ is the equation right-hand side and $x$ is the state variable.

## D. Stiff Equations

Stiffness occurs when there are two or more very different timescales, for example, in systems that have very fast and very slow parts interacting with each other. Consider the following set of equations:

$$dx_1/dt = 998x_1 + 1998x_2$$

$$dx_2/dt = -999x_1 - 1999x_2$$

$$x_1(0) = 0, \qquad x_2(0) = 0$$

The solution to these equations is

$$x_1(t) = 2e^{-t} - e^{-1000t}$$
$$x_2(t) = -e^{-t} + e^{-1000x}$$

To integrate this system of equations, we must define the step size of any numerical methods $h << 0.001$ if we want the method to be stable. However, the term $e^{-1000t}$ disappears very quickly when $t$ grows. One could suppose that for $t$ between, say, 0.1 and 1, we could apply a greater integration step because the fast term does not influence the solution. Unfortunately, that is not the case, because the integration process becomes unstable even if this term is negligible. This problem is called *stiffness* and it provokes serious difficulties in system simulation.

The most popular methods for stiff equations are these:

- Generalizations of the Runge–Kutta method, of which the most useful are the Rosenbrock methods, and their implementation by Kaps–Rentrop
- Generalization of the Burlisch–Stoer method, due to Bader and Deufhard
- Predictor-corrector methods

## VI. EXAMPLE OF CONTINUOUS SIMULATION USING ODES

Let us simulate the behavior of the suspension of a car. First of all, the simulationist should ask the fundamental question: For what? Remember that to simulate something without defining the aim of the task is a waste of time. In our case, the problem is to see how the system response depends on the parameter of the damper. This can help the designer to choose the damper that provides an acceptable response of the vehicle to a rectangular obstacle on the road (Fig. 3).

This is a very simplified model, where only one wheel is considered and only vertical movement is simulated. In the more advanced models of vehicle dynamics, we must simulate all of the forces the vehicle receives and look for the three-dimensional model of the system. But to start with a simple academic example, the following model can be quite illustrative.

Suppose that the car moves forward with a constant horizontal velocity $V$. So, considering all variables as differences between their actual values and the initial equilibrium state, we take into account the following:

$y$ = movement of the wheel
$x$ = movement of the vehicle
$F_1$ = force determined by the compliance of the tire
$F_2$ = force of the suspension spring
$F_a$ = force produced by the damper
$u$ = external excitation, due to the shape of the road

All of the above variables are functions of time and all represent vertical movements and forces. $M_1$ and $M_2$ represent the mass of the wheel (together with the moving part of the suspension) and the mass of the car, respectively (M2 being 1/4 of the car total mass).

To derive the model equations we must use the force balance for the two masses, including the dynamic forces. This provides the following equations.

$$\begin{aligned} F_1 - F_2 - F_a - M_1 d^2y/dt^2 &= 0 \\ F_2 + F_a - M_2 d^2x/dt^2 &= 0 \end{aligned} \tag{9}$$

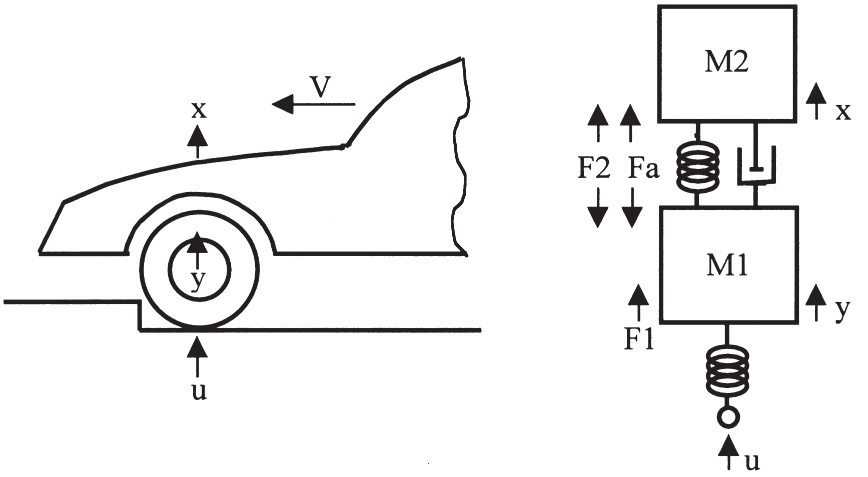


**Figure 3** A suspension of a car and its mechanical scheme.

The forces of the two springs and of the damper depend on the corresponding changes of the spring and damper length, so

$$F_1 = K_1(u - y), \qquad F_2 = K_2(x - y)$$
$$F_a = K_a(dx/dt - dy/dt)$$

Note that the damper force depends on the velocities and not positions. The above forces are supposed to be linear with respect to the positions and velocities. The next version of the model might include nonlinear functions for the forces. This would not complicate significantly our simulation task; the only necessary change would be to replace the above expressions with nonlinear functions.

Because the most convenient form of an ODE model is a set of equations of the first order, let us redefine the equations using the following notation:

$$x_1 = x, \quad x_2 = dx/dt, \quad x_3 = y, \quad x_4 = dy/dt$$

After substituting the new variables into Eq. (9) and reordering the equations, we obtain the following:

$$\begin{aligned} dx_1/dt &= x_2 \\ dx_2/dt &= [K_2(x_3 - x_1) + K_a(x_4 - x_2)]/M_2 \\ dx_3/dt &= x_4 \qquad\qquad (10) \\ dx_4/dt &= [K_1(u(t) - x_3) - K_2(x_3 - x_1) \\ &\quad - K_a(x_4 - x_2)]/M_1 \end{aligned}$$

The last set of equations can be used to simulate our system. Note that we obtain four equations, and that the state vector of the model is $x = (x_1, x_2, x_3, x_4)$. This is correct, because the movement equation for each mass is of the second order.

Now, we must decide what to do with our mathematical model. A beginner could choose the following procedure: First, find a numerical algorithm to solve the equations, for example, Runge–Kutta–Fehlberg. The algorithm can be found in any book on numerical methods. Then, prepare the corresponding code, in Basic, Pascal, Fortran, C, or some other programming language. Insert the code of our model [Eq. (10)] in the program, compile it, and run it.

This may result in a correct simulation program and provide good results, but in most cases it is simply a waste of time. The same task can be completed 20 times faster while using an appropriate simulation language. In any directory of simulation software, like the Directory of Simulation Software of the Society for Computer Simulation you can find hundreds of simulation languages and packages, at least half of them for continuous ODE models. A good ODE simulation package should only ask you to type the right-hand sides of the model equations, and give model parameters and initial conditions. The rest should be done automatically, providing all needed reports, trajectory plots, etc. Figures 4 and 5 show examples of simulation results. The plots were generated by the ODE module of the PASION simulation system

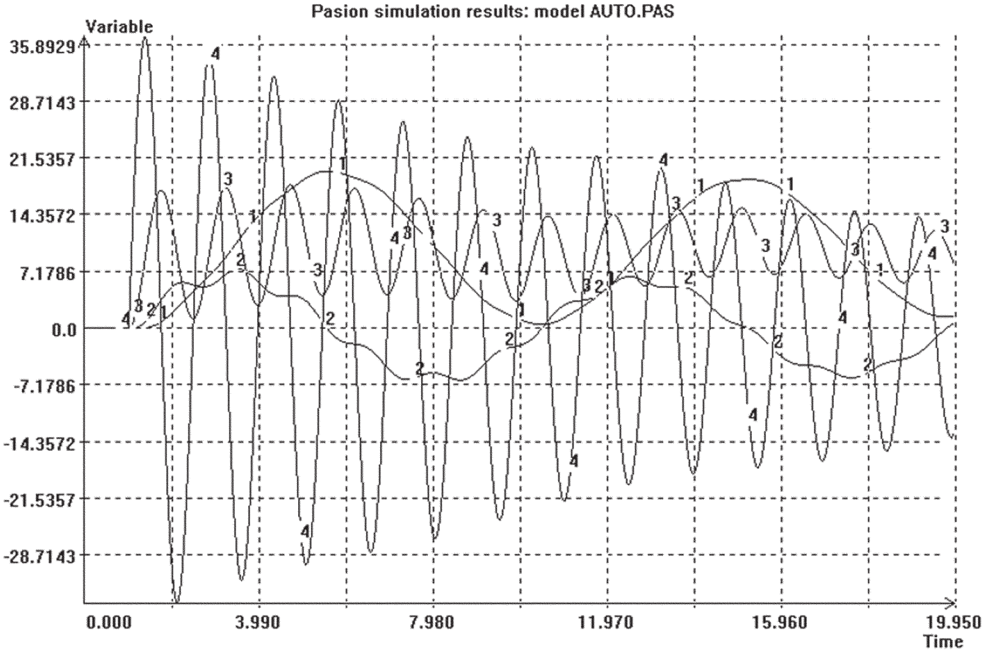


**Figure 4** System trajectory. 1, car position; 2, car (vertical) velocity; 3, wheel position; 4, wheel velocity. Dumping coefficient $F_a = 5$.

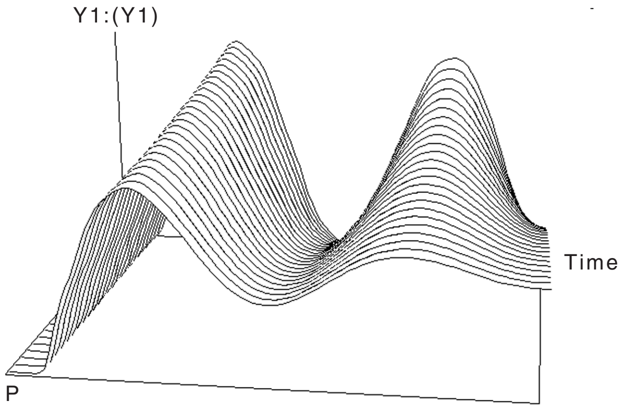


**Figure 5** A set of system trajectories with different values for the damper parameter.

(http://www.raczynski.com/pn/pn.htm). The integration algorithm was the fifth-order Runge–Kutta–Fehlberg algorithm, with about 2000 integration steps over the interval of 20 sec of model time. The real time of simulation is less than 0.1 sec on a 450-MHz PC. The system parameters are as follows: $M_1 = 50$ kg, $M_2 = 200$ kg, $K_1 = 1000$ N/cm, $K_2 = 100$ N/cm. The external excitation $u(t)$ was a step function with amplitude 10 cm, starting at $t = 1$ sec. Note that if you use the SI unit system, the mass must be given in kilograms and the force in newtons.

The aim of our task is to see how the damper parameter $F_a$ affects system performance. The value used for the simulation of Figure 4 is too small, and results in a highly oscillatory response. Figure 5 shows the result of a simulation experiment where $F_a$ changes automatically from 5–80 in 25 steps. A trajectory is automatically simulated for each value and the 3-D plot shows the changes in the trajectory shape. The vertical coordinate is the car vertical position, the axis from left to right is the model time and the other one (marked P) is the value of $F_a$. By running the program several times the designer can choose a satisfactory value for $F_a$.

Most of the dynamic systems are subject to stochastic disturbances. The same package permits other kinds of experiments, where one or more inputs are random signals. Figure 6 shows the trajectory of our model (car vertical position), where the excitation $u$ is as before, plus a uniformly distributed random value with mean zero and amplitude 20, changing at each integration step of 0.01 sec. The plot of Fig. 6 shows the confidence intervals for the simulated variable as a function of time. The confidence level is 0.9, which means that with probability 0.9 we are within the limits marked with vertical sections.

In Fig. 7 we can see the probability density function for the same experiment. "Output Y[1]" is the car position and the vertical value is the probability

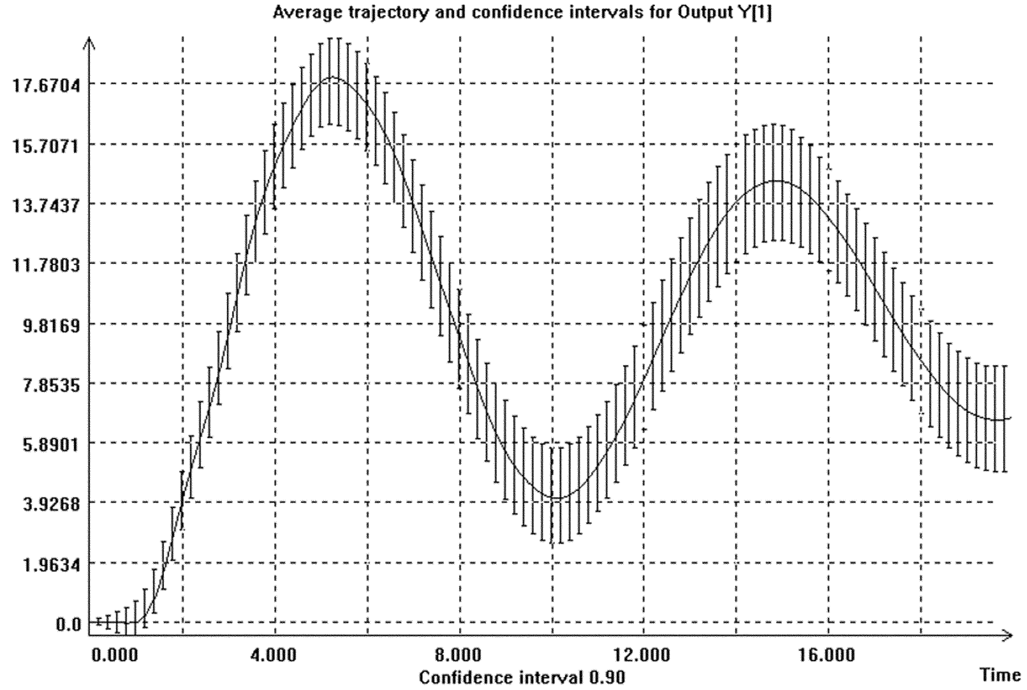


**Figure 6** A system trajectory and confidence intervals in the case of a stochastic disturbance. Damper parameter Fa = 30.

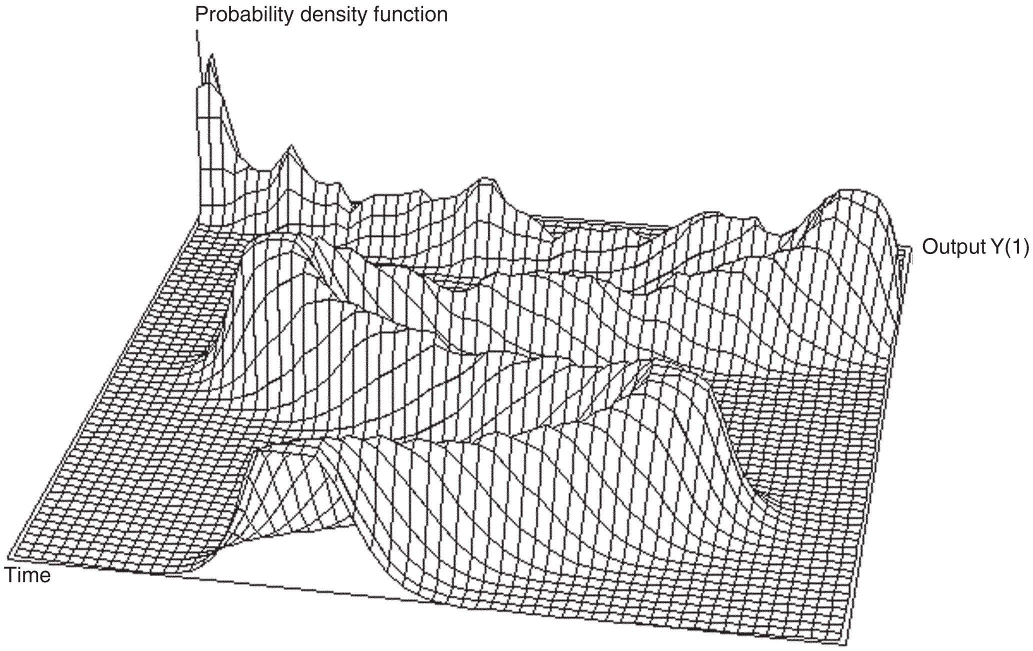


**Figure 7** The probability density function for car elevation.

density function for the corresponding time–position point.

The preceding example shows how you could prepare an ODE model and what should result based on the simulation tool used. This can serve as a criterion to select the software. Of course, PASION is not the only package that does the job. The Internet offers many similar tools; use the search keywords *simulation, ODE, continuous, modeling, dynamic system,* or similar terms.

## VII. SIGNAL FLOW GRAPHS

Signal flow graphs (SFGs) can be used to represent the dynamics of a modeled system, instead of differential equations. The very traditional way of simulating dynamic systems has been to obtain the system equations, prepare the corresponding code in a programming language, and run the simulation. However, note that a simulationist need not be a mathematician or a programmer. The simulation software and new modeling methods make it possible to eliminate both the mathematics and programming from the simulation task. What the simulationist must do is understand the structure and the dynamics of the modeled system and be able to describe it precisely enough to be interpreted by a computer. Both SFGs and bond graphs (see the next section) are graphical representations of dynamic system models. If the model is described in such a graphical way, the simulation software should automatically generate model equations and the corresponding code.

When we refer to a SFG we mean a network composed of nodes and directed links. Nodes represent signals and links represent transfer functions. The direction of a link shows which signal is the input to the link (the cause) and which is the output (the result). Figure 8 shows a graph that describes an integrator. Signal B depends on the signal A, and not vice versa. The transfer function (in terms of the Laplace transform) for the link A → B is $1/s$, which means that B is the integral of A.

The equation $A = dB/dt$ is also valid for this graph. However, we must remember that A is the cause and B is the result. If two or more links enter the same node, then the signal at the node is the sum of signals generated by the entering links. Figure 9 shows a simple circuit with an operational amplifier (a follower) and the corresponding signal flow graph.

Consider a simple mechanical system composed of a spring, a mass and a damper as shown in Fig. 10. The

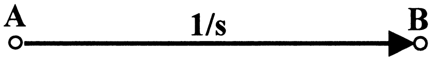


**Figure 8** An example of a link of a signal flow graph.

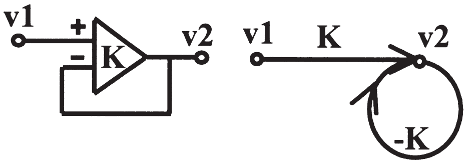


**Figure 9** Simple circuit with operational amplifier and corresponding SFG.

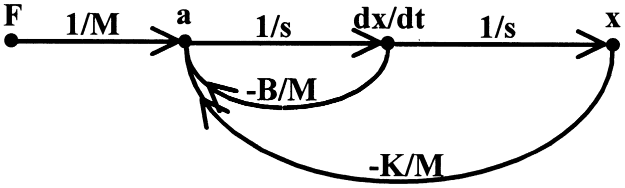


**Figure 11** The signal flow graph for the system of Fig. 10.

movement of the mass is the result of an external force *F* and of the two forces produced by the spring and by the damper (no gravity force supposed). The force of the spring is supposed to be equal to *kx* where *k* is a constant, and the force of the damper is *Bdx/dt*, *B* being a constant. Figure 11 shows a graph, which describes the system (*a* is the acceleration). The corresponding differential equation is

$$(F(t) - kx(t) - Bdx/dt)/M = \mathrm{a}$$

Note that *F*(*t*) is a source node of the graph, that is, it does not depend on any internal signal of the system.

Figure 12 represents the dynamics of a feedback control system. The signal *u* is the set point, *e* is the control error, *x* is the controlled physical variable, and *h* is its measured value. The link PID is the controller with proportional–integral–derivative action, *G*(*s*) is the transfer function of the controlled process, and *H*(*s*) is the dynamics of the measurement instrument. Using SFGs the user paints the graph on the screen, then specifies the transfer functions for the links. The simulation software should do the rest. For example, the flow graph module of the PASION simulation system does the job. The program permits various types of links like linear or nonlinear function, time delay, sample-and-hold, transfer function, or a "superlink," which is a whole model prepared earlier.

The generation of the system equations and the corresponding code is completely automatic and transparent to the user. This permits the simulationist to concentrate on the most relevant and important conceptual work, and not on reordering equations and programming.

## VIII. BOND GRAPHS

Bond graphs are the widespread tool in modeling of physical systems. The fact that a bond connects two variables—the effort and the flow—makes this tool the most appropriate for modeling physical systems, because the power produced at the bond is the product of these two variables (e.g., voltage and current, force and velocity, liquid pressure and flow). This method again permits us to eliminate both mathematics and coding from the simulation task.

A bond graph model is composed by the nodes or junctions and the links named *bonds*. A bond is a directed link with a harpoon. The harpoon is placed on the left of the link (related to its direction). The two variables are indicated shown in Fig. 13. The effort is placed on the side of the harpoon and the flow is on the other side.

There are several types of nodes in bond graph models. At the node of type 0 the sum of flows is equal to zero, while the efforts of all connected bonds are equal to each other. At the node of type 1 the sum of efforts must be zero and the flows of all corresponding bonds are the same. Thus, we can represent graphically any system that obeys any number of balance equations.

The node equations for node 0 and the node 1 are as follows (see the example in Fig. 14).

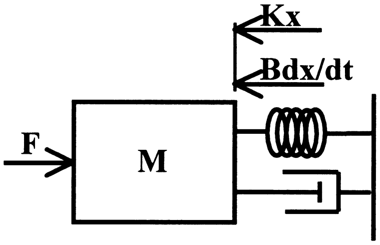


**Figure 10** A mechanical system.

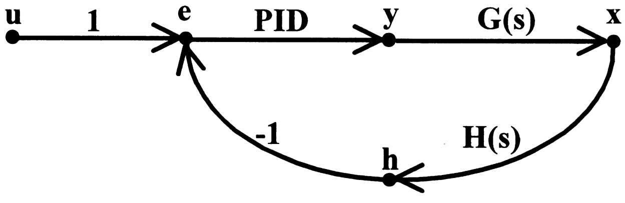


**Figure 12** SFG model of a control system.

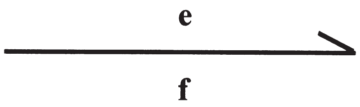


**Figure 13** A bond.

$$f - g - h = 0 \text{ (node of type 0)}$$

$$e - v - w = 0 \text{ (node of type 1)}$$

The sign of a term in the node equation depends on the direction of the corresponding bond, the outgoing bond having the negative sign of the corresponding variable. Other possible nodes accepted by the BOND module of the PASION simulation system are as follows:

*SE node*—effort source, e.g., an external force or ideal voltage source
*SF node*—flow source, e.g., mandated velocity in a translational system, ideal current source
*R node*—dissipative element, e.g., damper or electrical resistance
*C node*—capacitance, e.g., a spring or electrical capacitance
*L/I node*—inertia/inductance, e.g., a moving or rotating mass, electrical inductance.

The causality in bond graph diagrams is denoted by a stroke at one of the ends of the bond. This means that the flow variable is evaluated at the end with the stroke and the effort variable at the other end.

Figure 15 shows the possible combinations of bonds and nodes of type SE, SF, R, L/I, and C, and the implied causalities. All free ends of the bonds can be connected to nodes of type 0 or 1.

The node–bond combinations are as follows:

(a) (SE node) Effort source. The effort e is defined at the node.
(b) (SF node) Flow source. The flow f is defined.

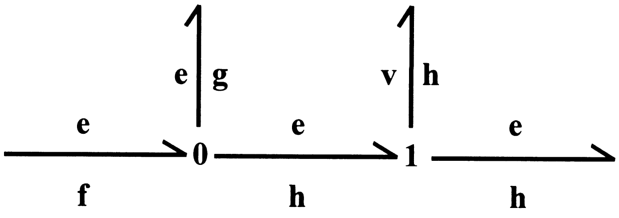


**Figure 14** Nodes of type 0 and 1.

(c),(d) (R node) Dissipative bonds. The equations are $f = e/R$ and $e = fR$, respectively.
(e) (C node) Capacitance. It has the desired causality as shown. The equation is $de/dt = f/C$, where $C$ is the capacitance.
(f) (I/L node) Inertia or Inductance. The equation is $df/dt = e/L$, where $L$ is a constant (mass, inductance, etc.).
(g) (TF bond) Ideal transformer. The equations are $e_1 = m\,e_2$, $f_2 = m\,f_1$, where $m$ is a constant.
(h) (TF bond) Ideal transformer. The equations are $e_2 = e_1/m$, $f_1 = f_2/m$, where $m$ is a constant.
(i) (GY bond) Ideal "gyrator." The equation is $e_1 = rf_2$, $e_2 = rf_1$, where $r$ is a constant.
(j) (GY bond) Ideal "gyrator." The equation is $f_2 = e_1/r$, $f_1 = e_2/r$, where $r$ is a constant.

Note that R node–bond combinations and the TF and GY bonds have two possible causalities, whereas the other bonds have the causalities indicated in the previous figure. These causalities may be given by the user.

Consider a simple mechanical system as shown in Fig. 16. Here $v$ is the velocity of the point $a$, $w$ is the car velocity, and $M$ is the mass of the car. The corresponding bond graph is shown in Fig. 17, where

$p = v - w$
$h$ is the force of the damper
$h = rp$
$dg/dt = p/k$
$r$ is the constant of the damper
$k$ is the constant of the spring

In mechanical systems the efforts are flows and flows are velocities, respectively. In Fig. 17 the parameters $p$ and $h$ are shown as $px$ and $hx$ because $p$ and $h$ are reserved symbols of the software used (BOND GRAPH module of the PASION simulation system).

The causality in bond graph models is not the most important issue. The user model needs no causalities defined. Good bond graph software must determine causalities automatically, then generate the model equations and corresponding code and run the simulation. The only situation, when the user must check the causalities is when the software detects a causality conflict. This can result from a physically invalid model. For example, the causality conflict will occur when you put a voltage source in parallel with a capacitor or a current source in parallel with an inductance. Such models imply a possibility of infinite pulses

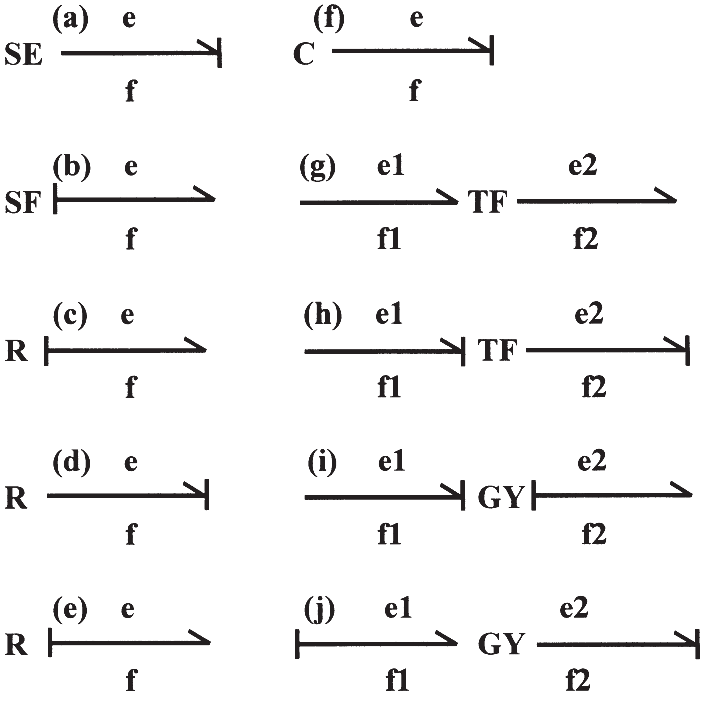


**Figure 15** Bond types.

of current or voltage. This makes it difficult or impossible to calculate the system trajectory, and the simulation fails.

## IX. ALTERNATE MODELING TOOLS AND DYNAMIC UNCERTAINTY

### A. Introduction

As stated in previous sections, the ordinary differential equations represent only one of many possible ways to describe the dynamics of a concentrated parameter system. The widespread old belief that ODEs are the only possible tool in continuous modeling and simulations is wrong. In many cases simulation projects fail because the tools, like ODEs, are too primitive to describe the system on which we are working.

The lack of reliable data in computer simulation is an important obstacle in many simulation projects. Models that are nice and valid from the academic point of view often turn out to be useless in practical applications, when the user cannot provide reliable data. In the past, a common way to treat this lack of exact data was to assume some model parameters or input variables to be random ones. This results in a stochastic model, where every realization of the system trajectory is different, and the problem is rather

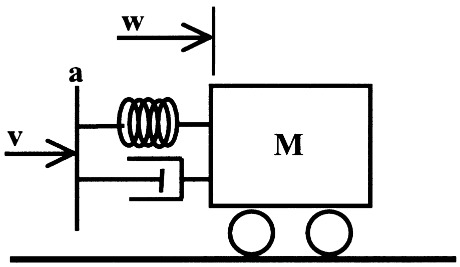


**Figure 16** An example of a mechanical system.

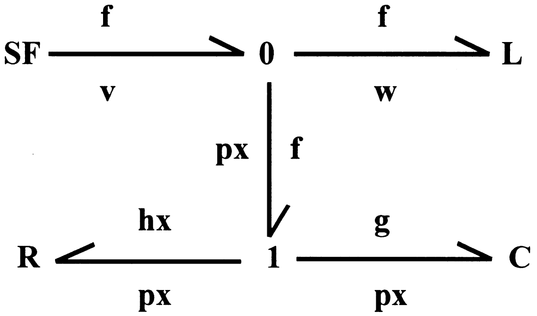


**Figure 17** Bond graph model for the system of Fig. 17.

to determine the probability density function in the system space for certain time-sections, the variance, confidence intervals, etc.

Such stochastic analysis is interesting but not always possible. The common problem is again the lack of data. Some parameters of the model have "uncertain" values, and the user of the model has no idea about their probabilistic behavior. More likely we are given an interval to which the uncertain parameter belongs, instead of its probability distribution or sampled real data. Some external disturbances can fluctuate within certain intervals, and what we are asked to is to give the interval for some output variables. The stochastic simulations with randomized variables do not give such intervals. Moreover, frequently the user wants to know possible extreme values rather than the probability of reaching them. (Recall the law of Murphy!) The uncertainty treatment has nothing, or very little, to do with Monte Carlo or stochastic simulation. The intervals we are looking for are not confidence intervals or any other statistics.

There is no room here to mention the huge number of publications available on uncertainty problems; see the Bibliography.

## B. Differential Inclusions as a Modeling Tool

Most simulationists who deal with continuous systems use, as a basic tool, ordinary or partial differential equations. As suggested before, ODEs and PDEs as modeling tools may be too primitive for many system models except, perhaps, simple mechanisms, circuits, and very academic examples. Let us consider a simple example of a second-order system:

$$d^2y/dt^2 + a\ dy/dt + y = b \tag{11}$$

This is an ODE model. Introducing the notation $x_1 = y$, $x_2 = dy/dt$, we obtain

$$\begin{aligned} dx_1/dt &= x_2 \\ dx_2/dt &= b - ax_2 - x_1 \end{aligned} \tag{12}$$

In more general notation, the state equation is

$$dx/dt = f(a,b,x) \tag{13}$$

where $x$ is a two-dimensional vector, $t$ is the time, and $f$ is a vector-valued function.

Now suppose that parameters $a$ and $b$ are uncertain and that the only information we have is the corresponding intervals where their values may belong, or a permissible irregular set on the plain where the point $(a,b)$ must belong. Note that we know nothing about a possible probability distribution of these parameters and we do not treat them as random variables. Thus, Eq. (13) takes the following form:

$$dx/dt \in F(x,t) \tag{14}$$

where $F$ is a set. What we have obtained is a *differential inclusion* (DI). The right-hand side of Eq. (12) determines the set $F$. However, this is merely one of many possible ways to represent the set. In this case it is parameterized by $a$ and $b$.

The solution to a DI is the reachable set for the possible system trajectories that is exactly the solution to our uncertainty problem. In this very natural way the uncertainty in dynamic system modeling leads to differential inclusions as a corresponding mathematical tool. Note that this tool has been known for about 70 years and that much literature is available on DI theory and applications. The first works were published in 1931–1932 by Marchaud and Zaremba. They used the terms *contingent* or *paratingent* equations. Later, in 1960–1970, Wazewski and his collaborators published a series of works that referred to the DIs as orientor conditions and orientor fields.

As always occurs with new theories, their works received severe criticism, mainly from some physicists who claimed that it is a stupid way of wasting time while dealing with such abstract and useless theories. Fortunately, the authors did not abandon the idea and developed the elemental theory of differential inclusions. In the decade from 1930–1940 such problems as the existence and properties of the solutions to DIs were solved in finite-dimensional space. After this, many works appeared on DIs in more abstract, infinite-dimensional spaces. Within a few years after the first publications about them appeared, DIs became the basic tool in optimal control theory. Recall that optimal trajectories of a dynamic system are those that are on the boundary of the system's reachable set. In the works of Pontiragin, Markus and Lee, Bellman, and many others, one of the fundamental problems is the properties of the reachable sets.

## C. Differential Inclusion Solver

A differential inclusion is a generalization of an ordinary differential equation. In fact, an ODE is a special case of a DI, where the right-hand $F$ is a one-point set. One could expect that a solution algorthm for a DI might be obtained as some extension of a known algorithm for the ODEs. Unfortunately, this is not the case. First of all, the solution to a DI is a set. In particular, it is a set in the time–state space, where the graphs of all possible trajectories of a DI are included. Finding the boundary of such a set (named *reachable set,* or *emission zone* as in the works of Zaremba and Wazewski) is not an easy task.

One of the properties of a reachable set (RS) is the fact that if a trajectory reaches a point on the boundary of the RS at the final time, then its entire graph must belong to the RS. This fact is well known and used in the optimal control theory. Observe that any trajectory that reaches a point on the boundary of the RS is optimal in some sense. Such trajectories can be calculated using several methods, the main one being the maximum principle of Pontriagin. This can be used to construct an algorithm for RS determination. If we can calculate a sufficient number of trajectories that scan the RS boundary, then we can see its shape. The trajectories should be uniformly distributed over the RS boundary. This can be done by some kind of random shooting over the RS boundary. Such shooting has nothing to do with a simple random shooting, when the trajectories are generated randomly inside the RS.

My first attempts to develop a DI solver were presented at the IFAC Symposium on Optimization Methods on Varna in 1974. This was a random shooting method, but not a simple shooting. That algorithm generated trajectories inside the RS, but the control variable had been modified to obtain a nearly uniform distribution of ponts inside the RS at the end of the simulated time interval. The DI solver presented in the *Transactions on SCS* in 1996 is much more effective.

Briefly, the DI solver works as follows. The user provides the DI in the form of an equivalent control system. To do this, he or she must parametrize the right-hand size (the set $F$) using an $m$-dimensional auxiliary variable $u$. The DI solver determines the equations of so-called conjugated vector $p$ and integrates a set of trajectories, each of them belonging to the boundary of the RS. Over each trajectory the Hamiltonian $H(x,p,u,t)$ is maximized. This procedure is similar to that used in dynamic optimization.

In the optimal control problem the main difficulty exists in the boundary conditions for the state and conjugated vectors. For the state vector we have the initial condition defined, and for the conjugated vector only the final conditions (at the end of the trajectory) are known, given by the transversality conditions. This means that the optimal control algorithm must resolve the corresponding two-point-boundary value problem. In the case of a DI we are in a better situation. There is no object function and no transversality conditions. As a consequence, for vector $p$ we can define the final as well as the initial conditions.

At any rate, we obtain a trajectory whose graph belongs to the RS boundary. Defining initial conditions for $p$ we integrate the trajectory only once, going forward. The only problem is how to generate these initial conditions in order to scan the RS boundary with uniform density. The algorithm is quite simple: The initial conditions for $p$ are generated randomly, due to a density function that changes, covering with more density points that correspond to trajectories that fall into a low-density region at the RS boundary. Trajectories that are very close to each other are not stored (storing only one from each eventual cluster). As a result, we obtain a set of trajectories covering the RS boundary that can be observed in graphical form and processed.

## D. An Example

Consider a second-order dynamic system where the acceleration and damping coefficient are uncertain. An example of the corresponding DI in parametrized form is as follows:

$$dx_1/dt = x_2$$

$$dx_2/dt = u_1 - u_2x_2 - x_1$$

where $-1 < u_1 < 1$ and $0.5 < u_2 < 1.5$. Figure 18 shows the 3-D image of the RS in coordinates $x_1$ (upward), $x_2$ (to the right), and $t$. In Fig. 19 you can see a time section of the RS for some fixed time. Observe the small cluster of trajectories at the origin of the coordinate system. These trajectories (10,000 in total) are obtained by a simple random shooting, where both controls had been genrated randomly within the limits defined above. The other pixels (the RS con-

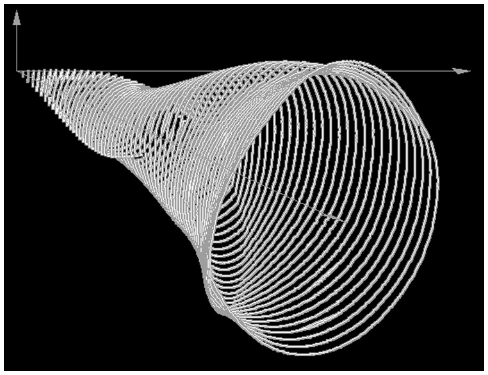


**Figure 18** A 3-D image of the solution to a differential inclusion.

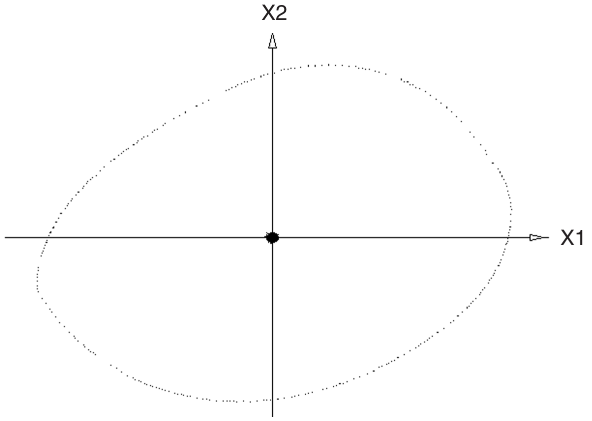


**Figure 19** A time section of the reachable set of a differential inclusion.

tour) are the end points (400 in total) of the trajectories generated by the DI solver. This demonstrates just how useless the simple shooting method is for solving DIs.

## X. DISTRIBUTED PARAMETER SYSTEMS

As stated before, the set of all possible states of a distributed parameter system (DPS) is greater than the set of all real numbers. In fact, the state of a DPS is given as one or more functions of the position in a space (continuum). A classic example of such a system is a vibrating string or the temperature distribution in a piece of metal. The name is due to the fact that the parameters of the system cannot be concentrated in discrete points and are instead distributed over a given region. The basic modeling tool for DPSs is the partial differential equation. Normally, a PDE model establishes a relationship between the derivatives of the dependent variable (e.g., the temperature) with respect to the model time and the space position coordinates.

We do not discuss here the numerical methods for utilizing PDEs. From the computational point of view the problem of solving a PDE is difficult. Consider, for example, the following heat transfer equation:

$$\frac{\partial T}{\partial t} = \frac{k}{\rho c}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) \qquad (15)$$

where $T$ is the temperature, $x$, $y$, $z$ are the 3-D coordinates, $t$ is the time, $\rho$ is the material density, $c$ is the specific heat, and $k$ is the thermal conductivity. This is a common situation in simulation problems that involve PDEs. One or more dependent variables propagate forward in time. The highest time derivatives of these variables should be put alone on the equation's right-hand side.

A common way to find the solution to this and similar equations is to discretize both time and space, and replace the derivatives with their discrete approximations. As the time derivative appears only at the left-hand side of Eq. (15), we can calculate the "slope" (the rate of change) of the temperature at each (discrete) point of the 3-D space. Then applying Euler-like procedure we can calculate the solution at the next time step, advance the model time, and so on.

Obviously the number of points to calculate grows with the fourth power of the required time–space resolution. But that is not the only difficulty. Note that the right-hand side of (15) is a *differential operator.* A well-known fact (at least for mathematicians) is that such operators are not bounded. This is the other, profound cause of our numerical difficulties. The *stability* of the numerical algorithm can be easily lost and we must use sophisticated methods to avoid that. This is why the simulation software for DPSs (e.g., the famous ANSYS package) is so expensive and why we need hours of computer time to simulate PDE models.

A well-known approach to the DPSs simulation is the *finite element method* (FEM). In fact, a FEM solves a PDE using a user-defined grid of spatial elements with finite dimensions. Each finite element approximates the behavior of the corresponding real element. However, the equation that governs the state of a single finite element is not a PDE. In the global scale the set of (thousands) of finite elements approaches the numerical solution to the corresponding PDE. The advantage of FEM is that the user can define an irregular net of elements, where the regions that require greater accuracy are covered by smaller elements. The challenge for the future is to develop efficient methods for such types of models. This may help with problems such as weather simulation and predictions, human body simulation, plasma dynamics, and new material simulation.

The DPS problems that can be solved on a PC are discretized models, which can reach up to hundreds of thousand of finite elements or regular grid points. Figure 20 shows an example of a solution to the gas dynamics equation in a duct with an obstacle. The image was generated by the Fluids32 program (consult http://www.raczynski.com/pn/fluids.htm). The figure shows a vertical section of the duct. The program uses a uniform 3-D grid of points (about 150,000 in total) and finds a steady flow solution provided it exists. Note that this is equivalent to solving 150,000 nonlinear algebraic equations for five variables (3-D velocity, temperature, and pressure).

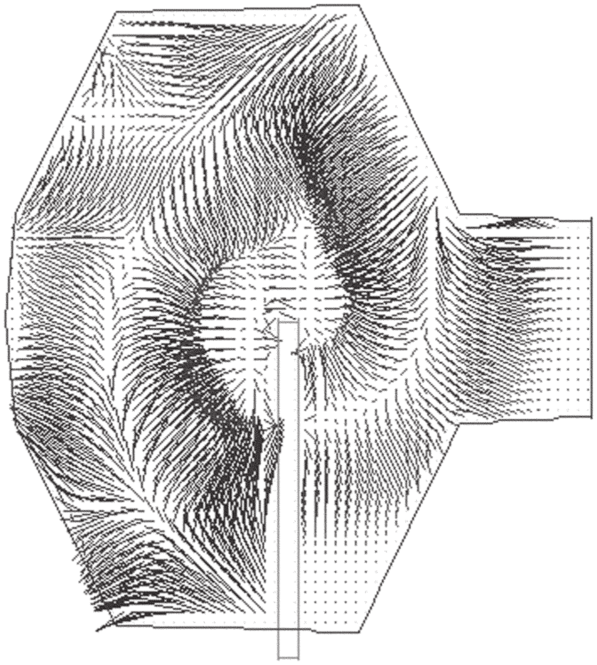


**Figure 20** Simulation of shock waves in a gas flow (Fluids32 program).

## XI. SYSTEM DYNAMICS

System dynamics is a computer-aided approach to policy analysis and design. With origins in servomechanisms engineering and management, the approach uses a perspective based on information feedback and mutual or recursive causality to understand the dynamics of complex physical, biological, and social systems.

What SD attempts to do is understand the basic structure of a system, and thus understand the behavior it can produce. SD takes advantage of the fact that a computer model can be of much greater complexity and carry out more simultaneous calculations than can the human mind.

During the 1970s to 2000s, SD has been applied broadly in such areas as environmental change, economic development, social unrest, urban decay, psychology, and physiology. Jay W. Forrester (http://web.mit.edu/sloan/www/faculty/forrester.html), the creator of system dynamics, defines it as follows:

> Unlike other scientists, who study the world by breaking it up into smaller and smaller pieces, system dynamicists look at things as a whole. The central concept to system dynamics is understanding how all the objects in a system interact with one another. A system can be anything from a steam engine, to a bank account, to a basketball team. The objects and people in a system interact through "feedback" loops, where a change in one variable affects other variables over time, which in turn affects the original variable, and so on.

The seminal book by Forrester, *Industrial Dynamics,* is still a significant statement of philosophy and methodology in the field. Since its publication, the span of applications has grown extensively and now encompasses work in corporate planning and policy design, public management and policy, biological and medical modeling, energy and the environment, theory development in the natural and social sciences, and dynamic decision making.

The STELLA simulation package is one of the most well-known implementations of the concepts of system dynamics. A STELLA model is defined in terms of *levels* and *flows.* Levels are represented graphically as boxes and flow as directed links (arrows). The levels are integrals of total flows that affect them. The flows can depend on the levels through algebraic functions, depicted as circles. Levels can represent, for example, the capital level of a corporation, the working power, or cash. The flows can be cash flow, working power flow, material or machinery flow, etc.

Figure 21 shows a possible fragment of a STELLA scheme. Here, $A$ and $B$ are levels, $f1$, $f2$, and $f3$ are flows, and $x$, $y$, and $z$ are external signals. The flow $f1$ depends on levels $A$ and $B$ through the function $F$. STELLA, as well as DYNAMO, PowerSim, and several other similar tools, has been widely used for simulation of dynamic systems in particular in soft systems simulation.

A huge number of works and publications are available on the applications of system dynamics models. The importance of this methodology lies in its accessibility for engineers, sociologists, decision makers, and businesspeople, rather than in the validity of system dynamics models. The false conviction that everything that changes can be simulated using continuous system dynamics sometimes results in strange and even wrong models. However, the model validity and accuracy is not always the most important aspect of the simulation task. When simulating social systems, it is important to understand the system structure and its internal interactions rather than to produce exact projections of systems trajectories. Such a modeling effort can be useful in other "dimensions" of modeling. A good example of this is a model of Shakespeare's *Hamlet* presented at one of the system dynamics conferences. A paper entitled "Implementation of Stella to illustrate *Hamlet,*" by Pamela Lee Hopkins, Desert View High School, Tucson, Arizona, was presented at the 1984 Systems Dynamics Conference in Boston. (The text, however, is not included in the proceed-

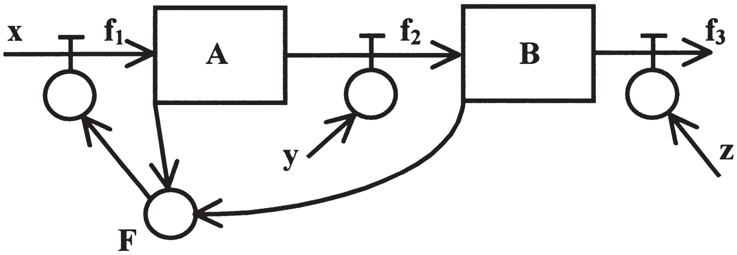


**Figure 21** Fragment of a STELLA scheme.

ings.) The model is designed to expose the effect plot events have on Hamlet's willingness to kill Claudius. This permits the examination of the impact of each event as it occurs and as Hamlet continues to contemplate the situation. The model variables are, among others the motivation to avenge, new evidence, old evidence, motivation per death, aging, opportunity to act, etc. Obviously the author's objective was not to repeat the sequence of events of the original Hamlet, or to write new Shakespeare-style dramas. As stated in the article:

> The model forces the students to sequence events, thereby improving their understanding of the dramatic action of the play. The model provides a meaningful language, in the form of numbers, for students to discuss psychological concepts, emotions, and the resulting behaviors. This promotes student involvement in discussion of the difficult concepts of a critically acclaimed piece of literature.

## XII. GALACTIC SIMULATIONS AND THE *N*-BODY PROBLEM

Recall that the problem of integrating the equations of movement of $N$ bodies in space is extremely difficult for even just three bodies, and no analytic solution can be found for more than four bodies moving due to their reciprocal gravitational forces. A huge literature exists along with many numerical methods for solving the $N$-body problem. We do not pretend to give any exhaustive discussion here; only some recent results are mentioned.

To solve the $N$-body problem, we must integrate a system of second-order ordinary differential equations of the form

$$m_i\, dx_i^2/dt^2 = f_i(x) \qquad \text{for } i = 1, 2, ..., N$$

where $f_i(x)$ is the total force, $x = (x_1, x_2, x_3, ..., x_N)$, $x_i \in R^3$. Here $f_i$ is the vectorial sum of all the forces the body $i$ receives from the other bodies and $m_i$ is the mass of the $i$-th body. This force is proportional to $1/d^2{}_{ij}$, where $d_{ij}$ is the distance from body $i$ to body $j$. Thus, the number of forces grows with $N$ square, and for big $N$ the computer time may be unacceptable. Another difficulty arises from the nonlinearity of the problem. In fact, looking at the preceding equation, we can see that the space where the bodies move is full of singularities, each body position being a singularity, where the force tends to infinity as $d$ approaches zero. This implies more computational problems and may result in numerical instability. Fortunately, several decomposition algorithms are available that can reduce the computational time.

Consider the large-scale structure (LSS) formation dynamics problem. When the number of particles is big, a common approach is to use the Fourier transform. A regular grid is defined in the region, and then the Fourier transform is used to speed the calculation of the potential field. This field is the result of a convolution of the point distribution and the Green's function for the force. However, such methods have some disadvantages and lack flexibility, in particular when the particles form many local clusters leaving other regions nearly empty. Some improvements of the method were proposed and work well with great numbers of points. At any rate, this approach always needs a grid definition.

Another approach is to decompose the problem using a binary tree, which has nodes that represent the body clusters. In the binary tree, the bodies are sorted and a tree is formed, where the leaves represent the bodies. Every body (leave) is connected to its nearest neighbor and forms a cluster (next level of the tree). Thus, after such sorting, it is relatively easy to decompose the system in clusters and to assign the corresponding movement tasks to the processors of a parallel machine. As a result, machines like CRAYs and similar supercomputers can be used quite efficiently. Actually, the number of bodies moving simultaneously in the most advanced galactic and particle simulation can reach more than 200 millions. This permits us to simulate the dynamics of huge star clusters and possibly approximate galactic dynamics. Other applications are molecular simulations and simulations of gas dynamics.

The number of bodies involved in galactic simulations in not necessarily the most important issue. Surprisingly, simulating about 1000 bodies we also can obtain interesting results. Such simulation can even be implemented on a PC. For example, it can be shown that the rotational movement, disk formation, and appearance of spiral shapes is a natural consequence of the Newtonian dynamics applied to a cloud of particles. Note that the same methodology can be applied to the simulation of particle movement in small, molecular scale. If we replace the gravitational force with the Van der Waals forces, then we obtain a model of a particle in a gas. With other definition of the interactions, the model might describe, for example, a granular media, the dynamics of which is highly difficult to analyze analytically. A PC can be used in simulations of this type.

## SEE ALSO THE FOLLOWING ARTICLES

Discrete Event Simulation • Model Building Process • Monte Carlo Simulation • Optimization Models • Simulation Languages • Software Process Simulation

## BIBLIOGRAPHY

Ames, W. F. (1977). *Numerical methods for partial differential equations,* 2nd ed. New York: Academic Press.

Aubin, J. P., and Cellina, A. (1984). *Differential inclusions,* New York: Springer-Verlag.

Bader, G., and Deufhard, P. (1983). *Numerishe mathematik,* Vol. 41, 373–398.

Bargiela, A. (1998). Uncertainty—a key to better understanding of systems. *Proc. 10th European Simulation Symposium,* 11–29.

Barnes, J., and Hut, P. (1986). A Hierarchical $O(N \log N)$ force calculation algorithm. *Nature,* Vol. 324, 446.

Binney, J., and Tremaine, D. (1987). *Galactic dynamics.* Princeton, NJ: Princeton University Press.

Cellier, F. (April 1992). Hierarchical non-linear bond graphs: A unified methodology for modeling complex physical systems, *Simulation.* Vol. 58, No. 9, 230–248.

Cellier, F. E., and Elquist, H. (1993). Automated formula manipulation in object-oriented continuous-system modeling, *IEEE Control Systems,* Vol. 13, No. 2, 28–38.

Dalquist, G., and Bjork, A. (1974). *Numerical methods.* Englewood Cliffs, NJ: Prentice Hall.

Dorf, R. (1992). *Modern control systems.* Reading, MA: Addison-Wesley, Sec. 2.7.

Gear, C. W. (1971). *Numerical initial value problems in ordinary differential equations,* Englewood Cliffs, NJ: Prentice Hall, Chap. 9.

Hernquist, L. (1998). Hierarchical N-body methods. *Computer Physics Communications,* Vol. 48, 107.

Kandrup, H. E., and Mahon, M. E. (1994). Chaos and noise in galactic dynamics. *Annals of the New York Academy of Sciences,* Vol. 751, 93–111.

Kaps, P., and Rentrop, P. (1979). *Numerishe mathematik,* Vol. 33, 55–68.

Lambert, J. D. (1981). Safe point methods for separably stiff systems of ordinary differential equations. *SIAM Journal on Numerical Analysis,* Vol. 18, 83–101.

Lee, E. B., and Markus, L. (1967). *Foundations of optimal control theory.* New York: Wiley.

Marchaud, A. (1934). Sur les champs de demi-cones et les équations differielles du premier ordre, *Bull. Soc. Math. France,* Vol. 62, 1–38.

Mills, P., Nyland, L., Prins, J., and Reif, J. (1992). Prototyping high-performance parallel computing applications in proteus, *Proc. 1992 DARPA Software Technology Conference,* April 28–30, 1992, Los Angeles. Arlington, VA: Meridian, 433–442.

Nyland, L. S., and Prins, J. F. (1992). Prototyping parallel algorithms. *Proc. DAGS/PC Symposium,* June 1992. Hanover, NH: Dartmouth College, 31–39.

Plis, A. (1962). Trajectories and quasitrajectories of an orientor field. *Bull. Acad. Polon. Sci. Ser. Math. Astronom. Phys.,* Vol. 10, 529–531.

Raczynski, S. (1974). On the determination of the reachable sets and optimal control by the random method, *Proc. Symposium IFAC on Optimization Methods,* Varna, Bulgaria, 1974.

Raczynski, S. (1984). On some generalization of "bang-bang" control. *Journal of Mathematical Analysis and Applications.* Vol. 98, No. 1, 282–295.

Raczynski, S. (1986). Some remarks on nonconvex optimal control. *Journal of Mathematical Analysis and Applications.* Vol. 118, No. 1, 24–37.

Raczynski, S. (1996). Differential inclusions in system simulation. *Trans. Society for Computer Simulation,* Vol. 13, No.1, 47–54.

Raczynski, S. (1996). When systems dynamics ODE models fail, *Simulation,* Vol. 67, No. 5, 343–349.

Raczynski, S. (November 1997). Simulating the dynamics of granular media—The oscillon phenomenon, *Computer Modeling and Simulation in Engineering,* Vol. 2, No. 4, 449–454.

Raczynski, S. (March 2000). Creating galaxies on a PC, *Simulation,* Vol 74, No. 3, 161–166.

Ralston, A., and Rabinowitz, P. (1978). *A first course in numerical analysis,* 2nd ed. New York: McGraw-Hill.

Rice, J. R. (1983). *Numerical methods, software and analysis.* New York: McGraw-Hill.

Turowicz, A. (1963). Sur les trajectoires et quasitrajectoires des systémes de commande nonlinéaires, *Bull. Acad. Polon. Sci. Ser. Math. Astronom. Phys.,* Vol. 10, 529–531.

Wazewski, T. (1962). Sur les systemes de commande non lineaires dont le contredomaine de commande n'est pas forcement convexe, *Bull. Acad. Polon. Sci. Ser. Math. Astronom. Phys.* Vol. 10, No.1.

Zaremba, S. K. (1936). Sur les équations au paratingent. *Bull. Sci. Math.,* No. 60, 139–160.

# Control and Auditing

**Mary S. Doucet and Thomas A. Doucet**
*California State University, Bakersfield*

## GLOSSARY

**application controls** Controls which affect only specific applications and include the steps within application software and the manual procedures designed to control the processing of various types of transactions.

**fiduciary objectives of internal control** Objectives which support an organization's business processes (include reliability, compliance, effectiveness, and efficiency).

**general controls** Controls designed to prevent or detect errors or irregularities on a more extensive basis than application controls. They include all controls over the information processing facility, access controls, environmental controls, business continuity planning controls, systems and program software acquisition and maintenance controls, network controls, and data communication controls.

**information security** "The protection of the interests of those relying on information, and the information systems and communications that deliver the information, from harm resulting from failures of availability, confidentiality, and integrity" (IFAC 1998, Executive Summary).

**information systems audit** The process of determining whether a computer system safeguards assets, maintains data integrity, allows organization goals to be achieved effectively, and uses resources efficiently.

**internal control** A process designed to provide reasonable assurance regarding the achievement of objectives in the following categories: (1) effectiveness and efficiency of operations, (2) reliability of financial reporting, and (3) compliance with applicable laws and regulations.

**THE DIGITAL WORLD** has increased the critical importance of the control and audit of information systems to the survival of organizations. The International Federation of Accountants (IFAC) suggests that this increased importance arises from "the increasing dependence on information and the systems and communications that deliver the information; the scale and cost of the current and future investments in information; and the potential for technologies to dramatically change organizations and business practices, create new opportunities, and reduce costs." Therefore, control and audit of information systems must be considered in the context of managing the organization and providing assurance to users of information. Control of information systems is not just a technology issue; it is a management issue.

A primary facet of the control of information systems is to ensure information security. The objective of information security is "the protection of the interests of those relying on information, and the information systems and communications that deliver the information, from harm resulting from failures of availability, confidentiality, and integrity" (IFAC). To protect against harm resulting from failures of availability, confidentiality, and integrity, and to ensure other aspects of internal control, an organization has to assess the risks to information systems. These risks must be considered within the context of the potential threats to the organization, the probability of those threats occurring, and the impact on the organization if the threats are realized.

Additionally, the audit of information systems must be considered within the context of the purpose of the audit. The objectives and intensity of the evaluation of internal control will vary depending on the type of audit service being provided. If the auditor is issuing assurance on internal control (or on just the information security aspects of internal control), the scope and intensity of the testing of the control system will be much greater than if the auditor is performing a financial statement audit. In the first case auditors are reporting primarily on the internal control system, which requires a much more rigorous evaluation than the second case for which auditors need only sufficient assurance to rely on the information that comprises the financial statements.

## I. BROAD OBJECTIVES OF INTERNAL CONTROL SYSTEMS

While there are many definitions of internal control, one definition that appears to have gained widespread support is the definition provided by the Committee of Sponsoring Organizations of the Treadway Commission (COSO). COSO defines internal control as follows: Internal control is a process, effected by an entity's board of directors, management, and other personnel, designed to provide reasonable assurance regarding the achievement of objectives in the following categories: (1) effectiveness and efficiency of operations, (2) reliability of financial reporting, and (3) compliance with applicable laws and regulations.

Several fundamental concepts are inherent in this definition. Perhaps the most important concept is that internal control cannot provide absolute assurance that an organization can meet its objectives. Internal control, however, should provide reasonable assurance that an organization's objectives in the separate but overlapping categories will be met. Another fundamental concept inherent in this definition is that internal control is a process (a means to an end, not an end itself) carried out by people at all levels in the organization. Finally, there is the fundamental concept that the effectiveness of the internal control system depends primarily on the people who implement it.

A number of broad information systems control objectives can be derived from the COSO definition of internal control. While each organization may differ in the emphasis it places on these internal control objectives, the following broad objectives (the term "requirements" is used by the COBIT Framework) are common to all organizations (Information Systems Audit and Control Foundation): (1) availability, (2) integrity, (3) confidentiality, (4) reliability, (5) compliance, (6) effectiveness, and (7) efficiency. The size, structure, and other characteristics of an organization will have an impact on the emphasis placed on each of these objectives. In the COBIT Framework, these seven objectives of information are categorized as information security objectives (availability, integrity, and confidentiality) and fiduciary objectives (reliability, compliance, effectiveness, and efficiency). While the fiduciary objectives of information systems are important to all organizations, an organization cannot achieve these fiduciary objectives without also achieving the more critical information security objectives. Therefore, organizations should design internal control systems to meet both the information security and fiduciary objectives. Each of these broad objectives is discussed briefly below.

### A. Information Security Objectives of Internal Control Systems

Because of their importance in providing support for an organization's overall objectives, the information security objectives are discussed first. The security objectives of information provide the underlying support for the internal control of information systems. These security objectives are discussed in their order of priority for many business managers.

#### 1. Availability

The objective of availability of information demands that information be available to support the organization's processes and its interactions with other organizations and with customers. Additionally, the availability of appropriate information is necessary to support the safeguarding of an organization's resources and capabilities. Availability of information is essential to the operation of any organization; many organizations are paralyzed when information is unavailable. Availability of information provides the foundation for a quality organization.

#### 2. Integrity

Integrity of information implies that it is valid and complete. If the available information lacks integrity, the organization's decisions will be based on information that lacks credibility and, thus, the probability of faulty decisions is increased. Thus, integrity requires that information be protected against unauthorized alteration. If information integrity is compromised, the result can be as devastating as unavailable information and some might argue that it can be more devastating.

### 3. Confidentiality

Confidentiality requires that sensitive information should only be accessed by those with authorization (such authorization should only be provided on a "need to know, need to do" basis). Maintaining confidentiality is critical to an organization's ongoing success. When confidentiality is breached, an organization often pays with its reputation, and customers, suppliers, and business partners may decide to take their business elsewhere.

## B. Fiduciary Objectives of Internal Control Systems

The fiduciary objectives described below are derived from the COSO Framework. These fiduciary objectives are intended to support an organization's business processes.

### 1. Reliability

The reliability of information objective necessitates that appropriate information be provided to management to fulfill its fiduciary objectives. As stated in the *Statement of Financial Accounting Concepts No. 1,* reliable information is verifiable, represents what it purports to represent, and is neutral with respect to parties it affects (Financial Accounting Standards Board, FASB). Reliable information allows management to operate the organization and fulfill the organization's financial and compliance reporting responsibilities.

### 2. Compliance

The fiduciary objectives of information require information to be provided that allows an organization to comply with laws, regulations, and contractual relationships. In addition to providing the information, the use of information itself must also comply with laws, regulations, and contractual relationships.

### 3. Effectiveness

The effectiveness objective means that to be useful to an organization, information must support the effective operations of that organization. In order to support the effective operations of an organization, information must be relevant to the organization's business processes and must be provided in a timely manner. Additionally, the integrity of information, discussed previously, is essential for information to support the effective operations of the organization's business processes.

### 4. Efficiency

The usefulness of information is increased by the provision of information through the most productive and economical use of resources. Providing information in the most efficient manner enhances the quality of any organization.

## C. Meeting Security and Fiduciary Internal Control Objectives

To meet the fiduciary and information security objectives, organizations must develop internal control policies and procedures that address the security objectives while enhancing the fiduciary objectives of internal control of information systems. The next section provides a discussion of the internal control of information systems.

# II. INTERNAL CONTROL OF INFORMATION SYSTEMS

Historically, internal controls have often been viewed by many managers as necessary burdens. Controls were often seen as a hindrance to operating effectiveness and efficiency. Today internal controls are often viewed as an integral part of the management processes of planning, executing, and monitoring business activities. In many organizations separating out the control aspects of a process is difficult because controls are so highly integrated with the process. This integration can provide a challenge to the ongoing monitoring of internal controls and to the separate periodic evaluations of the internal control system. However, as will be discussed in the next section, this ongoing monitoring and separate periodic evaluation are integral parts of a good system of internal control of information systems.

## A. Current Frameworks for the Control and Audit of Information Systems

The control of information systems should be addressed within the broader context of an entity's organization-wide control system. The objective of the organization-wide control system is the safeguarding of all of an organization's assets and capabilities. One such organization-wide control model is the COSO control model. In the United States the COSO control model is widely accepted as providing businesses with an ideal control framework. While this organization-wide control model has a broader focus than the control of information systems, it has influenced frameworks that deal more directly with the control of information systems.

There are two major control models that focus primarily on the control of information systems and/or its related technology: (1) The *Systems Auditability and Control* (SAC) *Report,* developed by the Institute of Internal Auditors, which provides control principles and techniques related to specific technologies; and (2) the *Control Objectives for Information and related Technologies* (COBIT) *Report,* developed by the Information Systems Audit and Control Foundation, which provides a conceptual framework of control objectives designed to meet specific business goals.

In the following three sections, an overview of the COSO Report, the SAC Report, and the COBIT Report are provided. These frameworks are complementary with different as well as overlapping audiences. In fact, the COBIT Report (the most recent of the three) incorporates concepts from both the COSO Report and the SAC Report.

These three frameworks have been instrumental in laying the groundwork for discussion of the control and audit of information systems. Following a discussion of these three frameworks is a discussion of types of information systems controls. This is followed by a discussion of the impact of recent technologies on the control of information systems, and the potential future impact of technology on the control of information systems.

## 1. COSO Framework

The internal control of information systems must be discussed within the broader framework of an organization-wide control system. In fact, one of the key issues that organization managements are concerned with is how to integrate the internal control system with the organization's overall objectives. In 1992, COSO published *Internal Control—Integrated Framework.* The purpose of the COSO Framework was to integrate various internal control concepts into a framework that would help managers to better control their organizations' activities. This framework provides a useful structure from which to design and implement an organization-wide control system.

The COSO Framework suggests that internal control consists of five interrelated components: (1) control environment, (2) risk assessment, (3) control activities, (4) information and communication, and (5) monitoring. The control activities and the information and communication components of this model relate more directly to internal control of information systems; however, to better understand the COSO internal control model it is beneficial to look at all the components.

The first component, the control environment, is the foundation for all of the other components in that it provides the climate in which the people (the core of any organization) perform their tasks and carry out their control responsibilities. The control environment includes the "tone at the top" and reflects the corporate culture inherent in the ethical values adopted by and the integrity of management. An important aspect of the control environment that has a direct impact on control of information systems is that the control environment is influenced by the degree to which employees recognize that they will be held accountable.

The second component, risk assessment, is the process by which management identifies and analyzes the risks that might prevent the achievement of organizational objectives and, thus, is a vital component of any organization's internal control. An organization's ability to survive, compete, maintain its financial strength, maintain the quality of its deliverables, and maintain its positive image are all affected by risks to the attainment of these objectives. Therefore, before an organization can assess risks to meeting its goals, it must first develop operational and fiduciary (reporting and compliance) objectives at all levels in the organization.

The risk assessment process involves identifying factors that contribute to or increase risks as well as performing a risk analysis to estimate the significance of the risk and assessing the probability of the risk occurring. Risks that do not have a significant impact on the organization, or that have a low probability of occurring, should not receive as much attention as those that have a high probability and/or a significant impact on the organization. Once the risk assessment process has identified the potential risks that need to be addressed, management needs to implement policies and procedures that are necessary to address the risks. In essence, the result of risk assessment is an evaluation or reevaluation of the control activities of the organization.

The policies and procedures that management implements constitute the control activities identified as the third component of the COSO Framework. These control activities are designed to help ensure that management directives are satisfied. The information systems of most organizations include both manual and computerized elements. The control activities component, as discussed in the COSO Framework, assumes this environment. It is this aspect of the COSO Framework that has been almost universally adopted as a means for discussing the control of information systems.

The COSO Framework categorizes information systems control activities by scope: (1) general controls (also referred to as information technology controls or general computer controls) and (2) application controls. General controls are broader in nature and are designed to prevent or detect errors or irregularities on a more extensive basis than application controls. They include all controls over the information processing facility, access controls, environmental controls, business continuity planning controls, systems and program software acquisition and maintenance controls, network controls, and data communication controls. The information systems environment in large and complex organizations can be thought of as consisting of a group of independent segments to which general controls must be applied. With distributed processing and distributed information systems becoming the norm for these organizations, it is important to recognize that general controls may not be implemented uniformly across all segments in the organization. Application controls affect only specific applications and include the steps within application software and the manual procedures designed to control the processing of various types of transactions. These controls allow for the secure capture and communication of information.

Information and communication comprise the fourth component of the COSO Framework. Information and communication, along with control activities, are perhaps the most critical components of the COSO Framework for the control of information systems. Information, as used in the COSO Framework, refers to information systems that identify, capture, process, and report information. It is important, and increasingly so, that an organization's information system generates high quality information (information that meets the security and fiduciary objectives described earlier) in order for management to make the appropriate decisions in controlling and managing an organization's activities.

Communication means that an organization should communicate to all personnel the importance of the internal control system and their responsibilities regarding internal control. In order for an internal control system to work, personnel must be given clear guidance as to what their responsibilities are, must be provided with relevant information to perform their duties, and must be given a mechanism for communicating significant information upstream in their organization.

Monitoring is the fifth component of the COSO Framework and involves assessing the system's performance over time either through ongoing monitoring activities, separate evaluations, or a combination of the two. All internal control systems need to be monitored. Monitoring helps an organization to determine whether its internal control system continues to be effective. When deficiencies are addressed in a timely manner through the monitoring process, the operating effectiveness of the internal control system is ensured.

The COSO Framework broadly defines the components of an organization's internal control system. While the COSO Report is intended to provide a broad framework to help managers to better control their organizations' activities, the SAC Report discussed below is intended for a wider audience, provides more extensive control principles and techniques, and addresses specific concerns with the widespread use of current information technologies. While its purpose differs somewhat from that of the COSO Framework, the control principles and techniques described in the SAC Report are consistent with and complementary to the framework described in COSO.

## 2. SAC Framework

The *Systems Auditability and Control Report* lists the results of a research project conducted by the Institute of Internal Auditors and provides guidance on the control and audit of information resources. One of the major findings of the research project was that the most important challenge facing management is the ability "to integrate the planning, design, and implementation of complex application systems with the strategy of the organization." As we begin the 21st century this challenge continues to confront organizations of all sizes.

Data security and contingency planning were also found to be key control concerns for managers. The reasons for these concerns include: (1) the heavy reliance organizations place on their information systems for critical operations; (2) the use of applications (such as electronic data interchange, EDI) that allow third parties to exchange structured transaction information; (3) greater on-line access to an organization's information systems; and (4) the inadequate consideration of security and recovery controls by end users. These concerns have not diminished since the initial SAC Report and, in fact, have increased especially with the advent of greater internet connectivity, more telecommuting capabilities, electronic commerce, and an increasing emphasis on client/server architecture.

The SAC Report focuses on the business perspective of information technology and the accompanying risks associated with planning, implementing, and using this technology. The objectives of the SAC Report include: providing senior management with necessary guidance to establish priorities for the implementation of information technology controls and providing internal audit and information systems practitioners with guidelines and technical reference materials to facilitate the implementation and verification of appropriate controls. The emphasis in the SAC Report is on control principles and techniques, rather than specific implementations. Thus, these control principles and techniques should be applicable to emerging technologies.

The SAC Report contends that regardless of the significant changes experienced in information technologies and their impact on control and audit, "the basic philosophy of a controlled environment has not changed" (SAC). Information and process integrity are still primary concerns for most organizations. Management is still responsible for identifying risks, designing internal control systems that mitigate the identified risks, and ensuring that the control systems are operating effectively. Auditors are still responsible for determining that internal control systems exist, are adequate and effective, and that reliance can be placed on them.

The SAC Framework emphasizes the complex relationship between risks and controls. Some controls may address several risks, while some risks may need several controls to be mitigated. This is why the SAC Framework recommends that risks and controls be evaluated in the context of the organization-wide internal control system. Taking this organization-wide view has led to a migration of the control focus from the applications environment to one which focuses on a balance of application and general controls. General controls are more extensive (can affect multiple applications) than application controls. Thus, the current trend in information technology is toward placing additional attention on general controls. The audit implications of the migration of controls include more emphasis being placed on the periodic general control reviews and the audit of technology components.

The SAC Framework suggests that the key elements of an internal control system are the control environment, manual and automated systems, and control procedures. Consistent with the COSO Framework, the SAC Framework indicates that a sound control environment provides the foundation of the internal control system and contributes to its reliability. Also consistent with the COSO Framework is the classification of control procedures by scope: general information systems controls and specific application controls.

The SAC Report describes the audit tools and techniques relevant to auditing information technology controls. It emphasizes that manual auditing techniques are no longer sufficient since much of the data required for auditing are in an electronic format and the volume and complexity are greater than can be handled manually. As technological environments become increasingly complex, auditors themselves will need to apply innovative uses of information technology to audit these environments.

The SAC Report also addresses an organization's core information systems environment and addresses the risks, controls, and audit considerations of the telecommunications technologies that enable the various components of a business operation to "talk" with each other. The SAC Report does this within the context of specific technologies; however, as mentioned previously, the control principles and techniques are broad enough to be applied to emerging technologies.

## 3. COBIT Framework

The *Control Objectives for Information and Related Technology* Framework provides a comprehensive and usable control model over information technology in support of business processes. While the SAC Report provides control principles and techniques related to specific technologies, the COBIT model provides a more conceptual view of the control objectives related to information technology processes. The primary goal of the COBIT model is to provide "clear policies and good practices for security and control in IT (information technology)." To enable the COBIT model to be applicable over time, it defines the IT objectives in a generic way so that the objectives are not dependent on any particular technical platform (COBIT Framework). COBIT recognizes that some special technical environments may need separate coverage for control objectives. For example ISACA has published *Control Objectives for Net Centric*

*Technology* to address the specific technical issues for this environment.

The COBIT model recognizes that information is "the result of the combined application of IT-related resources that need to be managed by IT processes." The COBIT model starts from the following premise: "In order to provide the information that the organization needs to achieve its objectives, IT resources need to be managed by a set of naturally grouped processes." The COBIT model consists of 34 processes supported by high-level IT control objectives. The control objectives are grouped into four domains: (1) planning and organization, (2) acquisition and implementation, (3) delivery and support, and (4) monitoring (see Table I for a listing of the 34 processes as they relate to these domains). Planning and organization encompasses the achievement of business objectives via the best use of IT resources. Acquisition and implementation deals with identifying the IT solutions required to meet business objectives, developing or acquiring the necessary IT resources, and implementing and integrating these solutions into the business process. Delivery and support concerns itself with the actual delivery of required services. This domain includes the processing of data by application systems. Monitoring deals with the monitoring of IT processes for quality and control compliance. The COBIT model ties the control objectives of information (availability, integrity, confidentiality, reliability, compliance, effectiveness, and

**Table I** **Domains and Processes**

| COBIT Framework | |
|---|---|
| **Domains:** | **Processes:** |
| Planning and organization | PO1 Define a strategic IT plan<br>PO2 Define the information architecture<br>PO3 Determine the technological direction<br>PO4 Define the IT organization and relationships<br>PO5 Manage the IT investment<br>PO6 Communicate management aims and direction<br>PO7 Manage human resources<br>PO8 Ensure compliance with external requirements<br>PO9 Assess risks<br>PO10 Manage projects<br>PO11 Manage quality |
| Acquisition and implementation | AI1 Identify solutions<br>AI2 Acquire and maintain application software<br>AI3 Acquire and maintain technology architecture<br>AI4 Develop and maintain IT procedures<br>AI5 Install and accredit systems<br>AI6 Manage changes |
| Delivery and support | DS1 Define service levels<br>DS2 Manage third-party services<br>DS3 Manage performance and capacity<br>DS4 Ensure continuous service<br>DS5 Ensure systems security<br>DS6 Identify and attribute costs<br>DS7 Educate and train users<br>DS8 Assist and advise IT customers<br>DS9 Manage the configuration<br>DS10 Manage problems and incidents<br>DS11 Manage data<br>DS12 Manage facilities<br>DS13 Manage operations |
| Monitoring | M1 Monitor the processes<br>M2 Assess internal control adequacy<br>M3 Obtain independent assurance<br>M4 Provide for independent audit |

efficiency) and the IT resources that deliver the information (people, application systems, technology, facilities, and data) with its four domains and their related control objectives.

An example should clarify how the model may be used by business process owners, auditors, and users. First, the COBIT model enumerates the high-level control objectives by tying each to one of the 34 IT processes. Each high-level control objective is presented as a statement in the following format:

**Control over the IT process of** name of one of the 34 IT processes (see Table I)

- **that satisfies the business requirement of** specific business requirement
- **is enabled by** statement of control objectives
- **and takes into consideration** specific control practices.

Additionally, the COBIT model clearly presents which information systems control objectives are primarily and secondarily supported by the high-level control objective. The model also specifies which IT resources are managed by the process under consideration, not just which IT resources are affected by the process.

## B. Types of Information Systems Controls

There are two well-established ways to classify controls: (1) on the basis of their function and (2) on the basis of their scope. Classifying controls by function distinguishes whether the controls are preventive, detective, or corrective in nature. Preventive controls are most desirable because they are designed to prevent an error, omission, or unauthorized access from occurring. Preventive controls can be as simple as having clear instructions for data entry or as complex as firewall security. Detective controls detect and report when an error, omission, or unauthorized access occurs. Detective controls include input programs which identify data incorrectly entered and reports on failed access attempts. Corrective controls are designed to correct errors and omissions, and terminate unauthorized access once detected. Corrective controls include resubmitting data which was entered incorrectly and programs designed to correct electronically transmitted data corrupted by line noise.

The classification of controls on the basis of their scope (i.e., general controls versus application controls) is widely accepted and is the primary means by which information systems controls are classified in this discussion. General controls, as previously defined, include all controls over the information processing facility and its related functions. Examples of general controls include the use of log-on IDs and passwords to authenticate users, locked doors, data encryption, and business continuity planning. Application controls include the steps within application software and the manual procedures designed to control the processing of various types of transactions. Examples of application controls include data validation controls, control totals, and valid report distribution lists.

### 1. General Controls

General controls include: sound personnel and communication practices; information systems organizational controls, access controls (both logical and physical); environmental controls; business continuity planning controls; controls over acquisition, development, and maintenance of systems and programs; network controls; and data communication controls.

#### a. Sound Personnel and Communication Practices

Sound personnel and communication practices include hiring practices. Good hiring practices match skills to positions, and ensure the good character of those who fill sensitive positions such as the network administrator, security administrator, and database administrator. They also include the continuing development of current employees' professional and technical skills. In addition, practices that promote security awareness via appropriate documentation of the internal control system and communicating security concerns to all personnel are an integral part of an effective general control environment.

#### b. Information Systems Organizational Controls

Information systems management is responsible for the day-to-day operations of the information processing facility, which includes ensuring that application systems can accomplish their work and that development staff can develop, implement, and maintain application systems. In addition to responsibility for day-to-day operations, information systems management is responsible for information systems planning, strategic planning, and information systems project management. Information systems management is responsible for control over (1) data entry when not performed in user departments; (2) library functions such as recording, issuing, receiving, and safeguarding all program and data files maintained on computer disks and tapes; and (3) data control group functions such as the collection, conversion, and control of input and the balancing and distribution of output to users, computer operations, security administration, quality assurance, database administration, systems analysis, application programming, systems programming, network administration, and help desk administration.

Information systems management is also responsible for the proper separation of duties within the information processing function. The proper separation of duties helps reduce the possibility that transactions are improperly authorized or recorded. At a minimum systems analysis and programming should be separated from computer operations to prevent an analyst or programmer from making unauthorized changes to a program and then using that changed program in operations. In fact the only function that application programming might be combined with even in a small operation is the systems analysis function. The application programmer would have too much control if combined with other functions within the information systems organization.

### c. Access Controls

Access controls need to be pervasive and apply to end users, programmers, computer operators, security administrators, network administrators, management, and any others authorized to use the organization's computing capacity, including trading partners and other outsiders. Other outsiders include interconnection to affiliates' networks and links to external networks of these affiliates' networks. Once the local area network of an affiliate has access to an organization's network, it becomes a trusted network by default and has direct access to the network and its resources.

Logical access exposures result from unauthorized access which can lead to disclosure, manipulation, or destruction of programs and data. These exposures include changing data before or as it is entered into the computer, hiding malicious code in an authorized computer program, manipulating a program for personal gain, viruses, worms, logic bombs, trap doors, asynchronous attacks, data leakage, wire tapping, and denial of service attacks.

The most widely used logical access controls, which can also be classified as preventive controls, are the use of log-on IDs and passwords. Log-on IDs provide individual identification of the user and passwords provide authentication. The computer then contains a list of log-on IDs and access rules that allow access to files and data on a "need to know, need to do" basis. If these access rules are specified at the operating system level, they are more pervasive and provide better security than if specified only at the application level. Passwords should be easy for the user to remember, but hard for a perpetrator to guess. Some critical control features for passwords are that they: (1) should be internally one-way encrypted; (2) should be changed on a regular basis; (3) should be five to eight characters long; (4) should include alpha and numeric characters; (5) should not be easy to guess, such as a spouse's name, child's name etc., (6) should be masked (not appear on the screen when typed); and (7) inactive user IDs should be deactivated and eventually deleted from the system. Additionally, log-on IDs should be deactivated after several unsuccessful attempts (generally three) to enter the correct password and the system should automatically disconnect a log-on session if there is no activity for a specified length of time.

Increasing telecommuting has made remote access security critical. Dial-back procedures for personnel who are telecommuting from a set location do not work for personnel who are dialing in from various remote locations such as airports and hotel rooms. Remote access security requires the use of surround security measures such as firewalls, as well as the encryption of messages and sensitive files stored on the computer. Firewalls should filter dial-in access in such a manner as to deny access except where explicitly permitted.

Logical access controls are strengthened via the use of organization-wide classification schemes and naming conventions. These enable better control over computer files and aid in the verification and monitoring of security. Classification of data should be based on its relative sensitivity and should use a simple scheme such as high sensitivity, medium sensitivity, and low sensitivity. For example, the Department of Defense has the simple classification scheme: sensitive, but unclassified; confidential; secret; and top secret. Access should be denied to any user who does not have the appropriate level of authorization. However, even personnel who might be allowed access to highly sensitive data in their area should be denied access to highly sensitive data in another area. For example, someone might be cleared for access to highly sensitive data in production, but should be denied access to highly sensitive data in personnel or payroll. Naming conventions can be used to provide this type of filter.

Physical access exposures include unauthorized entry, damage, vandalism, or theft of equipment, copying, viewing, and abuse of data processing resources. Physical access controls should include, at a minimum, using bolting door locks to facilities that house computer equipment and hardware. More advanced physical access controls might include biometric controls, cipher locks, and electronic door locks.

Manual or electronic logging of visitors, along with escorted or controlled visitor access, also reduces physical access risks. Photo identification badges, video cameras, and security guards provide an even greater level of security. If photo identification badges are used, visitors should be required to wear a different color badge. Deadman doors, which consists of two doors and require that the first door close before the second door opens, provide a higher level of security to computer rooms and document stations. Maintenance personnel should be bonded. The location of sensitive facilities, such as the computer room, should not be advertised nor should they be identifiable or visible from the outside.

Some access problems must be addressed by the use of both logical and physical access controls. For example, many organizations operate in client/server environments. In this environment sensitive corporate data may be stored on or accessed from PCs that are not centrally located. To adequately protect PC data, both logical and physical access controls are required. The most widely used logical access controls over PC data are passwords and the use of encryption. The use of encryption will protect against any unauthorized use of the data stored on a PC even if the password protection is penetrated. Physical access controls over PC data include removing and locking up the storage medium when the PC is not in use, using lockable enclosures to protect against someone just taking the PC, and using an alarm system that will alert security if the PC is moved.

Both logical and physical access controls may also be needed to protect an organization against viruses, worms, and logic bombs. Client/server environments, local and wide area networks (LANs, WANs), and ready access to the internet, require sound control policies and procedures, as well as technical controls such as virus scanners and active monitors (active monitors look for virus-like activities and prompt the user to confirm they want to perform the activity requested). Sound internal control policies and procedures that would help protect against viruses, worms, and logic bombs include, but are not limited to, write protect all disks with .EXE or .COM extensions, allow no disk (including commercial software) to be used until it has been scanned (preferably on a stand-alone machine), boot only from disks that have been continuously write-protected, ensure antivirus software is installed on all workstations and the server, ensure that antivirus software is updated frequently, ensure that a sound backup policy is in place and operating effectively, educate users so they will comply with these good internal control policies and procedures, use hardware-based passwords, and use workstations without floppy drives.

### d. Environmental Controls

Information-processing facilities also face environmental threats. These include, but are not limited to, power failures, power spikes, natural disasters, fire, equipment failure, air conditioning failure, water damage and/or flooding. While these often result from events that the organization has no control over, certain environmental controls will reduce the damage caused by these events.

A variety of control procedures can help mitigate the damage caused by these environmental threats. Surge protectors can prevent the damage caused by power spikes, while an uninterrupted power supply (UPS) can mitigate the damage caused by power failures. The computer room should not be located in the basement or on the first two floors of a multistory building. This can help prevent water damage due to flooding. Water detectors can be used to reduce the impact of water damage due to pipe leakage, etc. To prevent fire and smoke damage, an organization should implement a combination of controls such as fireproofing walls, floors, and ceilings around the computer, using fire-resistant office materials, using fire suppression systems, and having regular inspections by fire department personnel. Smoke detectors will help to detect potential fires and warn personnel. The use of hand-held fire extinguishers can reduce the damage if a fire is detected early enough. If, in fact, there is a fire or a need for an emergency evacuation, there should be two emergency power-off switches, one located inside the computer room and one located just outside the computer room. Both switches should be easily accessible to authorized computer room personnel and clearly identified, but should be protected from accidental or unauthorized activation. Controlled keypad access to both of these switches could prevent inadvertent activation.

These environmental controls will also help mitigate the damage caused by natural disasters, but generally will not completely alleviate the impact of such disasters. When such disasters or other less severe business interruptions occur, the organization should have a plan for resuming normal operations as quickly as possible. These business continuity planning controls are discussed in the next section.

#### e. Business Continuity Planning Controls

Successful organizations must have a plan to continue business after interruptions caused by access violations, as well as a recovery plan in the event of environmental threats ranging from the interruption of electrical services to natural disasters such as floods, tornadoes, hurricanes, and earthquakes. While much of this business continuity planning process can be considered preventive in nature, once business is interrupted, the business continuity plan is a corrective control for resuming operations quickly.

A major component of any business continuity plan includes sufficient backup of systems and data. This requires sound policies and procedures to ensure that adequate backup exists. Particularly problematic is ensuring that end users follow appropriate backup procedures.

In recent years, information systems professionals and business managers have recognized the need for a more comprehensive business continuity plan than the traditional disaster recovery plan designed to be invoked in the event of a physical disaster. Blatnik asserts that businesses are probably suffering greater financial loss from less severe computer disruptions than from major physical disasters. These information systems professionals and business managers have also recognized the need for end-user personnel to be involved in the business continuity planning process. Disaster recovery and business continuity planning can no longer be viewed as the exclusive purview of the information systems processing function. With distributed information systems becoming the norm, it is imperative that all those who may have an impact on or may be affected by business interruptions be involved in the business continuity planning and maintenance process.

Business continuity planning begins with assessing the risks and identifying the critical processes that would require a quick recovery in the event of a business interruption. In order to identify critical processes it is important to have a complete understanding of the network and system design and to know what systems supports exist. Network diagrams and an assessment of system supports, including service arrangements, key vendor relationships, and maintenance agreements help to identify points of systems failure that might affect critical processes. These failures may range from the interruption of service due to a failure in a communication line, a hub, a host computer, or an overarching failure due to a large-scale physical disaster.

Once the critical processes have been identified, they need to be prioritized according to criticality and time sensitivity. Prioritizing the critical processes requires the input of business managers (or at least end-user personnel), as well as key technical staff and information systems auditors. Prioritizing the critical processes allows the organization to focus on those processes requiring immediate attention in the event of a business interruption.

A recovery plan for all critical processes should be developed such that each critical process has a clearly specified recovery plan. For example, the recovery plan could be as simple as keeping a spare hub on hand so that if a hub goes out, it can be replaced with minimal interruption of service. Another example would be if the main server fails, recovery might involve contacting the service personnel with the vendor and/or rerouting critical main server functions to backup servers. All recovery plans should be designed to minimize downtime.

To ensure that business continuity plans will enable the organization to quickly recover, testing of the business continuity plan is critical. At a minimum, a paper walkthrough of the plan by the personnel who are responsible for the plan's implementation should be performed. This paper walkthrough involves reasoning out what might happen in the event of various types of business interruptions. While this paper walkthrough provides the organization with useful information, it is not sufficient to conclude that the business continuity plan is complete. The information provided by the paper walkthrough should be used to modify and improve the business continuity plan. The paper walkthrough should then be followed by tests on different aspects of the plan where actual resources are expended in simulation of a system interruption. These tests should be performed at a time when actual disruption of business is minimized.

A full operational test, which requires shutting down operations, should only be performed after adequate paper tests and tests on different aspects of the plan have been performed. The results of every phase of this full operational test should be documented and analyzed. The analysis should include quantitative measures of the elapsed time, the number of critical systems that were successfully recovered, actual versus required measures of such items as the number of records successfully carried to the backup site, measures of the accuracy of data entry at the backup site versus normal data entry accuracy, and other measures of success.

The business continuity plan must be tested on a regular basis. It is important that the business continuity plan not be viewed as static. The pace of change in this environment is significant, and that puts additional stress on continuity plans that need to change correspondingly. The business continuity plan should include recovery plans for smaller scale business interruptions that should also be tested regularly and changed as necessary.

### f. Controls over Acquisition, Development, and Maintenance of Systems

A variety of systems development methodologies are in use today, with the systems development life cycle (SDLC) approach perhaps being the oldest and most frequently cited methodology. Whatever methodology is adopted, well-defined standards for all phases of systems development are required in order to provide adequate control of the acquisition, development, and maintenance of information systems. The systems development standards adopted need to be flexible enough to adapt to the type of systems development being undertaken, whether it is a local or a global system or application. For example, the standards adopted for the development of a system-wide database will differ substantially from the standards adopted for a small system developed by a group of end users. Well-defined standards help reduce the risk that the new or modified system will not meet users' needs, will take longer to complete than expected, or will be over budget. The most serious of these risks is that the new system will not meet the needs of the user.

While there are a variety of systems development methodologies, all essentially contain the following phases during the systems development life cycle: (1) requirements analysis, (2) solution definition, (3) design and build (or acquire), (4) testing, and (5) transition.

Before any development takes place it is important to clearly define the business requirements. One critical aspect of defining the business requirements that is often overlooked is the study of the existing system, including studying what works and what does not work, and the actual need for replacing or modifying the existing system. Users should be involved in this phase of development and should be aware of the need for systems development standards and how they can help meet those standards for the development project being considered.

After the business requirements have been clearly defined, the specific phases of the development process and the key deliverable for each of those phases should be decided early in planning phases of the project. Some of this might occur during the requirements analysis phase; however, much of it will depend on the definition of the solution and thus will occur during the solution definition phase.

The deliverables of the solution definition phase become the blueprint for the design and build phase of the project. The deliverables of the solution definition phase should include a definition of key user interfaces with the system, a definition of manual and automated processes, specification of how interfaces will work, specification of data structures and content, and specification of "error handling, security, control, backup, and contingency procedure" (Warren *et al.*). In essence, the solution definition defines how the system will appear to users.

The design and build phase includes developing the technical design of the system (i.e., how the system will be built), programming and testing of the system, developing user procedures, and training user personnel. Some key deliverables from this phase include procedures for the data conversion system, detailed test plans, software ready for systems testing, technical documentation, user documentation, help screens, training materials, and trained operations and user staffs.

The testing of systems prior to implementation is critical to the success of any systems acquisition or development; however, often not enough time is allocated to this critical phase of the development cycle. Often, subsystem testing is not performed in a timely manner during the design and build phase of the project. The amount and timing of testing should be related to the criticality of the subsystem/application. The last step in systems testing is to perform a comprehensive test of the entire system. "The key deliverable of this phase is the tested, formally accepted, and fully documented new system, ready for installation and production operation" (Warren *et al.*).

The last phase in systems development is the transition phase, which has as its key deliverable the new system implemented in the production environment. Part of the transition phase is the conversion of old files to new formats which must be completed before a new system is implemented. Often this conversion process is not controlled as well as it should be and, in particular, if conversion of master files is not adequately controlled it can have a far-reaching impact on the organization. Hence controlling the conversion process to ensure correct and complete conversion of the master files is critical. Conversion programs should be tested thoroughly and application control procedures, such as control totals and programmed edit checks, should be applied to the conversion of data. Because of the high risk of error and the potential far-reaching effects of those errors, the conversion process should be monitored. At a minimum, the following techniques should be used:

(1) test checking of individual data and records on the new file against the source records; (2) using audit software or verification programs to look for exceptional or unusual data on the new file; and (3) using audit software or specially designed programs to compare information on the new application files with that on the existing application files and reporting the differences.

### g. Network Controls

In order to address the complexity of networked systems, the International Standards Organization has developed network control standards in the following areas: configuration management, fault management, performance management, and security management. Configuration management consists of defining the network management system, the devices to which it connects, and the network configuration characteristics. In addition to a detailed map of the network topology and all the hardware within the network, good configuration management also supports fault management, performance management, and security management.

Fault management is the most important function of network management because it deals with maintaining "the availability of the network services and resources" (Warren *et al.*). There are four phases to fault management: (1) problem identification, (2) problem classification, (3) problem recovery, and (4) problem reporting. Having well-defined procedures for performing each phase will help maintain availability of network services and resources. Problem identification requires monitoring the network for conditions that are considered to be failures. Problem classification requires a comprehensive diagnostic testing capability in order to classify the cause of the failure. Recovering from the failures and restoring service rapidly (problem recovery) requires well-defined steps given the different types of network failure (equipment, high-capacity private lines, etc.). Proper records of failures, such as trouble tickets initiated by operators or help desk technicians, should be kept and periodic reports should be prepared indicating types of failures, steps taken to resolve failures, time to resolve failures, and the ultimate resolution of a problem.

Performance monitoring is as important, if not more important, for networked systems as it is for mainframe and client-server systems. Performance monitoring requires the collection, storage, and reporting of network availability, fault management performance, and network performance. In addition to supporting network management, performance monitoring can also help detect the misuse and abuse of network resources. Preventing the misuse and abuse of network resources requires defining and regulating access to network resources through the use of appropriate access controls such as authentication and authorization as well as the use of encryption.

### h. Data Communication Controls

LANs and WANs depend upon controlled data communications systems. Hence, good network controls depend upon good data communication controls. Data communication controls should be designed to mitigate the effects of three types of exposures: (1) passive or active subversive attacks; (2) failure of communication components; and (3) impairment of data during transmission (line errors). The discussion of access controls addressed the key exposures and controls related to passive or active subversive attacks. In this section the discussion of data communication controls will center on the failure of communications components and the impairment of data during transmission.

There are three broad categories of communications system components: (1) transmission media, (2) hardware, and (3) software. Transmission media include both bounded media (twisted-pair wire, coaxial cable, optical fiber) and unbounded media (microwave, satellite, and infrared). Examples of transmission media failure include a sliced cable or a microwave failure. Communications hardware includes amplifiers, repeaters, modems, ports, routers, bridges, gateways, multiplexors, concentrators, switches, etc. Examples of communications hardware failures include a power surge that knocks out a modem or a concentrator that fails. Communications software includes message, line, or packet switching software, data compression software, electronic funds transfer software, electronic data interchange software, polling software, buffer software, etc. Examples of software failures include program bugs or message collisions due to polling software failure.

The impairment of data during transmission can be caused by attenuation, delay distortion, or noise. Attenuation is the weakening of the signal as it travels along the transmission medium. To help prevent attenuation from occurring, amplifiers can be used to boost the signal strength for analog signals and repeaters can be used to boost the signal strength for digital signals. Delay distortion only occurs on bounded transmission media and occurs because the varying frequency components of a digital signal will arrive at the receiver at different times. Noise is an electrical fluctuation in the transmission medium that increases as more data are transmitted over a medium.

Several controls exist which help to mitigate the impact of the impairment of data: (1) redundancy checks, (2) echo checks, (3) automatic retransmission, (4) forward error correction, and (5) sequence checking. Redundancy checks require that additional information be sent along with the message which allows the receiver to perform an error check on received data. Echo checks, sometimes called loop checks, call for the receiver to send the message back to the sender so the sender can compare what was received with the stored message. Automatic retransmission requires the use of special characters appended to the message that allow the receiving unit to send a positive acknowledgment of receipt of a correct message or to send a negative acknowledgment of an incorrect message. If the sender receives a negative acknowledgment, the sender resends the message automatically. Forward error correction allows the receiver not only to detect if the message is incorrect, but also sends enough information to allow the receiving unit to correct the message without further involvement of the sender. Message sequencing allows for checking that all messages were received in the correct sequence.

## 2. Application Controls

Application controls include controls over data input, processing, files, and output and include both physical and logical controls. Input control procedures should ensure that every transaction that is received for processing is recorded accurately and completely. Input control procedures should also ensure that all transactions that should be processed are indeed processed and that all valid transactions are processed only once. Processing control procedures should ensure that proper processing of transactions takes place. File control procedures should ensure that only authorized processing is performed on stored data and that data integrity is maintained by not allowing concurrent access to a data item. Output control procedures should ensure that information systems output is only distributed or displayed to authorized users.

### a. Input Controls

Input controls include both data capture controls and data validation controls. Data capture controls should ensure that: (1) all transactions are recorded in the application system; (2) transactions are only recorded once; and (3) rejected transactions are identified, controlled, corrected, and reentered into the application system. There are two primary approaches to capturing data: (1) batching source documents of like transactions and processing them together or (2) on-line data entry. Batched input can consist of physical batches (a physical group of transaction source documents; physical batches are becoming less and less prevalent but still do exist) or of logical batches (transactions entered on-line but batched on a logical basis, such as employee ID number, before further processing).

Data capture controls for batched input include identifying batches by unique identifiers that include batch numbers, transaction types, and employee ID numbers; using control totals such as record counts and financial totals; and indicating when the batch was prepared and processed. Another critical batch control is to provide information on errors detected in the batch and the ultimate resolution of the errors (i.e., how the errors were corrected). Additionally, in order to keep track of physical batches as they are routed through the organization, every employee who handles batches should keep a batch control register in order to log the movement of a batch from initiation to processing and back to storage.

The primary data capture control for on-line data entry is a system generated transaction log. This transaction log should contain detailed information on each transaction, such as date and time of entry, transaction type, customer/vendor identification, and who input the transaction and from which terminal.

Data validation controls can be performed during data input or during data processing. For batch processing, data validation is generally performed during data processing, while for on-line input data validation generally occurs during input. There are three types of data validation checks: (1) field checks, (2) record checks, and (3) file checks.

Common field check controls include alphanumeric field tests, missing data (completeness) tests, range tests, limit tests, existence (validity) tests, and check-digit verification tests. An alphanumeric field test checks to ensure that an alphabetic field contains only alphabetic characters or that a numeric field contains only numeric characters. A missing data test checks to see that a field contains data, not blanks or zeros. A range test checks to see if the value in a field falls within an allowable range of values, while a limit test checks to see that the value does not exceed a predetermined limit. An existence test checks to determine whether the value is one of a set of permissible values (generally through a table look-up procedure). Finally, check digit verification checks to see that the check digit is valid for the value in the field.

Record tests are designed to determine whether the value in a field is consistent with the field's logical relationship to other fields. Three common record checks are reasonableness tests, valid-sign tests, and sequence checks. Reasonableness tests ensure that the value in the field is consistent with other data contained in the record. An example of a reasonableness test would be to test whether the value for an individual's salary is consistent with that individual's job classification. A valid-sign test ensures that the sign of the value is consistent with the transaction type. A sequence test checks for sequential numbering of transactions and issues an error message if the current transaction is out of sequence.

File checks determine whether the file used during data entry is the correct file type and is the latest version of the file. If the wrong file is accessed notification should be made immediately after it is accessed so that no processing is performed on the wrong file. File checks include internal labels that identify the file type, generation numbers that identify whether it is the correct version, a retention date to ensure processing is not performed on a file past its retention date, and control totals on the contents of the file to add to the other file check controls.

The resolution of data input errors is a critical aspect of data input controls. Data input error messages should be clear and concise and should lead to quick and accurate correction of the errors. Records should be kept of data input errors and the steps taken to correct the error and reenter the transaction into the system.

### b. Processing Controls

Processing controls are responsible for ensuring the proper computing, sorting, classifying, and summarizing of data. In addition to the application programs that perform these functions for specific subsystems within the organization, the processing of data is affected by the performance of the central processor, the operating systems that manage system resources, and the real or virtual memory where the program instructions are stored. Hence, proper processing of transactions not only requires strong controls within the application program, but also requires strong general controls. More specifically, strong controls over the acquisition, development, and maintenance of systems and programs, strong business continuity controls, and strong organizational controls, such as segregation of duties of information systems personnel, are required.

Within the application program, processing validation procedures help ensure that numeric fields have been properly authorized and are accurate, complete, and calculated correctly. In addition to the range check described previously, another field check that should be performed on numeric fields in a record is an overflow check. An overflow check detects whether a field used for calculations is zeroed out initially. If a field is not zeroed out initially, subsequent calculations can be in error. Additionally, record checks that should be performed on numeric fields at the processing stage include reasonableness tests and sign tests discussed previously.

File checks include the use of run-to-run control totals and crossfooting. Run-to-run control totals help ensure the accuracy of computations by allowing comparison of the master file balance before and after processing with the amount the file balance should be after the transactions have been processed. If the current balance of a cash account is $50,000 and incoming transactions contain a total of $7000 in additions to cash and $10,000 in subtractions from cash, the resulting balance in the cash account after processing should be $47,000. Crossfooting involves calculating separate control totals for related fields and crossfooting them at the end of processing. In a payroll file, separate control totals could be calculated for gross pay, total deductions, and net pay. After processing the control totals can be crossfooted to ensure that gross pay minus deductions equals net pay.

Other controls that are necessary to ensure proper processing include audit trail controls and checkpoint/restart controls. Audit trail controls should include an accounting audit trail that would allow auditors "to trace and to replicate the processing performed on a data item that enters the processing subsystem" (Weber). Additionally, an operations audit trail should allow the collection of resource consumption data (i.e., hardware consumption, software used, data files accessed, and personnel interventions required), the logging of security sensitive events such as failed attempts to use resources, the collection of hardware malfunction data, and the logging of user-specified events such as allowing a user-written program to access data.

Checkpoint/restart controls are critical to ensuring the proper processing of transactions. If a program is interrupted before normal termination, accurate and complete processing up to that point should not be repeated during the recovery process. Checkpoint/restart controls allow programs to be restarted and completed from a point just prior to the interruption.

### c. File Controls

File controls are designed to ensure that stored data is not improperly accessed and changed. In addition to the processing controls that help ensure processing is only performed on the correct version of the file and is accurate and complete, other controls over data files should be used to protect the stored data from corruption. Before and after processing image reports make it possible to determine the impact that transactions have had on stored data. All critical data files should record and report before and after processing image reports which allows the comparison of the data file before processing to after processing in case of a problem with the processing of transactions against the data file. In addition to internal labels described in Section II.B.2.b, external labels should be used for removable storage media to ensure the proper file is loaded for processing. Data file security controls should be used to ensure that only authorized users have access to stored data. This helps ensure that data is not corrupted by unauthorized access. Transaction logs which detail all transaction input activity (such as date of input, user ID, and terminal location) should be kept in order to help locate exceptions when they do occur.

Additionally, file controls (especially in a database system) should ensure that two processes are not allowed access to the same data item concurrently so that one process is not updating the data item at the same time that the other process is trying to update the data item. The result could be that one of the updates is not processed. Concurrency controls, which are quite complex particularly in a distributed database system, reduce the possibility of this occurring.

#### d. Output Controls

Output controls are designed to ensure that reports, checks, documents, and other displayed or printed information are not distributed or displayed to unauthorized users. Output controls should include report distribution logs, the secure logging and storage of sensitive and critical forms (especially negotiable forms such as blank checks), controlled computer generation of negotiable instruments, controlled physical distribution of reports (including having the recipient sign a distribution log indicating receipt of output), controlled disposition of physical reports, strict adherence to a formal record retention policy, periodic reconciliation of output to control totals, and establishment and adherence to formal output error handling procedures.

## III. AUDIT OF INFORMATION SYSTEMS

Weber defines information systems auditing as “the process of collecting and evaluating evidence to determine whether a computer system safeguards assets, maintains data integrity, allows organization goals to be achieved effectively, and uses resources efficiently.” This definition indicates that the objectives of an information systems audit are broader than those of the traditional financial statement audit (which is primarily concerned with safeguarding assets and financial statement data integrity) or traditional internal audit (which historically has been concerned with effectiveness and efficiency). The information systems audit encompasses the objectives of both. Because information systems auditing provides a competent, independent evaluation of the internal controls of an information system, it enables organizations to better meet both the security objectives (availability, integrity, and confidentiality) and fiduciary objectives (reliability, compliance, effectiveness, and efficiency) of internal controls described previously.

Networked computer systems are much more complex than the manual or legacy systems they replaced and this complexity impacts the collection and evaluation of audit evidence. In order to deal with the complexity of these computerized systems, information systems auditors break the information system down into its various subsystems. Subsystems of an information system can be logically grouped by whether they are part of the general controls subsystem or the applications control subsystem. The general controls subsystem consists of those functions which are designed to provide the support for the planned and controlled development, implementation, operation, and maintenance of the information system.

The applications control subsystem consists of all the application functions needed to accomplish reliable information processing. In the typical organization these application functions relate to the following accounting cycles: revenue and collection, acquisition and expenditure, production and payroll, and finance and investment. Within each application function, controls over input, file access, processing, and output are needed.

Information systems auditors should have a general understanding of the various subsystems that make up the organization-wide information system and the relationship of the various subsystems to each other in order to plan and conduct the audit. In the broad sense information systems auditors should have an understanding that the general controls subsystems provide the underlying foundation for the controlled operations of the applications functions within the applications control environment. The information systems audit involves evaluating evidence about the reliability of controls in each of these subsystems.

To assess subsystem reliability in the general controls subsystems, the information systems auditor determines the major functions performed by each subsystem as well as how those functions should be performed. The information systems auditor then evaluates how well the subsystem performs in comparison to how it should perform. When a function within the subsystem does not perform as it should, the information systems auditor determines that a failure of the subsystem function has occurred.

The approach to assessing subsystem reliability in the applications control subsystems is similar to the approach for the general controls subsystems. To assess subsystem reliability in the application controls subsystems, the information systems auditor determines the possible transactions that occur in the particular application and what would constitute proper processing of transactions for that application. A failure occurs if a transaction is not properly authorized, accurate, complete, and performed consistently with effective and efficient operations. Applications control subsystems failures may result in errors in an organization's financial information. The information systems auditor is concerned with errors which could cause material losses to the organization or material misstatements in the financial information reported by the organization and failures that have or could cause material losses to the organization through ineffective and inefficient operations.

Information systems auditors generally audit the controls in the general controls subsystems first to determine the reliability of the general controls environment, keeping in mind that general controls may not be consistently applied across all segments of large and complex organizations. The absence or improper functioning of a control at the general controls level indicates that a protective control which generally applies to many application functions is not functioning. This is a more serious control failure than the failure of a control in a particular application function which only affects that particular application function.

There are four categories of systems- and nonsystems-related audit procedures that are used to collect and evaluate evidence regarding subsystem reliability: (1) procedures to gain an understanding of the information system and its controls; (2) tests of controls; (3) substantive tests of events/transactions; and (4) substantive tests of balances/overall outcomes. Procedures to gain an understanding of the information system and its controls include inquiries, inspections, and observation. The auditor uses these procedures to determine what controls exist to meet management's control objectives, how they are designed, and whether they have been placed into operation.

Tests of controls focus on determining whether the controls are well-designed to meet specified control objectives and whether they have operated effectively throughout the entire audit period. The specific audit procedures used to test controls include inquiries, inspections, observation, and reperformance of the control procedure. To test information systems controls auditors can: (1) use generalized audit software (e.g., IDEA™, Audit Automation Software; ACL) to access and evaluate the contents of data files; (2) use specialized software to assess operating system controls; (3) use flow-charting techniques to document and evaluate automated applications; and (4) use operating system audit reports to evaluate operating system controls (ISACA).

The objectives of substantive tests of events/transactions and substantive tests of balances/overall outcomes are quite different from the objectives for tests of controls. The objective of substantive tests of transactions depends on whether the auditor is testing transactions from an attest perspective or from an operational perspective. From an attest perspective the auditor is concerned with whether the subsystem processing has led to erroneous or irregular processing which results in materially misstated financial statements. From an operational perspective the auditor is concerned with whether transactions/events have been processed effectively and efficiently. The objective of tests of balances and tests of overall results is to obtain sufficient evidence to make a final judgment as to the extent of loss or account misstatement that occurs when the information system fails to meet the objectives of safeguarding assets, maintaining data integrity, and achieving operating effectiveness and efficiency. For both types of substantive tests, the information systems auditor can use generalized audit software, specialized operating system software, and audit reports from operating systems to perform tests of balances/overall results.

The audit of most information systems, particularly networked systems, is quite complex and requires a well-planned process. The steps in an information systems audit mirror the steps taken in a traditional financial statement audit, but as previously discussed the objectives are broader. The audit objectives center around substantiating that internal controls exist to minimize risks to the organization and include both financial statement objectives (safeguarding assets and integrity of data) and internal audit objectives (effectiveness and efficiency of operations). The steps to be taken in an information systems audit include: (1) planning the audit by obtaining an understanding of the information system (or subsystem); (2) performing tests of the relevant controls; (3) performing tests of events (generally transactions for application subsystems) occurring in the subsystem; (4) performing tests of balances or overall results of the subsystem (not all subsystems will generate balances, but all subsystems should meet certain criteria for overall performance); and (5) forming an audit opinion and reporting the results of the audit.

The information systems auditor will generally follow these steps in the audit of the various subsystems of the information system; however, before the audit can be planned for the various subsystems, the auditor should first gain an understanding of the organization-wide information system. Specifically, to gain an understanding of the organization-wide information system and its controls, information systems auditors should familiarize themselves with the technical, managerial, and physical environment of the information systems processing facility. This familiarization process includes such steps as reviewing network diagrams, documenting access paths, touring the information systems processing facility, interviewing systems and network personnel, reviewing reports from access control software, and reviewing written policies, procedures, and standards. In reviewing policies and procedures particular attention should be paid to physical access policies, logical access policies, and whether the organization provides formal security awareness training. If the information systems auditors have prior experience with this information system, this familiarization process would include reviewing the prior year's working papers and noting any changes in the technical, managerial, and physical environment since the prior audit.

Once information systems auditors have familiarized themselves with the technical, managerial, and physical environment of the information systems processing facility, they are able to plan the audit of the various subsystems. They will prepare audit programs which detail the steps to be taken in auditing each subsystem of the information system. A primary concern of the information systems auditor is that control objectives for each subsystem are being met.

The information systems auditor performs tests of the subsystem controls which are designed to meet each of the subsystem control objectives. The information systems auditor tests policies and procedures as well as whether the proper separation of duties of subsystem activities is being followed. Where the proper separation of duties for subsystem activities is not being followed, the information systems auditor will assess whether compensating controls exist and will test them.

The testing of automated controls has moved beyond random sampling of transactions to include the use of audit software to analyze entire data files and the use of system exception reports for audit purposes. For example, the auditor can electronically analyze an organization's inventory file for slow-moving inventory and so forth. Auditors may also analyze system exception reports to gain an understanding of how effectively the system controls are working. The results of the tests of controls will provide the information systems auditor with evidence which can be used to determine the extent to which the auditor will need to perform tests of transactions/events and tests of balances/overall results.

When internal control objectives are being met and internal controls are operating effectively, the information systems auditor can rely on these controls. Such reliance can reduce the amount of substantive audit evidence required to perform the audit. The next two steps in the audit process are for the auditor to gather and evaluate evidence about the proper processing of transactions/events and then to perform tests of balances/overall outcomes.

The final step in the subsystem audit is to evaluate all the evidence gathered and make a final judgment on the overall operations of the subsystem. Information systems auditors will also develop recommendations for improving the operations of each of the subsystems.

Once all of the subsystems have been audited and the results of each subsystem audit have been compiled, information systems auditors are ready to report their findings for the overall audit of the information system. The overall report on the information system reliability will take into consideration how all of the subsystems interact with each other. This requires more than just an understanding of how well each subsystem operates; it requires an overall understanding of the complexity of the organization-wide information system.

Information systems auditors are all too aware that they not only must be well-versed in the techniques and control emphasis of the traditional financial statement audit, but must also draw on the expertise from the areas of management information systems, computer science, and even behavioral science. The knowledge bases of these disparate fields of study are far too broad for any one individual to be well-versed in each; hence, information systems audit teams should be composed of individuals with a variety of talents and the ability to communicate with each other about their individual areas of expertise.

## IV. CONTROLS IN THE CONTEXT OF CURRENT AND FUTURE TECHNOLOGIES

The increasingly widespread computerization of information, combined with the widespread access to this information through the technology provided via the internet, has magnified the need for the implementation of good internal control policies and procedures. Much information that was previously obtained through a long and tedious process is now virtually at the fingertips of millions of internet users.

Staying abreast of control issues related to advances in existing technologies, as well as new and emerging technologies presents an ongoing challenge for information systems auditors. Control issues related to these advances will require new and innovative control approaches. For example, the authentication of users in public networks became an issue in the late 20th century and will continue to be an issue. One way to address this issue is through the use of digital certificates, which allow one to send and receive secure e-mail. To help auditors stay abreast of these and other technology related issues, the Information Technology Section of the American Institute of Certified Public Accountants issues an annual list of top ten technologies, top ten technology issues, top ten technology applications, and emerging technologies (see www.toptentechs.com). Many of the technologies, technology issues, technology applications, and emerging technologies addressed by the Information Technology Section are likely to be of concern to information systems auditors during the early part of the 21st century.

## ACKNOWLEDGMENTS

We thank Lyn Graham and Bill Powers of BDO Seidman LLP for their useful insights into this topic.

## SEE ALSO THE FOLLOWING ARTICLES

Accounting • Benchmarking • Computer-Aided Manufacturing • Cost/Benefit Analysis • Procurement • Productivity • Public Accounting Firms • Supply Chain Management • Transaction Processing Systems

## BIBLIOGRAPHY

ACL web site at www.acl.com/en/.

Blatnik, G. (1998). Point of failure recovery plan. *IS Audit and Control Journal,* Vol. IV, 24–27.

Committee of Sponsoring Organizations of the Treadway Commission (COSO). (1992). *Internal Control—Integrated Framework: Framework.* New York: Committee of Sponsoring Organizations of the Treadway Commission.

Financial Accounting Standards Board (FASB). (1978). *Objectives of financial reporting for business enterprises. Statement of Financial Accounting Concepts No. 1.* Stamford, CT: FASB.

IDEA, Audit Automation Software web site at www.caseware.com/.

Information Systems Audit and Control Association (ISACA). (1998). *1999 CISA Review Technical Information Manual.* Rolling Meadows, Illinois: Information Systems Audit and Control Association.

Information Systems Audit and Control Association (ISACA). (1998a). *COBIT: Control Objectives for Information and Related Technology: Executive Summary.* Rolling Meadows, Illinois: Information Systems Audit and Control Association.

Information Systems Audit and Control Association (ISACA). (1998b). *COBIT: Control Objectives for Information and Related Technology: Framework.* Rolling Meadows, Illinois: Information Systems Audit and Control Association.

International Federation of Accountants (IFAC). (1998). Managing security of information. Information Technology Committee, web site at www.ifac.org/. New York, IFAC.

International Federation of Accountants (IFAC). (1999). Managing information technology planning for business impact. Information Technology Committee, web site at www.ifac.org/. New York, IFAC.

International Standards Organization (ISO) web site at www.iso.ch/.

Messier, W. F., Jr. (2000). *Audit & Assurance Services: A Systematic Approach,* 2nd ed. New York: McGraw-Hill.

Warren, J. D., Edelson, L. W., and Parker, X. L. (1999). *Handbook of IT auditing.* Boston: Warren, Gorham & Lamont.

Weber, R. (1999). *Information Systems Control and Audit.* New Jersey: Simon & Schuster Company.

Wilson, G., International Federation of Accountants (IFAC). (2000). Articles and Speeches Library, Electronic Authentication Technologies: E-mail Security and DigitalCertificates, web site at www.ifac.org/. New York, IFAC.

# Copyright Laws

**Alisha D. Malloy**
*Georgia State University*

**Kannan Mohan**
*Georgia State University*

**Detmar Straub**
*Georgia State University*

**Amrit Tiwana**
*Emory University*

## GLOSSARY

**copyright** The right of literary property as recognized and sanctioned by positive law. An intangible, incorporeal right granted by statute to the author or originator of certain literary or artistic productions, whereby he is invested, for a period of up to 70 years after death, with the sole and exclusive privilege of copying, publishing, and selling them.

**fair use doctrine** A privilege in others than the owner of a copyright to use the copyrighted material in a reasonable manner without the owner's consent, notwithstanding the monopoly granted to the owner.

**intellectual property** Intangible property created by individuals or corporations that is subject to protections under trade secret, copyrights, and patents laws.

**patent** A grant of some privilege, property, or authority, made by the government or sovereign of a country to one or more individuals for the unique appearance or design of an article of manufacture made to protect against duplication of a design if it is original, nonobvious, and ornamental.

**trademark** A distinctive mark of authenticity through which products of particular manufacturers or the vendible commodities of particular merchants may be distinguished from those of others.

## I. INTRODUCTION

Copyright is the body of law that deals with the ownership and use of works of literature, music, and art. By extension, it has come to be applied to works that have produced information and are novel. By United States law, it is an intangible, incorporeal right granted by statute to the author or originator of certain literary or artistic productions, whereby he is invested for a period of up to 70 years after death, with the sole and exclusive privilege of multiplying copies of the same and publishing and selling them. In response to rapid technological changes, laws concerning information use are undergoing continuous change, including the evolution of copyright laws. The ease of replication, transmission, and alteration of digital media has increased the lack of understanding in applying copyright laws to information in electronic form.

The advent of the Information Age and the Internet raises a number of new issues concerning use of information. Much of the responsibility for this rests with information systems (IS) management. One of the most important of these is how to deal with information as a commodity. Rapid advances in technology have brought with them new responsibilities for organizations to design adequate safeguards to protect against wrongful use of information. This article focuses on copyright issues as they pertain to information systems and the Internet. A brief summary of the current legal situation is given; followed by issues concerning international copyright laws and the Internet; then recommendations are given to IS and general management on how to mitigate these risks.

## II. DEFINITIONS

### A. Copyright

Copyright is a United States statutory grant, which protects creators of intellectual property against copying by others for any purpose. United States federal law grants copyright exclusively. A copyright provides an author with a tool to protect his work from being taken, used, and exploited by others without permission. The owner of a copyrighted work has the exclusive right to reproduce it, prepare derivative works based upon it, distribute copies by sale or other transfer of ownership, and to perform and display it publicly and to authorize others to do so. The basic purpose of copyright is to enrich a society's wealth of culture and information. While copyrights protect original works of authorship, it does not protect against independent creation of similar or identical works.

Authors are not required to publish or register their fixed works in order to secure copyright protection. In the information systems community, copyrights ensure that companies whose livelihood depends upon intellectual property, like software companies or Internet-based publishing companies, can compete in the marketplace. Without the ability to prevent unauthorized copying, sale, and distribution of its products, intellectual property-based companies would not be able to survive.

### B. Fair Use Doctrine

The "fair use" doctrine is a privilege of those other than the owner of a copyright to use copyrighted material in a reasonable manner without the owner's consent, notwithstanding the monopoly granted to the owner. To determine whether fair use has been made of copyrighted material, the nature and objects of the selections made, the quantity and value of the material used, and extent to which the use may diminish the value of the original work must be considered.

Fair use involves a balancing process by which a complex of variables determines whether other interests should override the rights of creators. The U.S. Copyright Act explicitly identifies four interests: (1) the purpose and character of the use, including its commercial nature; (2) the nature of the copyrighted work; (3) the proportion that was "taken"; and (4) the economic impact of the "taking." Fair use is for purposes such as criticism, comment, news reporting, teaching, scholarship, or research but not restricted to these. It is an affirmative defense to an action of copyright infringement.

### C. Patents

A patent is a legal grant to exclude others from making, using, or selling one's invention and includes right to license others to make, use or sell it (*Valmont Industries, Inc. v. Yuma Mfg. Co.,* 296 F. Supp. 1291, 1294, (1969)). Governments establish patent systems and grant patents to encourage innovation, technical development, and ultimately economic prosperity. One important aspect about patents is that they do not give the patent owner the right to practice the invention claimed in the patent, but only to exclude others from practicing this invention. The requirements for obtaining patents are that the invention should be "new" and useful and it must not be an obvious extension of previous inventions or technology.

### D. Trademarks

Trademarks are the brand names used by manufacturers to identify their products. These are effective ways for manufacturers to distinguish their products from their competitors' products in the marketplace.

### E. Copyright versus Patents versus Trademarks

Copyrights, patents, and trademarks are grouped together as intellectual property, which could be seen as the product of one's intellectual endeavors. The difference between these three is that the purpose of copyright is to promote the progress of science and useful arts. The purpose of a patent is to encourage inventions and their disclosure to the public. The purpose of a trademark is to prevent confusion in the marketplace, thereby assuring accurate information and the maintenance of quality goods and services. Copyright is a form of protection to the authors of "original works of authorship" including literary, dramatic, musical, artistic, software, databases and certain other intellectual works. Patents grant the owner an exclusive monopoly on the ideas behind an invention. It is the ideas, not the invention that can be patented although the ideas have to be put in a tangible form in an invention before they can be patented. Patents protect the ideas, which can be implemented in practice while copyrights protect the expression of ideas.

## III. SCOPE

The necessity for the general management of organizations to understand the scope of copyrights, patents, and trademarks is justified by the responsibility of the management to protect the organization from exposure to information liability.

### A. What Can Be Copyrighted/Patented/Protected as Trademarks?

Copyrights protect creative endeavors such as writings fixed in a tangible medium. Under the Copyright Act of 1976 the expressions that can be copyrighted include literary works, artwork, sculpture, photographs, and music. Copyrights do not protect ideas, but they protect the expression of ideas set forth in a tangible medium.

Different types of work on the Internet that are protected by copyright laws are written works like e-mail, articles placed on ftp or web servers, musical or audiovisual works, digitized images, software, web pages, and databases.

Patents protect products, machines, methods, or compositions as they are embodied in utilitarian or aesthetic designs. They can protect new and better utilitarian objects, methods of making those objects, and methods of using them.

Trademarks protect words, symbols, or other forms of expression, which identify the source of the goods or services provided by a person or entity.

### B. Who Owns Copyrights?

Employers are owners of any work created by an employee within the scope of employment. This case of "works made for hire" where the employer owns the copyright for the work is an exception to the concept that the creator of a work is the copyright owner of the work. Legislation has been enacted to set a federal standard that would prevent an employer from demanding an assignment of rights in an employee's invention unless it falls within the definition of an "employment invention." The party who pays for the work owns only a tangible copy of the work, while the creator of the work owns the copyright of the work. If a group creates a work, the copyright may be held jointly. In the case of a joint work, if one author gains some benefits by exploiting the work, the benefits should be shared with all other authors.

Works of authorship are protected upon creation, i.e., it is subject to copyright protection immediately after it is put in any tangible medium. The work must satisfy the threshold standards of originality and creativity in order to be subject to copyright protection. The concept of originality differs from that of novelty in the case of patents in the aspect that the work can be identical to another work created by another party and still be subjected to copyright protection as long as the party that had created the second work had not copied it from the existing work. A work must be sufficiently creative in order to be protected by copyright laws.

### C. Duration

Copyright works created before January 1, 1978, are protected by United States law for the length of the author's life plus another 50 years. Works created after January 1, 1978, are protected for 70 years after the author's life due to an extension granted by the 104th Congress, 1st Session in 1995. In the case of joint works, copyright protection is granted for the length of the life of the last surviving joint author plus another 50 or 70 years depending on the creation date.

### D. Types of Rights

The two major categories of copyrights are moral and patrimonial rights. Moral rights include the right of publication or the right to choose when to disclose his or her work, right of attribution or association with the work as its author, and the right to the integrity of the work or the right to oppose any modifications made to his or her work. Patrimonial rights include the right of reproduction, right of transformation, the right of distribution, and the right of public communication.

## IV. CHALLENGES TO THE CURRENT INTELLECTUAL PROPERTY RIGHTS

The unique capabilities of the current technological systems pose a variety of challenges in the form of legal and political pressures on intellectual property rights. Some of the problems identified by the Office of Technology Assessment, Congress of the United States are as follows:

1. *Problems of identifying ownership.* The advanced networking capabilities that enhance collaborative work make it very unwieldy to administer the concepts of originality and original authorship.
2. *Problems of identifying infringements and enforcing rights.* With the current high speed communication media and large capacity storage technologies, it becomes much more than a case of individuals trading vast quantities of copyrighted material without the knowledge or permission of the copyright holders. It could be a case where individuals can inexpensively and privately share the contents of an entire library.
3. *Problems of private use.* It would be impossible to track down people who copy copyrighted material.
4. *Problems of functional works.* With the advent of software engineering, differences between inventions and writings cease to be clear cut. This demands changes in the legal framework in order to facilitate inclusion of the new information-based products.
5. *Problem of derivative use.* The use of secondary material would be restricted if the copyright holders have the privilege of benefiting from all subsequent works based on their original works.

## V. UNITED STATES LAWS, STATUTES, AND ETHICS

### A. Evolution of Copyright Laws in the United States

The first United States statute was formulated in 1790 offering protection for books and illustrations. It was in 1980 that a United States federal law known as The Software Act was enacted, which modified the existing copyright protection by including software programs as protected under the Copyright Act.

The copyright laws in the United States have evolved to a point where the omission of a copyright notice will not jeopardize the owner's copyright interest. The first step toward legal regulation of Internet copyright matters was taken in 1993, when the Clinton Administration formed the Information Infrastructure Task Force (IITF). The IITF was charged with the development of the national information infrastructure (NII) and has since produced two reports on what it perceives as needed changes in copyright laws. Critical changes to the right of transmission were made. In 1997, former President Clinton signed the No Electronic Theft (NET) Act, which amended Section 506 of the Copyright Act. This NET Act is specifically aimed at software pirates on the Internet.

### B. Licensing

United States governmental agencies regulate licensing. The Reproduction Rights Organization (RRO) has the authorization to license organizations to photocopy registered titles from their repertories. The Copyright Clearance Center (CCC) issues several types of licensing agreements based upon the type of organization seeking a license. The most common one is the Annual Authorizations Service Repertory License Agreement, which allows corporations to make unlimited photocopies provided these copies are used internally and the titles are part of the CCC's repertory of registered works.

The Digital Millennium Copyright Act (DMCA), signed in October 1998 provides two categories of protection against circumvention of technological measures used by copyright owners to protect their works. The first category includes measures to prevent unauthorized *access* to copyrighted work and the second category includes measures to prevent unauthorized *copying* of a copyrighted work. Violations of these provisions are subject to criminal prosecution.

Even though the World Intellectual Property Organization (WIPO) has decided that electronic transmission of material should be protected by copyrights, it has not yet been explicitly included in the United States copyright law.

### C. Infringement

Infringement is the unauthorized use of copyright material, i.e., the use without permission of copyright holder. In determining whether there is a copyright infringement, and not a fair use exemption, the factors to be considered include: (1) the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes; (2) the nature of the copyrighted work; (3) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and (4) the effect of the use upon the potential market for or value of the copyrighted work, see Copyright Act, § 107 (17 U.S.C.A.).

Remedies for copyright infringement include injunctive relief, impounding and disposition of infringing articles, and recovery of actual damages and profits. In lieu of actual damages, the Federal Copyright Act provides for statutory damages which will vary as to whether the infringement was willful and unintentional, see Copyright Act, § 504 (17 U.S.C.A.). There are three categories of infringement: direct, contributory, and vicarious.

### 1. Direct Infringement

A copyright is infringed directly when one of the exclusive rights of the copyright holder is violated. The plaintiff is required to prove just the ownership of copyright and copying by the defendant. The offending party can be prosecuted even if he or she did not know that the work in question was protected. Intent is considered as a factor only when calculating damages.

### 2. Contributory Infringement

Contributory infringement occurs where a person with knowledge of the infringing activity induces, causes, or materially contributes to the infringing conduct of another. Substantial or pervasive involvement is required.

### 3. Vicarious Infringement

Even without participation or knowledge a defendant can be held liable for the actions of a primary infringer where the defendant has the right and ability to control the infringer's acts and receives a direct financial benefit from the infringement.

## VI. COPYRIGHTS IN THE NEW ECONOMY

Global networks and communication channels cut across national boundaries and spawn a new realm of activity that, in effect, undermines both the feasibility and the legitimacy of applying laws based on geographic boundaries.

The self-regulated environment that characterizes the World Wide Web poses new and unforeseen challenges for existing copyright laws. Almost every transaction involving the transfer of information can be conducted on-line: education, health care provision, delivery of intangible services, publishing, music, and the practice of law. Explicable intellectual property can be reduced to digital form, and digital information can be globally redistributed at low or near-zero cost. As existing copyright laws are reactively retrofitted to address various issues surrounding copyright protection on the Web, their scope and breadth challenges the notion of fair use that has since long been upheld in the traditional publishing industry. The fundamental problem with copyright laws, as noted over a decade ago, is that they represent "a patchwork of new and reapplied laws that offer little clarity on underlying issues."

### A. Implications of the Internet on Content Digitization

The most critical implications of the widespread adoption of the Internet are observed in the context of goods that are intangible in nature. Such intangible goods can range from electronically packaged services to digital information products. Various descriptive terms such as information commodities, digital goods, and information products have been used in past research. (For the sake of brevity, we use the term "information products.") An information product is defined as a highly interdependent package of information that is capable of being distributed in digital form. Software engineering products, CD-ROM databases, print-on-demand services, electronic libraries, electronic newspapers, digitized music and video content, and web content are examples of such products. In economic terms, the fixed costs associated with their production are high and the variable costs are relatively low. If left to the market place, the price of an information product will be low due to its low marginal cost of reproduction. Furthermore, because such products are experience goods, their pricing is *perceived value*-based, and not cost-based. As the trade of such intangible goods increases, copyright laws become increasingly vulnerable and subject to unenforceability.

The Internet has provided a channel for the distribution and trade of such products at low overhead costs. When distributed over such a medium, information products' variable cost of production and distribution approaches zero, as the product has no physical form (unlike retail packaged software and music, or nondigital information products). However, their intangible nature also causes severe competitive market challenges that only worsen because of their low economic cost of reproduction coupled with high fixed costs.

### 1. Supply-Side Effects on Incremental Costs of Digitized Products

Digital technology and the Internet change two significant supply-side costs that govern the feasibility of content distribution. First, reproduction costs are dramatically reduced by the act of digitization of content. Perfect copies—indistinguishable from the original—can be made. Second, public networks generally and the internet specifically, facilitates distribution of these perfect reproductions "quickly, easily, and cheaply."

Information is an experience good—judgments about its value cannot be made until it is actually seen. Production costs of information products are determined by their "first copy costs." Information products often involve high fixed costs but low marginal costs. In the case of digital information products, this implies that the fixed cost of producing the first copy of the product is substantial and every subsequent copy is negligible or nears zero. The key implication of the first copy costs characteristics of an information product is that these costs are incurred by the producer and if copyright laws are unable to prevent subsequent unauthorized copying, it will guarantee distribution of the product at it's incremental cost—zero.

## B. Copyright Limitations and Electronic Commerce

With the emergence of electronic commerce, the notion of the term "copying" must be revisited. In an electronic, Internet-mediated environment, one cannot access any information without making multiple copies. Temporary copying occurring within a computer's memory to enable readers to read documents is considered "reproduction." Existing laws governing copyrights on the Internet take a restrictive view of what constitutes fair use. "The ability to redistribute to a friend one's own copy of a work" constitutes copyright violation according to the existing United States laws.

### 1. Recipient Liability Issues

Another issue relates to liability of the recipient of an illicit digital copy of a work. Copyright law stipulates that the owner of any computer on which appears a copyright infringing duplicate will be held liable along with the unwitting recipient of the information artifact, however innocent he may be. If this article of the law were to be enforced, the unwilling recipient of an e-mail attachment containing copyright infringing work could be held liable for copyright violation.

### 2. Database Protection and Compilation Copyrights

Article 2B of the DMCA, which is discussed in the following section, addresses the question of protection offered to information compilers—organizations that verify, organize, format, and "scrub" publicly available information in the final form of a database. Copyright laws do not provide protection to all such compilations, but only to those that exhibit sufficient novelty and creativity in their selection, organization, and arrangement. In digital form, such data compilations are, in United States law, afforded "thin" protection that only covers their creative organization, and not their content or arrangement. The European Union enforces a law for protecting intellectual property rights (against reuse and extraction for a period of 15 years) of database developers who make "substantial investments" in their development. In the United States, however, an equivalent law was struck down and declared as being both unconstitutional and detrimental to scientific research.

# VI. THE DIGITAL MILLENNIUM COPYRIGHT ACT

The DMCA which was passed by the United States Congress in November 1998 addressed some of the issues related to copyright protection on the Internet, and yet left others still calling for attention. The DMCA specifically addressed liability issues for online service providers including but not limited to Internet access providers, web sites, and telephone companies by introducing so-called "safe harbors." Under these safe harbors, on-line service providers are exempted from liability if they are willing to:

1. Terminate service to repeat infringers
2. Accommodate standard technological measures adopted by copyright industries to protect intellectual property and works
3. Remove infringing material when notified to that effect
4. Abide by certain conditions that are imposed to support the above in an evolutionary manner

These safe harbors facilitate system storage, links to other sites, system caching, and the transmission and routing/rerouting of information provided by the online service providers if they are unaware of the presence of copyright infringing material and do not benefit from such infringement.

The DMCA has provided a legal device for preventing electronic piracy for use software, entertainment, and music producers; it has, however, met with resentment from many consumer groups that believe that the DMCA places severe restrictions on their traditional access to and use of information products.

## VIII. COPYRIGHT ISSUES ON THE INTERNET: TWO CASES

Two cases of copyright problems illustrate these concerns: the MP3 music format and electronic books. Both technologies were originally commercialized to facilitate legitimate trade of both music and books.

### 1. Case 1: MP3 and Copyrights on Music

The MP3 music format is a public domain digital format for streaming music over the Internet. A song can be compressed into a small electronic file that can be transmitted rapidly over the public Internet and automatically decompressed on the receiving end. Independent artists and music producers that might want to distribute their music without the deep pockets that widespread distribution has traditionally entailed have considered the MP3 format a boon. Several electronic commerce businesses such as emusic.com, mp3.com, and tunes.com were founded on the basis of this format. However, new advertising supported software that is distributed free of charge to users enables them to use the same format and create a huge directory of digital music titles "ripped off" or converted from legally obtained music CDs. Other users navigating this directory can see both their own and thousands of others' stolen MP3 file collections and freely download them. Because the copy itself is digital, there is no degradation in music quality and thousands of pirated copies can be easily made. This led to one such software developer, Napster, being taken to court in March 2000 by two bands, Metallica and Dr. Dre, over violation of copyrights. Napster's plea for indemnity on the basis of being an on-line service provider was turned down for its failure to act when the company was aware of copyright violations that its software enabled.

The industry fought back with traditional methods such as lawyers and law enforcement agents, as well as with technology. While copyright protection technologists have tried pursuing novel technical solutions to prevent copying, MP3 proponents have repeatedly developed workarounds for any protection. While the introduction of digital music brings unforeseen opportunities, it also comes with its share of copyright protection problems. Although court orders have restricted uncontained piracy of copyright infringement through peer-to-peer networks such as Napster, the promise of digital distribution of information products has motivated several international record labels to replicate that model while digitally "tagging" individual songs. Such tagging will prevent widespread copying, it will also restrict the ability of the actual buyers to listen to legally acquired digitized music. Copyright laws need to be rethought for the digital age. Additionally liability and conspiracy, as defined in our existing laws, need to be clarified in this context.

### 2. Case 2: E-Books and Publishing Copyrights

The publishing industry has been toying with the idea of not just selling, but also publishing books through the Internet. The logic is straightforward: Readers can purchase a copy of the book on a real-time basis and at the same time publishers can benefit from the reduced production and distribution costs. This translates to a combination of lower prices and higher profit margins within these markets. The publishing industry can also better afford to electronically publish very specialized short print run titles that would not justify the expense of conventional printing.

Unfortunately, existing copyright laws afford little protection to on-line publishers, and that unhinges the value logic for electronic publishing. Markets for electronic books and their paper-based equivalents are cannibalistic; i.e., sales in one category are thought to reduce the same product's demand in the other. Given this market, selling books and similar productions online does not guarantee a revenue stream that justifies losing the same from the print-based market segment. The only justification is that publishers can be assured that one digital copy of the book will not be redistributed ("pirated") to a hundred other potential purchasers by the original purchaser.

## IX. OPTION VALUE-BASED PROTECTION

The enforceability of copyright laws is questionable in the boundless economy created by the Internet and electronic commerce. Legal differences, national culture, and differing value systems make it difficult, if not impossible, to enforce a common set of laws across all countries. Some have suggested that much chargeable value in the case of information products will be in certification of authenticity and reliability, not in the content itself. If consumer-perceived value is maximized, sustainable increasing economic returns can be generated through self-reinforcing positive network feedback loops that characterize information goods. Economists have also recommended further exploration of the notion of option value—select abilities valued by the consumer—associated with digital information products that are vulnerable to copyright infringement. The effectiveness of copyright protection then depends on a potent combination of technology, reasonably enforceable laws and good economic judgments.

## X. KEY ISSUES AND MANAGEMENT CHALLENGES FACING ORGANIZATIONS

Even with the current repository of ill-defined United States copyright laws and ethical issues dealing with the rapidly evolving information technology and the Internet, organizations are still liable for the misuse of intellectual property contained within these applications. Organizations must not only protect their own copyrighted intellectual property from infringers, but they must also instill in employees the importance of reciprocating this protection to material they may find from external sources, commercial databases, and the Internet. Information technology (IT) managers, as well as general managers, must be aware of the legal liabilities that can occur as a result of internal misuse of data, software programs, and the Internet.

Information technology is a double-edged sword. While it is the source of many benefits, it is also the source of many dilemmas. This is especially evident when dealing with copyright issues and IT. The exploding use of networks and the Internet further challenges copyrights ethical and legal frameworks and protections due to the ease of replication, transmission, and alteration of the information.

The 1990 Straub and Collins paper suggests that there are three main areas of concern for all managers: (1) how to provide functionality while respecting the intellectual property rights of external intellectual property creators; (2) how to acquire and utilize external information without violating licensing contracts or infringing on copyrights; and (3) how to collect and disseminate information on individuals while respecting individual rights and privileges. The following areas are specifically emphasized: ownership, protection, software piracy, and downloading from the Internet.

### A. Issues of Ownership of Intellectual Property

Who owns custom-made software? Is the owner the person who wrote the software program/intellectual property or the company for which the author wrote the program/intellectual property? What is to prevent a programmer from taking copies of programs from one job to another? The answers to these questions are well established within U.S. Code, Title 17, § 201, Ownership of Copyright, which states the following.

#### 1. Initial Ownership

Copyright in a work protected under this title vests initially in the author or authors of the work. The authors of a joint work are co-owners of copyright in the work.

#### 2. Works Made for Hire

In the case of a work made for hire, the employer or other person for whom the work was prepared is considered the author for purposes of this title, and, unless the parties have expressly agreed otherwise in a written instrument signed by them, owns all of the rights comprised in the copyright.

#### 3. Contributions to Collective Works

Copyright in each separate contribution to a collective work is distinct from copyright in the collective work as a whole and vests initially in the author of the contribution. In the absence of an express transfer of the copyright or of any rights under it, the owner of copyright in the collective work is presumed to have acquired only the privilege of reproducing and distributing the contribution as part of that particular collective work, any revision of that collective work, and any later collective work in the series.

Therefore if the author of the software (i.e., programmer) is in the employ of an organization, the software belongs to the organization, not to the programmer. For example, the programmer may not take the software to the next job. If the programmer is a consultant, however, the ownership of the software produced should be spelled out specifically in contractual form; otherwise, the parties enter extremely muddy legal waters.

Questions of just who owns intellectual property has resurfaced with the advent of digital publishing of intellectual property and commercial databases made available on the Internet. It is the customary practice in the information industry for the content provider to be responsible for obtaining all necessary copyright interest for online distribution. However, in *Tasini v. The New York Times Co. Inc.,* the Second Circuit Court of Appeals in Manhattan ruled that several publishers including *The New York Times* and *Newsweek* did not have the right to put freelance journalist articles into electronic databases without the authors' permission.

## B. Issues of Protection of Intellectual Property

In the information age of today, intellectual property is now an organization's primary source of competitive advantage, a situation that is very similar to physical products in the industrial age. In an information intensive industry, copyright should be an essential weapon for organizations, especially as they continue to migrate their intellectual property to networks and the Internet. Yet the ever-increasing lawsuits concerning copyright infringement show that there is a trend in most organizations to respond to these trespasses on intellectual property, IT, and the Internet with legal uncertainty, misinformation, and disinterest. Current attitudes within organizations could lead to further degradation of the copyright as a legal weapon in protecting trade secrets, if not its ultimate extermination as a powerful source of protection.

## C. Issues of Software Piracy

Software piracy, or stealing software, is illegally obtaining and using software. When individuals and organizations purchase software, the sole right that is inferred to them is the use of the software. Mere purchasing does not confer the right to copy or misuse the software outside the scope of the copyright. It is unlawful to copy any copyrighted software or intellectual property. Whether it is the casual sharing of copyrighted software among friends or assembly line copying by organized crime, unlawful copying incurs major losses for software vendors and content providers.

The proliferation of IT and the Internet have made illegal reproduction and distribution of copyrighted software and intellectual property commonplace. According to the Software and Information Industry Association (SIIA) and Business Software Alliance (BSA) fourth annual study on global software piracy, worldwide losses from software piracy amounted to almost $11 billion in 1998. Of the 615 million new business software applications installed during 1998, 231 million, or 38%, were pirated. Losses in the United States alone were estimated at $3.2 billion, 26% of the worldwide revenue losses.

With the increased popularity of local area networks (LAN) and client/server, organizations typically purchase only one package of each application software and place it on the network so everyone can use it, which saves money over purchasing individual copies. In order to do this legally in the United States, an organization must purchase a site license, which is a contract with a software manufacturer that grants the organization the right to let a specified number of people access and use the one network software copy. The site license agreement states how many people may simultaneously use a copy of the software. If more people need to use the package than stated in the site license agreement, additional fees must typically be paid for each new person added.

## D. Issues of Downloading Intellectual Property from the Internet

With new advances in IT occurring almost daily, extant United States copyright laws are in danger of becoming antiquated. Most organizations are faced with the inevitable migration to the Internet where copyright, as it relates to content, will face its greatest challenges yet. This is due to several factors: (1) increased accessibility brings increased risk; (2) sale to multiple often anonymous end users rather than known corporate entities dissolves the security of contractual relationships, making enforcement of copyright infringement more difficult; and (3) migration to the Internet may leave many organization's databases unprotected. As far back as 1991 a United States court concluded that a modicum of creativity in the selection, coordination, or arrangement of a database is required for it to be copyrightable *(Feist Publications v. Rural Telephone Service Co).*

# XI. POSSIBLE IMPLEMENTATION, POLICIES, AND SOLUTIONS

## A. Technological Solutions

In attempts to quell the flood of recent copyright violations, technological solutions have been created and improved measures are being developed to better protect digital works through varying combinations of hardware and software. These protection schemes can be implemented at the level of copyrighted work or at more comprehensive levels such as operating systems, networks, or both. Some of the current technological solutions include those listed below.

### 1. Copyright Protection Schemes for Software Programs

These schemes prevent unauthorized duplication of software programs, but they also slow down user computers. More importantly, this form of protection leaves legitimate users with no convenient recourse if the software was lost due to a hard drive crash.

### 2. Encryption

Encryption is the scrambling of data using mathematical principles that can be followed in reverse to decrypt or unscramble the data. File encryption therefore converts a file from a manipulatable file format to a scrambled/encrypted format. Authorization in the form of possession of an appropriate key is required to decrypt the file and restore it to its manipulatable format. Thus, the encryption of software and data is a form of protection against unlawful copying and reuse.

### 3. Digital Signatures/Watermark

Digital signatures use mathematical algorithms to place a seal on a digitally represented work. Generating the digital signature is referred to as signing the work. The algorithms can be implemented through software or hardware, or both. The digital signature serves as means for authenticating the work, both as to the identity of the entity that authenticated it and as to the contents of the file that encodes the information that constitutes the work.

### 4. Steganography

Steganography encodes digitized information with attributes that cannot be disassociated from the file that contains that information. Using the steganography technique, a firm can embed a hidden message in digitized visual or audio data. The embedded information does not degrade or otherwise interfere with the audio or visual quality of the work.

## B. Policies

Information system (IS) and general managers must be aware of the legal liabilities that can incur as a result of internal misuse of data, software programs, and the Internet in order to effectively protect the organization from potential litigation concerning copyright issues. These management responsibilities extend to dissemination of the approved information policy to the organization.

### 1. Code of Ethics for IS Professionals

Oz in the 1992 article "Ethical Standards for Information Systems Professionals" as well as many other authors have called for a unified code of ethics that will cover all IS professionals. Codes of ethics are promises by the profession to regulate themselves in the general interest of society. Most groups of professionals have adopted ethical codes and codes of conduct, which allow them to take on special rights and obligations due to their special claims to knowledge, wisdom, and respect. Physicians, lawyers, and engineers have moral responsibilities and know to whom and for what they are responsible. Professionals in the IS field need similar guidance. These professional groups take responsibility for the partial regulation of their professions by determining entrance qualification and competence. Because computer laws did not exist when computers where initially introduced, professional organizations initiated their own ethical codes. Although some individual IS professional organizations have established professional standards in the field, not all IS professionals are bound by the same set of rules.

### 2. Code of Conduct for Organization

Establishing corporate conduct policies that include IS concerns is a way in which to ensure that copyright protections are adhered. Managers need to be responsible for developing, implementing, and enforcing corporate conduct policies. Historically the IT area was the last to be considered with first priority going to financial integrity and personnel policies. In light of the current quandary of protections problems, it has become clear in the last four decades that organizations need stricter policy statements for their IT covering issues of privacy, ownership, accountability, data quality, and system security.

## XII. ACCOUNTABILITY, LIABILITY, AND CONTROL

New information technologies and the Internet are challenging the existing liability laws and social practices for holding individuals and institutions accountable. Questions that often arise are: How will traditional intellectual property rights be protected in a digital society in which tracing and accounting for ownership is difficult and ignoring such rights is so easy? Who can and will be held accountable for the harm done to individual and collective information and property rights? What standards of data quality and system security should be demanded to protect individual rights and the safety of society?

Information system managers are faced with these difficulties since they will be ultimately responsible for the harm done by the organizations with regards to these copyright issues. Companies need to show more responsibility with internal policies that expressly state that users must treat all intellectual material, including that on the Internet as though it is copyrighted, even if it does not have the © symbol affixed. End users sometimes assume that material is not copyrighted if they do not see a specific notice of copyright. They may conclude that, ethically, it is permissible to help themselves, but this is not the case. According to the U.S. Code: Title 17, § 401(a):

> Whenever a work protected under this title is published in the United States or elsewhere by authority of the copyright owner, a notice of copyright as provided by this section *may be* placed on publicly distributed copies from which the work can be visually perceived, either directly or with the aid of a machine or device.

Therefore copyrighted material no longer need include notice of copyright by use of © or even the word "copyright." Copyright protection exists from the time the work is created in fixed form. That is, United States law does not require copyright warnings in order to be protected from infringers; the notices one sees are simply reminders. Organizational guidelines should also include rules preventing the download of software without prior permission from the IT department.

## A. Control

Organizations should limit or control access to the source of a work; limit the reproduction, adaptation, distribution, performance, or display of work; identify attribution and ownership of a work; and manage or facilitate copyright licensing. Organizations should rely on a variety of technologies, based in software and hardware, to protect them against unauthorized uses of the intellectual property. A balance must be achieved that ensures that there is effective protection of copyrighted material that does not unduly burden use of the work by consumers or compromise their privacy.

Controlling access to the source of the work, information, or data server can also regulate distribution of digital works. Access can range from complete uncontrolled access to completely controlled access. This access control should be implemented through user identification and authentication procedures that deny access to unauthorized users.

## SEE ALSO THE FOLLOWING ARTICLES

Crime, Use of Computers in • Desktop Publishing • Ethical Issues • Internet, Overview • Law Firms • Security Issues and Measures • Software Piracy

## BIBLIOGRAPHY

Bielefield, A., and Chessman, L. (1997). *Technology and copyright law: A guide for the library, research and teaching professions.* New York: Neal-Schuman Publishers Inc.

Congress of the United States, Office of Technology Assessment. (1986). Intellectual Property Rights in an Age of Electronic and Information.

Digital Millennium Copyright Act. http://www.gseis.ucla.edu/iclp/dmcal.htm

Gerhardt-Powals, J., and Powals, M. (1999). The digital millennium copyright act: An international assault on fair use? *Proceedings of the ACM ITiCSE '99,* Cracow, Poland, 191–196.

Johnson, D., and Post, D. (1996). Law and borders—The rise of law in cyberspace. *Stanford Law Review,* Vol. 48, 1367–1398.

Koppius, O. R. (1999). Dimensions of intangible goods. *Proceedings of the 32nd Hawaii International Conference on Systems Sciences,* Maui, Hawaii.

Kurz, R. (1996). *Internet and the law: Legal fundamentals for the internet user.* Rockville, MD: Government Institutes Inc.

Oz, E. (1992). Ethical standards for information systems professionals. *MIS Quarterly,* Vol. 16, No. 4, 423–433.

Rosenoer, J. (1997). *Cyber law: The law of the internet.* New York: Springer-Verlag.

Samuelson, P. (1999). Good news and bad news on the intellectual property front. *Communications of the ACM,* Vol. 43, No. 2, 19–24.

Shapiro, C., and Varian, H. (November–December 1998). Versioning: The smart way to sell information. *Harvard Business Review,* pp. 96–114.

Straub, D. W., and Collins, R. W. (June 1990). Key information liability issues facing managers: Software piracy, proprietary databases, and individual rights to privacy. *MIS Quarterly,* pp. 143–156.

Straub, D. W., and Welke, R. J. (1998). Coping with systems risk: Security planning models for management decision-making. *MIS Quarterly,* Vol. 22, No. 4, 441–469. Copyright Act, U.S.C.A. 17.

Yeung, M. M. (1998). Digital watermarking. *Communications of the ACM,* Vol. 41, No. 7, 30–33.

# Corporate Planning

**Robert J. Thierauf**
*Xavier University*

## GLOSSARY

**business intelligence** Gives the manager a better understanding of the area under study by employing on-line analytical processing (OLAP) functionality, data warehousing, and data mining techniques.

**corporate planning** Centers on the planning of a company's activities from the short range to the long range that is necessary to realize a company's mission and purpose that are related to its corporate objectives and its measurable goals.

**critical success factors (CSFs)** Factors that are critical in making or breaking the organization from the short run to the long run.

**disruptive changes** Those factors that are present in an industry that may cause a company to experience major problems from the short range to the long range.

**e-commerce** Provides electronically direct access to new and established markets, thereby cutting costs for both buyers and sellers through the elimination of unnecessary paperwork and processes.

**executive visioning** The capability of top-level executives to envision the future and its impact on a company's operations.

**key performance indicators (KPIs)** A way of formalizing and measuring a company's critical success factors that are essential to its success and can be related to a company's financial ratios.

**long-range corporate planning** A process that provides the means for articulating a clear mission and purpose of an organization which provides the overall direction for setting appropriate corporate objectives and goals, employing corporate strategies and programs, and allocating an organization's resources optimally over time.

**medium-range corporate planning** A derivative of an organization's long-range corporate plans that centers on a two- to five-year time frame.

**multidimensional analysis** The capability to look at different dimensions of the same data to better understand a company's operations.

**problem finding** The process of going out into the future and determining potential problems and identifying future opportunities that are related to them.

**scenario planning** A tool used by corporate planners for utilizing information and knowledge to foresee what is ahead under good, average, and poor economic conditions.

**short-range corporate planning** A detailed financial plan that specifies both how the company's objectives and measurable goals for the coming year are to be attained and the operational procedures for managing operations.

**strategic intelligence** Centers on *understanding* the total picture of where the organization is headed today and tomorrow and is linked to executive visioning in a changing world.

## I. INTRODUCTION TO MORE EFFECTIVE CORPORATE PLANNING USING STRATEGIC INTELLIGENCE

A number of leading futurists predict that the 21st century will be times of great change. Many of the anticipated changes are not minor perturbations, but major adjustments in business and social environments. Several of the driving forces behind these changes are global competition, the continual restructuring of business organizations, the aging of the United States population, continued variations in the inflation (deflation) rate, the volatility of the stock markets, globalization of capital markets, periodic energy shortages, and accelerating technological changes of all types. Future decisions will involve more complex ones than in the past and, to be effective, must merge together both quantitative and qualitative analyses. Solving the problems of the future and developing new opportunities for the typical company requires the use of advanced computer systems to provide top-level managers and their staffs with a more effective approach to corporate planning.

From this broad perspective, managers need to employ *strategic intelligence* as found in current business intelligence systems (BISs). Such systems center on integrating core data, information, and knowledge with relevant contextual facts to detect significant events and to illuminate forthcoming problems and promising opportunities. In effect, these systems assist corporate managers in getting to know what their customer needs are before they do. Their focus is on fulfilling the strategic requirements of corporate managers at the highest level. Strategic intelligence (as discussed in a separate section below) centers on a *broad understanding* of what it takes to develop and implement appropriate corporate objectives, goals, strategies, and programs over the short to long term. In addition, BIS provide the ability to monitor business trends; to make intelligent strategic, tactical, and operational decisions based on uncertain judgments and contradictory facts; and to allow the organization to adapt quickly as situations change. Essentially, they rely on the exploration and analysis of related and unrelated data, information, and knowledge to provide relevant insights, identify trends, and discover opportunities.

## II. CORPORATE PLANNING DEFINED

For a company to be successful today, there must be a continuous pursuit of a company's mission and purpose that provides the means for setting ambitious corporate objectives and measurable goals and the concentration of a company's competitive energies and action for achieving them. A company's mission statement provides the operational substance for a company, that is, who the customers are, what customer needs are being met, and how the company can fulfill those needs. Related to a company's mission is its purpose for being in existence, that is, what useful need does it provide for its customers. In terms of corporate objectives, they serve as guidelines for action. More importantly, they serve as a standard for measuring performance and can provide incentives for employee performance. As such, most objectives take the form of corporate goals that are attainable, quantifiable, and measurable. Furthermore, every business needs both strategic objectives and goals as well as financial ones. A strategic orientation might include a larger market share, quicker design-to-market times, superior customer service, and the like while a financial orientation might include revenue growth, earnings growth, and stronger cash flows. In turn, strategic and financial objectives and goals can be set forth as specific strategies and programs that are needed to allocate a company's resources in an optimum manner.

Typically, a corporate planning approach should challenge a company to stretch itself and visualize what it might be. For a small-sized company, it might be to dominate a market niche or become a market leader. As will be seen in this chapter, corporate planning at the upper level, also known as strategic planning, is more about breaking down a company's mission and its purpose along with its objectives and goals into corporate strategies and programs that optimize a company's resources. It also includes the consequences that are expected which are articulated in the form of plans and reports, i.e., budgets. From this perspective, corporate planning centers on financial measurement over time. An important element of corporate planning is the improvement of a company's financial status over time. Overall, corporate planning can be defined as the appropriate planning of a company's activities from the short to the long range that is necessary to realize a company's mission and purpose that are related to its corporate objectives and its measurable goals and, in turn, to its strategies and programs for the optimum allocation of a company's resources.

Underlying effective corporate planning is *problem finding* where there is a proactive approach such that external- and internal-environmental factors that affect the organization, from the short to the long range, are taken into consideration. Essentially, top-level managers and corporate planners need to identify potential problems in the future and bring them back to the present time for resolution. Problem finding tends to center on finding solutions to problems that may be very difficult to solve. Going one step further, managers need to identify opportunities that are related to future problems. As in the past, top-level managers and corporate planners must also be involved in problem finding to allocate and use the organization's resources effectively.

An important benefit from problem finding is that, if warned early enough to take corrective action, a manager can prevent a mole hill from becoming a mountain. What is it worth to a manager to be warned of a business problem sometime sooner? What it is worth to the manager and to the company is the avoidance of a crisis. Many times, it is too late to react to problems that are already out of control. This advantage from using problem finding, while difficult to quantify, is very significant and real for the typical company today.

## A. The Essence of Strategic Intelligence

Today, an integral part of corporate planning is *strategic intelligence*. While the word *strategic* refers to or pertains to strategy, *intelligence* means understanding. Within this context, strategic intelligence centers on understanding the total picture of where the organization is going today and tomorrow. It is also linked to executive visioning in a changing world. Strategic intelligence is a forward-looking perspective and an articulated vision of direction that a company should take at the appropriate time and place. As such, it is a guiding force that allows corporate managers the ability to keep their hands on the pulse of the business every step of the way. Because strategic intelligence occurs at the highest level, it is oriented toward many sources based outside the organization. In addition, a relationship exists between a company's critical success factors (those factors that are critical to a company's success), which are related to outside factors germane to a specific industry and being successful in that industry. In turn, these factors are helpful in assisting top managers and their corporate planning staff in determining what strategic direction the company should take today and, more importantly, tomorrow. Underlying strategic intelligence is problem finding (as set forth above) that centers on solving future problems today and identifying future opportunities.

# III. THE ESSENTIALS OF CORPORATE PLANNING IN A CHANGING WORLD

Because *change can be the engine of growth,* the challenge lies not in embracing this business tenet, but in anticipating, adapting to, and generating fresh ideas that exploit change. Because corporate planning is a logical means for adapting to change, it centers on setting or changing organization objectives and goals as deemed appropriate, obtaining the resources to meet these objectives and goals, and determining the strategies and programs to govern the use and disposition of these resources. Because it occurs at the highest level and is related directly to top-level executives and their corporate planning staff, the appropriate external and internal factors are merged that are critical to setting the proper present and future direction for the organization. In turn, it provides input for lower and middle-level managers. Some of the important essentials underlying corporate planning are given in this section.

An integral part of the essentials of corporate planning are long-range followed by medium-range, and finally short-range corporate plans. Current and future products and services are basic to *long-range corporate planning*. A distinctive characteristic of this highest planning level is the use of marketing facts to discover opportunities, and then develop effective strategies and programs to capitalize on these opportunities. Similarly, the focus is on bringing future problems back to the present time for solution. Long-range corporate plans which embrace all aspects of the organization and its environment provide a basis for more detailed medium-range corporate planning.

*Medium-range corporate planning*, sometimes called tactical planning, is concerned with financial planning to place the organization in the best financial position for the next several years. This fiscal planning involves developing the operating programs and associated budgets for the next several years. On the other hand, *short-range corporate planning* or detailed operational planning is related to the financial plans of the current year only. For an organization that has practiced formal planning on a regular basis, it is normal for every major functional area to prepare annual plans for the coming year. Essentially, this financial planning is brought together from a detailed examination of the key measures of the business, such as product-line profitability, variable and fixed costs, inventory turnover, manufacturing capacity, and financial ratios for the coming year.

## A. Using Strategic Intelligence to Make Sense Out of Chaotic Times and Disruptive Changes

To help the typical company make some sense out of today's chaotic times, such as the velocity and volatility with which trade, capital, and currencies move around the globe, decision makers need to employ strategic intelligence to its fullest. The American chief executive officer of Britain's Cable & Wireless notes that "the Chinese character for crisis combines characters for danger and opportunity." There is promise as well as menace. The best businesses can do is to apply several approaches to make their companies winners in the accelerating shakeout of industries around the world.

For one approach, a company can intensify its intelligence gathering both internally and externally. In this tumultuous period, using published statistics to guide decisions is not the way to go. By the time employment figures rise or fall or retail sales rise or fall, it is too late. Typically, decision makers possess valuable untapped marketing intelligence within their own companies, if only they can get to it. The CEO of Du Pont holds a biweekly phone conference with 20 top managers around the globe to stay abreast of changes in customers, competitors, and local economies and politics. He asks different, pointed questions each time: What is happening to customers and their customers? What is the direction of local leaders to deal with the downturn? What should be done now to meet changing competitive rules? The sessions are not just for the CEO's enlightenment, but also for others. By hearing the answers from their peers, they broaden their perspective of the global landscape.

Another approach is to seize new opportunities created by the crisis and, at the same time, stay relentlessly on strategy since all ships are being tossed by the same storm. The unsound may run into trouble and that can open up opportunities if a company is alert. At least three major financial services companies—AIG, Travelers, and Merrill Lynch—have acquired distribution systems in Japan as that country's finance industry has struggled. Cargill has been trying to crack the Japanese market for 30 years. Now the troubles of a Japanese competitor may finally give it a chance: Cargill announced recently that it would buy Toskoku, a Japanese food-trading company that has filed for bankruptcy. Tough times can also be ideal for forging new business relationships that will serve long-term goals that previously just had not seemed urgent. Current conditions have led Orion Capital, a Connecticut-based property and casualty insurer, to investigate strategic partnerships with several insurance carriers, distributors, and service firms.

Still another approach is tightening up operations. There is no substitute for good business fundamentals—customer satisfaction, cost, quality, cycle time, and brand. Some ways of achieving them are smarter than others. For example, Du Pont has centralized and streamlined its efforts to combine purchasing by diverse units around the world to get better prices from vendors—a process it calls "vendor convergence." Managers who see an opportunity to combine purchasing with another unit decides whether it would be the right move for all the businesses involved. They have ultimate authority, which they normally would not have to make these deals happen.

Tied in with making sense out of chaotic times is the area of *disruptive changes* in a typical company's markets. Disruptive changes can be caused not only by a number of factors that related to a company's markets, but also can be related to a company's size per Clayton Christensen. When a company is young, its personnel, equipment, technologies, brands, and the like define what it can and cannot do. As it becomes more mature, its abilities, for example, stem more from its processes, such as product development, improved manufacturing, and financial capabilities.

Because companies, independent of the people within them, have capabilities, those capabilities also define disabilities. As a company grows, what it can and cannot do becomes more clearly defined in certain predictable ways. In the largest companies, values, particularly those that determine what are its acceptable gross margins and how big an opportunity has to be before it becomes interesting, define what the company can and cannot do. Because resources are more adaptable to change than processes or values, smaller companies tend to respond to major market shifts better than larger ones. Hence, it is suggested that companies capitalize on opportunities that normally do not fit in with their processes or values. The bottom line is that all companies start with an understanding of what they are capable of doing and create a framework that managers can use to assess the abilities and disabilities of their organizations as a whole.

## B. Use of Strategic Intelligence in Executive Visioning

An important element underlying strategic intelligence from the short range to the long range is *executive visioning*. Many times, executive visioning is tied in with problem finding. An executive view entails farsightedness along with the eagerness to look ahead from a practical viewpoint. Effective executive visionaries are not necessarily those who can predict the 21st century and beyond accurately. Rather, they are decision makers who can draw a conceptual road map from where the company is now to some imagined future, who can say, “This is how we get there.” Visioning implies a change from the status quo, which helps explain why visionaries are overrepresented in the ranks of entrepreneurs, and why they come in handy to an organization in deep trouble—think of Mr. Lee Iacocca saving Chrysler. Vision is not for the complacent. While the executive visionary sees things a bit differently, the individual is no mystic. The person’s sources of information are down to earth—customers and suppliers, for example—and extend beyond his or her gut-level feelings. The most visionary executive can take in large amounts of information and knowledge, and not just from inside himself or herself.

Vision by itself is not enough for the executive to possess. The executive visionary must be able to communicate what he or she has dreamed, and the company must have the required skills needed to *execute* it. The leaders of the organization must act consistently with the vision in everything they do. Too often in the past, top-management teams work up a statement of corporate vision, promulgate it, and then think their work is done. What they overlook and what dooms this kind of superficial effort is the need to plan and manage this vision over time. A BIS, backed up by appropriate information systems, is an excellent vehicle for assisting in the fulfillment of carrying out the executive vision. Executive visioning and its tie in with long-range corporate planning will be noted later in the chapter.

## C. Capitalizing on E-Commerce Via the Worldwide Internet

A most important strategy for the long run is centered around electronic business technologies, which have changed the ways of the business world. E-commerce provides direct access to new markets, strengthening relationships with customers and other business partners; cutting costs by eliminating unnecessary paperwork and processes; and empowering employees with better education, communication, and information access. All elements of a company’s supply chain, for example, can linked via E-commerce. If a person buys a pair of Nike running shoes, that information can be transmitted back to the plant in Taiwan where the shoes are made and the shoes Fedexed directly to the person’s home. Similarly, items purchased electronically from several suppliers in the same region could be shipped in a batch using electronic interfaces. A whole host of intermediaries in the chain who used to make their living coordinating and moving information can be eliminated. Intranet-to-intranet communication is blurring the lines between companies. The notion of where a corporation starts and stops is very different today. As another example, a company’s expertise might be harvesting timber or processing lumber, but the company also needs to move its products to the construction industry. Traditionally, all those steps were brought together in a soup-to-nuts operation. Now, given interconnectivity, someone else can run a company’s truck fleet, and it will still operate like one’s own fleet.

In the future, the greatest share of E-commerce revenue is not going to high-profile retail web sites, such as amazon.com and eBay. These are consumer commerce ventures, selling one product at a time to individual customers. While an admittedly huge market, it is not the largest. The bulk of E-commerce revenues will come from transactions between businesses, in which both purchase volumes and dollar values already dwarf the consumer commerce market. The business-to-business E-commerce market presents an opportunity so big and appealing that there will be many mini-monopolies. The vastness of these opportunities is what has spurred interest in on-line industry marketplace vertical markets in which companies within a given industry can easily buy and sell goods and services with one another.

An example, Chemdex (www.chemdex.com) is an on-line industrial chemical marketplace that lets buyers and sellers of industrial reagents more easily find one another, look up information about fh6 chemicals they need, and make on-line purchases. What it takes to make a good marketplace is an E-commerce infrastructure tailored to the particular needs of the industry’s buyers and suppliers, and a rich supply of information about the products available throughout the industry. This means that entrepreneurs with intimate knowledge of specific industries have a great opportunity to create such marketplaces and cash in on the trade they generate.

## D. Using Corporate Planning Software for Strategic Intelligence

Today, a wide range of corporate planning software for strategic intelligence is available to top management and its corporate planning staff. Typical business planning software packages include the following. Avantos ManagePro helps managers plan and track goals and progress while fostering success through focused management activities. It includes goal planning and tracking tools, such as the Top

Level Goal Planner, for planning and delegating key business objectives, strategies, and tactics. The Goal and People Status Boards enable managers to monitor the status of primary business goals and obtain at-a-glance reinforcement of where to focus attention. Jian BizPlan Builder is a complete business plan template on disk with more than 90 typed pages of example text that are formulated into word-processing files. Templates use standard spreadsheet applications like Excel or Lotus 1-2-3 to calculate financials and generate graphs. The step-by-step format guides the user through, explains issues, and gives clear and sensible advice.

Once a company's business plans have been developed (from the short- to the long-range), software can be employed that is useful in determining the company's overall financial performance. Such software goes beyond monthly actual versus budget reports that are routinely produced by today's information systems. Currently, companies are employing the use of *corporate scorecard software* which is a sophisticated business model that helps a company understand what is really driving its success. In effect, it acts like the control panel on an airplane, that is, the business equivalent of a flight speedometer, odometer, and temperature gauge all rolled into one. A corporate scorecard keeps track of a company's financial progress as well as its softer measurements. These range from customer satisfaction to return on investment that needs to be managed to reach a company's final destination, i.e., profitable growth. A scorecard, for example, might graph customer service to determine if it is improving or deteriorating and, at the same time, tally product defects to determine if they are rising or falling and where. Going a step further, scorecard software, which is usually distributed throughout a company's computer network, lets company employees across the entire organization be certain they are talking about like items when they get together. If customer satisfaction is declining, sales, manufacturing, and research and development will all be reading the same score, and thus will be able to tackle the problem from a common ground perspective.

Today, there are two views on utilization of scorecard software. One is that a company's yardsticks should be purely financial. Managers should employ indicators like revenue growth and return on investment to guide a business. The other is that a corporate scorecard should be balanced. A company, for instance, not only should keep close watch on performance numbers like gross-profit margins on new products, but also should use softer measurements. Other softer measures include the number of new products, product development cycle times, and the rate of on-time deliveries. The bottom line from this view is that it forces a company to evaluate critically those drivers that really determine its performance over the short run to the long run.

An example of *knowledge discovery software* that can assist management and its corporate planning staff is KnowledgeX. This software discerns hidden relationships among people, organizations, and positions. It is able to take information, organize it, and extract from that material the connections and webs of relationships. Using Intelligent Knowledge Exchange, KnowledgeX can perform a text processing task called "SmartParse" to do the selection and extraction of textual information. If it encounters terms it already knows, it will automatically assign the correct set of connections to that term. If it does not know the term, the user can define it once and all future uses of that term will be connected according to rules. KnowledgeX is object oriented. It allows multimedia information to be accessible from within the program.

Once the relationships are established, KnowledgeX has the ability to display these connections in a graphical way or as a text outline. The graphical display is easy to grasp and often reveals relationships that had been buried within the text and not obvious to the researcher. By creating maps showing relationships among customers, companies, individuals, and organizations, KnowledgeX makes it easy to grasp how these people and organizations are interconnected. The more information that the knowledge base contains, the better that KnowledgeX is at finding unknown and unnoticed relationships.

In addition, KnowledgeX can combine information entered by many people on the network and find the connections within that information. As such, fragments coming from other people can be converted into something like a centralized knowledge source. That is, KnowledgeX takes information and knowledge coming from many sources and reveals relationships that had never been known before because the pieces were too fragmented and dispersed among too many people. From this perspective, this knowledge discovery software is quite useful in bringing together diverse elements needed for resolving strategic planning issues.

## IV. APPLICATIONS THAT LEND THEMSELVES TO CORPORATE PLANNING

In order to understand corporate planning applications, reference can be made to either a manufacturing-oriented company or a service-oriented company.

For either type company, corporate planning is linked directly with a company's other activities, that is, planning activities are related to marketing, manufacturing, and finance that are linked to human resources. All of the external- and internal-environmental factors related to these company activities form the basis for short- to long-range corporate planning applications.

Within this broad-based framework, the essential focus of the applications covered in the next sections focus on short-, medium-, and long-range corporate planning. As a starting point for long-range corporate planning, executive visioning can be used by top-level managers to build a consensus around what it plans to accomplish and guides action at the lower and middle-management levels. Typically, top management is assisted by a corporate-planning staff who operate in an environment characterized by numerous product and technological change. The task of the corporate-planning staff is not only to prepare short- to long-range corporate plans using strategic intelligence, but also to monitor current results against these plans. If necessary, the corporate-planning staff should provide *feedback* to top management when targeted plans are not being met. Feedback gives top management the capability to study alternatives for improving an organization's operations.

## V. LONG-RANGE CORPORATE PLANNING

Within a BIS environment, corporate planners view long-range corporate planning as a *continuous process* over the long term (five years and beyond) that takes into account a relevant business model for the times. This process, shown in Fig. 1, begins with executive visioning (based on a current business model) that provides the means for articulating a clear mission and purpose for the organization. This overall direction provides the means for setting appropriate corporate objectives and measurable goals, employing strategies and programs to achieve these desired objectives and goals, and allocating a company's resources in an optimum manner. In turn, this chain of events provides a basis for the preparation of long-range corporate plans. Typically, many of the prior events have already been set in place in prior years. Hence, planners need to make adjustments before the preparation of the corporate plans. To assist in the preparation of final long-range corporate plans, there is generally a need to evaluate the plans using different sets of scenarios based on good, average, and poor economic conditions. This area is covered below under the tie in of a relevant business model for the times to scenario planning.

Once long-range corporate plans have been developed such that a company's resources have been allocated in an optimum manner, appropriate critical success factors (CSFs) are specified, as shown in Fig. 1. The process of identifying CSFs was originally defined by John Rockart of MIT. Typically, these factors have been set forth previously and may need to be updated to the present times. For example, a corporate-planning staff has determined in the past its CSFs to be: (1) price (responsiveness to competitive pricing); (2) cost control (reducing the cost of plant and office operations); (3) inventory turnover (improving the times the inventory turns over yearly); and (4) product mix (having the right products for the times). A current review of the company's operations indicates that selling on the Internet as well as buying from outside vendors on the Internet has become a critical success factor. Hence, it needs to be added at this time along with possible adjustments to the four mentioned above. In a similar manner, key performance indicators (KPIs) and financial ratios per Fig. 1 may have to be modified for the current times. KPIs and financial ratios can be developed not only over the long term, but also as important measures for current operations, especially when comparing budgeted figures to current actual amounts.

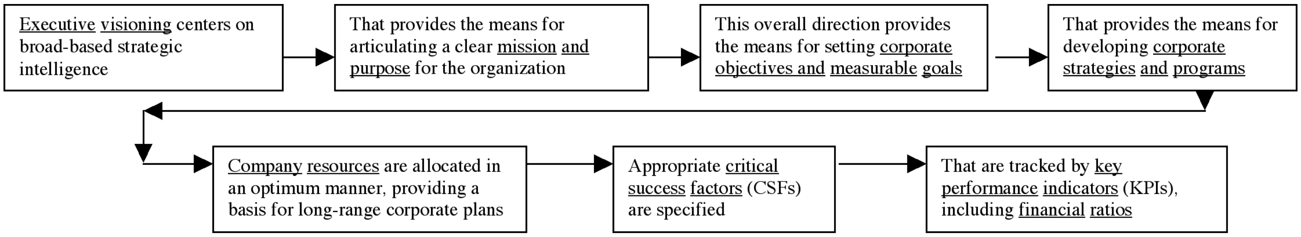

Executive visioning centers on broad-based strategic intelligence
That provides the means for articulating a clear mission and purpose for the organization
This overall direction provides the means for setting corporate objectives and measurable goals
That provides the means for developing corporate strategies and programs
Company resources are allocated in an optimum manner, providing a basis for long-range corporate plans
Appropriate critical success factors (CSFs) are specified
That are tracked by key performance indicators (KPIs), including financial ratios

**Figure 1** Long-range corporate planning process—from executive visioning to mission and purpose to corporate objectives and measurable goals to company resources that provide a basis for preparing long-range corporate plans which, in turn, allows for specifying a company's critical success factors that are tracked by key performance indicators and financial ratios.

## A. Tie In of a Relevant Business Model for the Times to Scenario Planning

To assist top management and corporate planners in getting a handle on their company's operations, a starting point is the relevance of the company's current business model to changing times. Conventional wisdom of the past focused on companies with the largest market shares who, in turn, would have the largest revenues and lowest costs for the highest profits. However, many large companies, like General Motors, IBM, and Sears have found this not to be true. What is happening is that companies are seeing a migration from products and services to what might be called "spinouts." Thus, the newer business model for changing times is centering on spinouts as drivers of profitability. Take, for example, General Electric's jet engine profits are not in building the engines, but in their financing, service, spare parts, and overhauling existing engines. Essentially, manufacturing of these engines has become almost incidental to the total operations for producing profits. The essential message here is that a typical company must change its basic business model to reflect changing times.

Having the appropriate business model in place, *scenario planning* can be utilized by top management and corporate planners to evaluate outcomes under good, average, and poor economic conditions. Basically, scenario planning simplifies the avalanche of information and knowledge into a limited number of possible states. Each scenario tells a story of how various elements might interact under certain conditions. When financial relationships between elements can be formalized, a company can develop appropriate financial statements under good, average, and poor economic conditions. In turn, each scenario needs to be evaluated for internal consistency and plausibility. Although a scenario's boundary might at times be somewhat unclear, a detailed and realistic narrative can direct attention to aspects that would otherwise be overlooked.

Generally, scenario planning attempts to compensate for two common errors in decision making—underprediction and overprediction of change. Most people and companies are guilty of the first error. For example, think in terms of a hundred years ago how hard it was to imagine the factors that propelled society into today's new technological world where automobiles, airplanes, televisions, and computers are commonplace. Yet a small group of futurists overpredicted, expecting levels of change that failed to materialize, notably in medicine, artificial intelligence, and space travel.

Scenario planning allows companies to chart a middle ground between under- and overprediction. It helps expand the range of possibilities that can be seen. It does this by dividing knowledge into two areas: (1) elements that are knowable and (2) elements that are uncertain or unknowable. The first component casts the past forward, recognizing that the world possesses considerable momentum and continuity. Assumptions about demographic shifts and substitution effects of new technologies can safely be made. Obvious examples of uncertain aspects are future interest rates, oil prices, results of political elections, rates of innovation, etc. It is not important to account for all the possible outcomes of each uncertainty; simplifying the possible outcomes is sufficient for scenario planning. Because scenario planning depicts possible future events and not specific strategies to deal with them, top management and corporate planners need to tie in company strategies with these scenarios.

## B. Using Strategic Intelligence to Better Understand Long-Range Corporate Planning

Company resources, as indicated previously, need to be allocated in an optimum manner, thereby providing a basis for developing long-range corporate plans under good, average, and poor economic conditions. To assist in developing these plans, corporate planners should begin with a knowledge and understanding of corporate strategies and programs that are reflected in existing products, services, margins, profits, return on investment, cash flow, availability of capital, research and development capabilities, skills and capacities of personnel, etc. For a manufacturing-oriented company, this intelligence is typically extended to an in-depth analysis of manufacturing operations that are linked to centralized operations at the corporate level. There is need to examine the past few years' and the current year's performance as part of a beginning overall-review process. Evaluating significant aspects of past and present operations is the basis for determining how well the organization objectives and goals are being met. In like manner, plans for the coming five years under different scenarios based on short- to medium-range plans for improving operations become an essential part of getting started on long-range corporate planning.

A typical five-year plan for a manufacturing-oriented company includes the *external-environmental factors* that are generally not controllable, such as customers, government, public, competitors, suppliers, investors, and financial institutions. On the other hand, four *internal-environmental factors* that are controllable by the company center on the following:

1. *Marketing planning* focuses on expanding the present product lines and entering new product markets; also, increasing use of selling outlets and/or distribution to sell the company's products, changes in pricing policy and pricing practices to effect higher sales, and consideration of new advertising media for more effective penetration of the company's markets.
2. *Manufacturing planning* centers on major facilities contemplated and improvements in processing efficiency, including the percent of capacity that is now and will be employed with present facilities and machinery as well as the steps that are being undertaken to use excess capacity.
3. *Financial planning* relates to projected sales by product lines, contribution (sales less direct manufacturing costs) by product lines, indirect manufacturing costs plus sales and general and administrative expenses, net profits before federal income taxes by product lines, fixed and working capital needs, return on investment by product lines, and comparable financial ratios and analyses.
4. *Human resources planning* which is related to marketing, manufacturing, and financial plans centers on projected requirements for key management personnel and production labor when considering turnover and future growth.

To develop projected strategic plans under different scenarios for a five-year period, the corporate-planning staff employs the company's corporate databases and data warehouses to analyze meaningful long-range data, information, and knowledge in order to discover pertinent intelligence about a company's operations over time. In turn, this output is used to finalize its five-year strategies and programs under the most likely scenario. Also, management needs to consider the company's critical success factors. To better understand strategic intelligence as a way of getting a handle on long-range corporate planning, reference can be made to Figs. 2 and 3 for a typical company that are based on three likely scenarios, i.e., good, average, and poor economic conditions. These represent the consensus of top management working with its corporate-planning staff. The dollar amounts for total sales, cost of goods sold, gross margin, selling and general expenses, and profits under good, average, and poor economic conditions five years hence are found in Fig. 2. In this figure, the graph of this data is shown, followed by the pie charts for good and poor economic conditions in Fig. 3. Although not shown, a number of other values, graphs, and pie charts could have been illustrated for the company's geographical regions—north, east, south, and west. Additionally, the geographical regions could have been ranked five years hence. The attendant circumstances will normally dictate what further analyses are necessary for a thorough understanding of a company's operations five years into the future.

A thorough understanding of Figs. 2 and 3 plus related illustrations provides an effective overall long-range strategic measurement framework for a typical company. Ever since Peter Drucker wrote that *what is measured gets done,* decision makers have considered performance measurement an essential part of their jobs. The potential effectiveness of a measurement framework for strategic intelligence needs to consider the following criteria in the form of these questions:

- Is it, or can it be, connected to a company's strategy?
- Is a balance of financial and non-financial measures included? Are key measures tied in with a model which expresses the causal relationship between them so the focus is on key performance factors and financial ratios?
- Can it be linked to important management processes, i.e., planning, management reviews, budgeting, etc.?

Typically, underperforming companies are more often the result of hundreds or even thousands of decisions made by individuals throughout the company than any grand strategic mistakes. Decision results at all levels impact the strategic capacity of the company. Too frequently, people must rely on a variety of generic inputs which are not targeted to their specific needs. An effective strategic intelligence capability must offer the collective history, facts, insights, and applicable analyses of the organization to decision makers. A most difficult part of strategic intelligence is continually undertaking the analyses needed for decision makers and looking at their results so future decision makers can learn from the past. This feedback loop is essential not only to populate the intellectual assets of the organization, but also to enable the BIS to monitor and make adjustments to key underlying business assumptions and rules. Decision makers need to be informed when the company is where it should be and when it is off track. Hence, when companies succeed, they need to know why so that there can be a positive approach to a company's decision-making behavior.

A

| | Good Economic Conditions<br>5% Growth Rate | Average Economic Conditions<br>2% Growth Rate | Poor Economic Conditions<br>3% Recession Rate |
|---|---|---|---|
| Sales | $200,200,550 | $194,450,275 | $184,875,290 |
| Cost of Goods Sold | 128,400,175 | 128,100,450 | 126,450,250 |
| Gross Margin | 71,800,375 | 66,349,825 | 58,425,040 |
| Selling & General Exp | 40,550,400 | 40,550,400 | 40,550,400 |
| Profits | $31,249,975 | $25,799,425 | $14,874,640 |

B

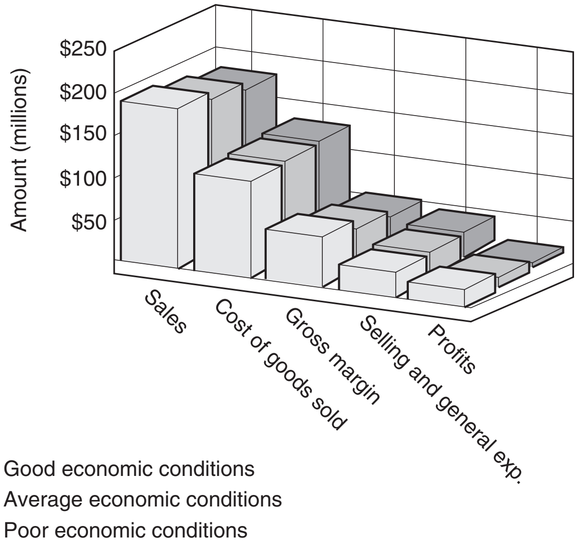


**Figure 2** (A) Total sales, cost of goods sold, gross margin, selling and general expense and profits under good, average, and poor economic conditions five years hence and (B) graph that compares the above amounts five years hence.

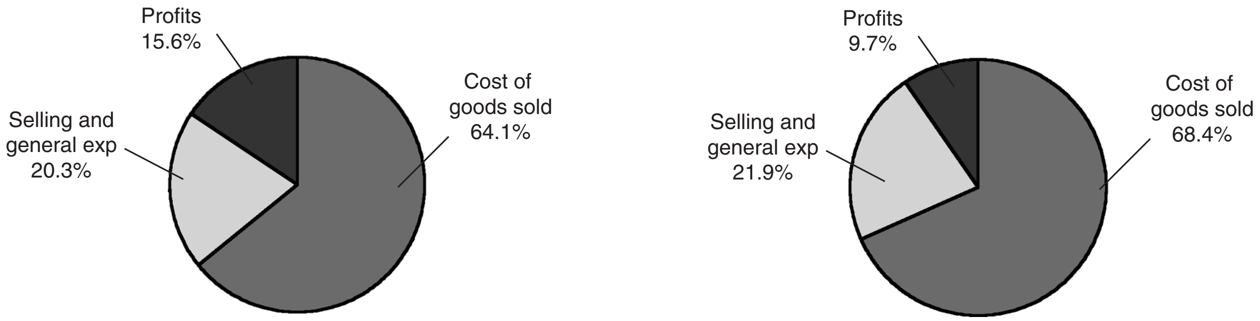


**Figure 3** A pie chart comparison of cost of goods sold, selling and general expenses, and profits under good and poor economic conditions five years hence.

## VI. SHORT- AND MEDIUM-RANGE CORPORATE PLANNING

Short-range corporate planning is a derivative of medium-range corporate planning which, in turn, comes from long-range corporate planning. Essentially, short-range corporate planning is concerned with the efficient use of a company's resources. It is a detailed financial plan that specifies both how the company's objectives and goals for the coming year will be attained and the operational procedures for managing daily operations. As such, the short-range plans outline the specific steps to accomplishing the medium-range plans. Also, they play a major role in implementing business strategies by translating long-range plans into action and take into consideration a company's critical success factors.

Additionally, short-range corporate plans center on detailed objectives and specific measurable goals and strategies of the company and a means for achieving them, usually in the form of flexible budgets or profit plans. Management needs cost and revenue margin information so that it can identify areas of strength and weakness. Knowledge about a company's product margin or contribution information and its competitors over time is also needed to measure the profit impact of alternative courses of action. Approved short-range corporate plans become budgets such that actual results can be measured and compared with them monthly for more effective control. Overall, these plans of a short duration represent top-down planning and budgeting that links performance to strategic vision. Essentially, these short-range corporate plans represent continuous plans that are owned by department heads who will be held accountable for results.

### A. Use of Strategic Intelligence to Develop and Evaluate Short- and Medium-Range Profit Plans

For a typical company's products, an annual profit plan is an integral part of corporation-wide corporate planning. In a similar manner, overall profit plans are determined for two, three, and four years hence. Typical output in terms of short-range reports (current year) include periodic (or monthly) balance sheet and income statements, monthly flexible budgets, monthly budget exception reports, and periodic (or monthly) KPIs and financial ratios. Medium-range reports (2 to 4 years in the future) include projected balance sheet and income statement, projected cash flow, KPIs and financial ratios, projected source and application of funds, and projected products, manufacturing facilities, and personnel requirements. As information becomes available that is reflective of changing times, profit plans for the coming year must be revised to reflect the changes and expected changes in the business environment. Effective profit planning, therefore, must be a *continuous* effort rather than a periodic one.

Although flexible budgets for profit planning are generally prepared by the accounting department, the responsibility rests with the corporate planning staff, who must not only select the appropriate financial information for specific planning decisions, but must also combine this information from corporate databases and data warehouses in useful and meaningful ways. Many times, companies need to employ knowledge discovery techniques to extract financial trends and patterns. Staff members must also review and coordinate the estimates provided by the functional managers involved in a particular decision. They must provide a measure of the profit impact of alternative courses of action and advice on the meaning and significance of financial analysis. In summary, long-range corporate plans for a typical company are translated into medium-range plans for the next several years and finally into short-range corporate plans, i.e., annual profit plans (including budgets) for the coming year.

Flexible budgets or detailed profit plans for the coming year are developed that take into account planning for marketing, manufacturing, and finance as well as human resources that have impact on the current year. Similarly, overall profit plans for medium-range corporate plans can be developed. To assist the corporate-planning staff, a series of "what-if" questions about planning need to be asked and answered where *sensitivity analysis* can be used to determine the impact the change of one or more variables might have on the final profit plans. Managers and their staffs can interact with the company's information and knowledge to answer what-if questions and to undertake sensitivity analysis.

For example, a six-month analysis of five new products can be undertaken that starts with their profits which are as follows:

| Month | Product 1 | Product 2 | Product 3 | Product 4 | Product 5 |
|---|---|---|---|---|---|
| 1 | $42,400 | $62,100 | $65,700 | $ 80,400 | $118,400 |
| 2 | 39,900 | 56,400 | 61,400 | 78,800 | 86,900 |
| 3 | 44,200 | 58,300 | 69,100 | 98,100 | 104,800 |
| 4 | 54,800 | 60,100 | 75,400 | 102,500 | 113,100 |
| 5 | 57,200 | 63,300 | 80,300 | 107,200 | 120,200 |
| 6 | 62,600 | 66,100 | 88,700 | 112,800 | 138,300 |

This data can be graphed (like in Fig. 2). Although not shown, pie charts for a company's best and worst profits in the five new products can be developed (as in Fig. 3). In turn, the profitability of these five new products over the forthcoming six months can be ranked as follows:

| Month | Product 1 | Product 2 | Product 3 | Product 4 | Product 5 | |
|---|---|---|---|---|---|---|
| 1 | $62,600 | $66,100 | $88,700 | $112,800 | $138,300 | Best |
| 2 | 57,200 | 63,300 | 80,300 | 107,200 | 120,200 | |
| 3 | 54,800 | 62,100 | 75,400 | 102,500 | 118,400 | |
| 4 | 44,200 | 60,100 | 69,100 | 98,100 | 113,100 | |
| 5 | 42,400 | 58,300 | 65,700 | 80,400 | 104,800 | |
| 6 | 39,900 | 56,400 | 61,400 | 78,800 | 86,900 | Worst |

Basically, profits increase over the months due to a number of factors, including larger sales volumes due to the acceptance of the product, a reduction in production times, and a reduction in scrappage as the learning curve improves. In addition, the percentages of profitability could have been set forth for the five products along with a graph and an appropriate ranking of the products for a different view.

Related to short-range (and medium-range) corporate plans is the measurement of actual performance against these plans. This can take the form of employing key performance indicators and financial ratios. Also, scorecard software (mentioned previously) can help management at this point. Besides getting an overview of how a company is doing overall, specific areas of a company can be measured and evaluated for its performance. As examples, scorecard software can check on defect rates plant by plant and indicate how each plant's quality is improving. It can link measurements like on-time deliveries to certain financial indicators. The software can measure the percentage of sales due to new product introductions, and gross margins on new products along with corporate-wide indicators like revenues and return on investment. It should be noted that different companies at different times have different needs and aims. For example, a service company that has just merged would not build a strictly financial model that focuses, say, on productivity. Output per employee is not a driver for that business. The real value of scorecard software is that it forces a company to reexamine its assumptions about what really drives performance. It forces a company to focus and become much more explicit about what matters to its customers and, ultimately, what matters to a company's total operations.

## VII. FUTURE CORPORATE PLANNING WILL MOVE TOWARD GREATER USE OF STRATEGIC INTELLIGENCE

In the future, the employment of strategic intelligence in corporate planning is expected to increase for a number of reasons. First, a company needs to think globally, then act locally in response to the local markets it serves. Second, sharpening a company's competitive intelligence is needed for growth and survival in the long run. Third, the impact of e-commerce on a typical company is expected to increase at an unprecedented rate. Fourth, the average company is in the early stages of a total revolution in organization structures, that is, companies may be run as virtual corporations. Fifth, there may well be a shift away from core competence where the focus will be on figuring out where the opportunities for a company lie. Sixth, there will be an open atmosphere where innovation is encouraged among all company employees. Still many other reasons can be given for the shift to strategic intelligence in corporate planning matters.

As with present corporate planning, decision makers at the very top need to have basic intelligence to assess the overall health of the company. Armed with that intelligence, they are ready to ask questions about a company's future direction. Knowing what questions to ask about a company's future direction is most important when retrieving strategic information and knowledge from the company's databases and data warehouses. For example, a decision maker can ask questions about the company's future profitability and receive information about the profitability or sales volume of the company as a whole. If the individual is satisfied, then the decision maker would go on to the next metric of interest. If something seemed out of line, for example, sales volume exceeded expectations, then the decision maker could look further into the sales volume measures, possibly requesting a breakdown by product line or division. In turn, a number of graphs, pie charts, and rankings of products along with their sales amounts and percentages can be presented to the decision maker for a better understanding of a company's future performance, i.e., future strategic intelligence.

## SEE ALSO THE FOLLOWING ARTICLES

Decision-Making Approaches • Decision Theory • Economic Impacts of Information Technology • Enterprise Resource Planning • Operations Management • Productivity • Reengineering • Strategic Planning for/of Information Systems • Value Chain Analysis

## BIBLIOGRAPHY

Ackoff, R. L. (1999). *Re-creating the corporation: A design of organizations for the 21st century.* New York: John Wiley.

Brown, S. L., and Eisenhardt, K. M. (1998). *Competing on the edge: Strategy as structured chaos.* Boston, MA: Harvard Business School Press.

Christensen, C. M. (1997). *The innovator's dilemma: When new technologies cause great firms to fail.* Boston, MA: Harvard Business School Press.

Colis, D. J. (2001). *Corporate strategy,* Second Edition. New York: McGraw-Hill.

Drucker, P. (1999). *Management challenges for the 21st century.* New York: HarperBusiness.

Godet, M. (1987). *Scenarios and strategic management.* London: Butterworths.

Kaplan, R. S., and Norton, D. P. (1996). *The balanced scorecard: Translating strategy into action.* Boston, MA: Harvard Business School Press.

Mintzberg, H. (1994). *The rise and fall of strategic planning: Reconceiving roles for planning, plans, planners.* New York: Free Press.

Siegel, D. (1999). *Futurize your enterprise: Corporate strategy in the age of the e-customer.* New York: John Wiley.

Thierauf, R. J. (1987). *A problem-finding approach to effective corporate planning.* Westport, CT: Quorum.

Velia, C. M., and McGonagle, Jr., J. J. (1988). *Improved business planning using competitive intelligence.* Westport, CT: Quorum.

# Cost/Benefit Analysis

**David L. Olson**
*University of Nebraska, Lincoln*

## GLOSSARY

**checklist** A means of evaluating initial project proposals in terms of a list of expected characteristics.

**contingent valuation method** Survey techniques asking for willingness to pay or to accept for a qualitive feature.

**discounted cash flow** Conversion of cash flows into their worth as of some base period, usually the present.

**discount rate** The rate used to convert future cash flows into their current worth, usually set at the firm's marginal rate of return (marginal value of money).

**internal rate of return** That marginal value of capital that yields a net present value of zero for a stream of cash flow.

**net present value** The worth, in today's terms, of a stream of cash flow over time, assuming a given marginal cost of capital.

**payback** Financial analysis estimating the time required to recover an investment.

**project profile** A display of expected project performance on important criteria, allowing comparison of alternative projects.

**screening** A method of implementing a checklist for project evaluation by listing the worst acceptable performance on a list of criteria.

**SMART** Simple multiattribute rating theory, a means of objectively comparing the values of projects considering multiple criteria.

**value analysis** Analysis of a proposed project by comparing subjectively described benefits versus estimated costs.

**COST-BENEFIT ANALYSIS** is presented, including demonstration of the method. In practice, complete information is often not available for full cost-benefit analysis, and the simplification of payback is often used to provide decision makers with a quick picture of investment opportunity. Some other alternative evaluation methods for information system project evaluation are also presented. The role of the time value of money is discussed, as well as the treatment of intangible factors.

Cost-benefit analysis seeks to identify accurate measures of benefits and costs in monetary terms, and uses the ratio of benefits to costs to provide a metric for evaluation of project proposals. Economists would view this calculation from the perspective of society. The same approach is often applied from the perspective of a firm. A ratio of benefits to costs greater than 1.0 indicates a positive investment, while a ratio less than 1.0 indicates that the investment will result in a loss. Net present value is usually the most important financial criterion used. Cost-benefit analysis and other financial/economic evaluations of project desirability are important considerations in any investment decision. However, there are limitations and weaknesses in cost-benefit analysis. While traditional cost-benefit and net present value tools are useful in evaluating information system projects, managers have recognized that rational decision making requires expansion of techniques used to recognize information technology's extensive impact on the organization, its customers, and its competitors. Financial measures alone are seldom sufficient to support decisions involving long-term impact. Therefore, other techniques are often applied.

# I. COST-BENEFIT ANALYSIS

Information system project proposals are generated in great volume. This presents management with the very important and difficult decision of whether to fund specific project proposals. The seemingly most accurate and precise methodology to evaluate project proposals is cost-benefit analysis.

While cost-benefit analysis is a valuable tool in selection of information system projects, it requires data accuracy usually beyond what is available. Therefore, in practice, many other approaches are used. Studies describing practice are reviewed, and some of the alternative methods used are demonstrated. For projects involving long time frames, considering the net present value of benefits and costs is important.

## A. Cost/Benefit Example

Assume that an automation project has been proposed to improve an organization's methods of providing service to customers. Currently this operation is accomplished by a staff of 12 people with a total payroll expense of $400,000 per year (including benefits). The proposed project would reengineer the system and only require eight staff (at a little more pay), yielding a personnel cost at the current rate of $320,000 per year. The project would take an estimated year to develop and install (including training of staff). Project costs include $30,000 for software obtained from vendors, $50,000 to vendor consultants, and $200,000 for internal staff to integrate the new software into the existing system. Training staff would cost $30,000 in staff time. The benefits of the proposed project are expected to be obtained for five years after project completion. Inflation for staff payroll expenses for this operation have consistently been 2% per year. The company has a marginal value of capital of 15% per year.

The cost-benefit calculation for the new project requires identification of benefits in monetary units. Use of net present value requires identification of the timing of monetary exchanges. The benefit from the new project consists of lowered personnel costs. A comparison of the old operation versus the new is given in Table I. The increased contribution to profit per year would be $424,650 over years 2 through 6 (Table II). Development and training costs would amount to $310,000, incurred in the first year.

**Table I** Simple Cost-Benefit Calculation (in Dollars)

| | Old process | New process | Benefit |
|---|---|---|---|
| **Benefits:** | | | |
| Personnel | | | |
| Year 2 | 408,000 | 326,400 | 81,600 |
| Year 3 | 416,160 | 332,928 | 83,232 |
| Year 4 | 424,483 | 339,587 | 84,897 |
| Year 5 | 432,973 | 346,378 | 86,595 |
| Year 6 | 441,632 | 353,306 | 88,326 |
| **Total costs** | 2,123,248 | 1,698,599 | 424,650 |
| **Costs:** | | | |
| Software | 30,000 | | |
| Consultants | 50,000 | | |
| Development | 200,000 | | |
| Training | 30,000 | | **Total costs:** 310,000 |

### 1. Ratio

The nominal cost-benefit ratio (disregarding the time value of money) is $424,650 / $310,000 = 1.37. This indicates that the project is worthwhile, in that the extra initial expenses of $310,000 would be exceeded by expected benefits by 37%. The cost-benefit ratio is easily interpreted. If the ratio exceeds 1.0, the project would be profitable if the firm's marginal value of capital were equal to the discount rate. The ratio can be used as a basis for rank-ordering projects, with higher ratios more attractive.

Nas (1996) provides a more complete coverage of cost-benefit analysis.

## B. Payback

Another measure of value is to identify the time for an investment to be repaid. Payback is a rough estimate, but presents a view of the transaction that is very understandable and important to managers.

Another time-related factor is the need for cash flow. One alternative may be superior to another on the net present value of the total life-cycle of the project. However, cost-benefit analysis does not consider the impact of negative cash flow in early periods. For instance, in our process reengineering example, the cash flow would be as given in Table II. The Net Benefit column is calculated by subtracting the cost of the new system from the cost of the old system by year. In the first year, this is negative, due to the high investment cost of the new system. In years 2 through 6, the new system provides a positive net benefit relative to

**Table II** Payback (in Dollars)

| Year | Old system | New system | Net benefit | Cumulative |
|---|---|---|---|---|
| 1 | 400,000 | 400,000–310,000 | −310,000 | −310,000 |
| 2 | 408,000 | 326,400 | +81,600 | −228,400 |
| 3 | 416,160 | 332,928 | +83,232 | −145,168 |
| 4 | 424,483 | 339,587 | +84,897 | −60,271 |
| 5 | 432,973 | 346,378 | +86,595 | +26,323 |
| 6 | 441,632 | 353,306 | +88,326 | +114,650 |

the old system. The new system gains a nominal advantage by the end of year 5 (payback is about 4.7 years), but $310,000 has been sacrificed at the beginning. One of the most common reasons for company failure in the United States is lack of cash flow. In this case, if the firm has cash-flow difficulties, the investment would be less attractive than if they had adequate cash reserves.

## C. The Time Value of Money

We can modify the cost-benefit ratio by considering the time value of money. In this project, for instance, the nominal expected gains of $424,650 are spread out over 5 years, while the extra costs of $310,000 are all incurred at the beginning. Having the $310,000 would mean that the company would not be able to adopt some other investments (and maybe even force the firm to borrow money). The marginal value of money for the firm is 15% per year. Net present value converts a time stream of money back to its worth in today's terms (or in terms of the project's start, or any other specific time of reference).

Table III shows the changes in cash flow between the two systems (shown in the net difference column). Discounting each year's net change in cash flow by the discount rate of 1.15 per year to the *t*th power, where *t* is the time period, we get the following. Note that initial expenses are treated as occurring during year 1. Viewed in this light, relative to obtaining a return of 15% per year on alternative investments, installing the new system would be unattractive, equivalent to writing a check for $23,359 to someone today. If there were some alternative investment on which the company could obtain 15% on their $310,000 investment, they would be ahead by adopting the alternative investment to the tune of over $23,000 over a 5-year period.

A related concept is *internal rate of return,* which is the marginal value of capital for which the net present value of a stream of cash flow would break even, or equal zero. In this case, the internal rate of return (IRR) amounts to 1.113, or 11.3% average return. Therefore the net present value at 15% is negative, and the new system does not appear to be attractive.

It is dangerous to compare projects with different time horizons.

**Table III** Net Present Value (in Dollars)

| Year ($t$) | Old system | New system | Net difference | Divide by $1.15^t$ |
|---|---|---|---|---|
| 1 | 400,000 | 710,000 | −310,000 | −269,565 |
| 2 | 408,000 | 326,400 | +81,600 | +61,701 |
| 3 | 416,160 | 332,928 | +83,232 | +54,726 |
| 4 | 424,483 | 339,587 | +84,897 | +48,540 |
| 5 | 432,973 | 346,378 | +86,595 | +43,053 |
| 6 | 441,632 | 353,306 | +88,326 | +38,186 |
| | | | Net present value | −23,359 |

## D. Other Factors

There are a number of complications that can be brought into the calculation of cost-benefit ratios. One of the most obvious limitations of the method is that benefits, and even costs, can involve high levels of uncertainty. The element of chance can be included in cost-benefit calculations by using expected values.

This section begins with discussion of some other limitations of cost-benefit analysis. This is followed by a calculated example. Then specific information system project characteristics relating to cost-benefit analysis limitations are emphasized.

The cost-benefit ratio does not reflect intangible benefits unless they are presented in monetary terms. Cost-benefit analyses have included measurements for intangible items, but they tend to be given lower values because of the uncertainty involved in their estimates. The *contingent valuation method* seeks to measure the decision maker's willingness to pay for intangible factors. This approach can be time-consuming and less than convincing. Respondents commonly understate their true preferences when asked how much they would be willing to pay, or overstate their bid if they are told they will not be responsible for financing. Governments have encountered some problems in applying cost-benefit analysis to public works, to include evaluating the benefit of recreational facilities. When a dam is built, there clearly is benefit obtained from providing many citizens much improved fishing and water sports. (There also is added cost in depriving citizens of the opportunity to view some flooded historical sites.) The approach usually taken has been to place some dollar value on recreation, based on some very insubstantial measures. The evaluation of risk to human life has also been tackled by economists, who have applied things like the net present value of the expected earnings of those whose lives are expected to be lost in some proposed investment project. This of course involves high levels of speculation as well, because the calculation assumes certain ages, assumes that the only value of a human is what they earn, and disregards who pays and who benefits.

If a firm was threatened with a severe monetary penalty for not complying with a governmental regulation with respect to environmental pollution or safe working conditions, a net present value analysis might well lead to the conclusion that it would be rational to pay the penalty and avoid improving operations. For instance, assume that a blast furnace is pouring out black matter at a phenomenal rate that the government finds terribly offensive. Governmental regulations call for a penalty of $10,000,000 if the pollution source is not cleaned up within one year, through applying information technology in conjunction with more modern processing equipment. Hard core cost-benefit analysis would identify the cost of cleaning up the facility, which might involve an expense of $12,000,000 in equipment and installation, and an added cost of operations of $5,000,000 per year over the next eight years, the remaining life of the equipment. At a discount rate of 12% per year, the net present value of benefits and costs would be as shown below. The net column shows discounted values for benefits and costs. Totals are given at the bottom of Table IV. Here the ratio of net present benefits to net present costs is $8,928,571/35,552,483 = 0.25, well below 1.0, indicating that the rational decision maker would pay the fine and keep operating as is. But the government didn't impose the fine limit for the purpose of raising money. They imposed the fine as a means to coerce polluters to clean up operations. The

**Table IV** Net Present Value (in Dollars)

| Year | Benefits | Net | Costs | Net |
|---|---|---|---|---|
| 1 | $10,000,000 | 8,928,571 | 17,000,000 | 15,178,571 |
| 2 | — | | 5,000,000 | 3,985,969 |
| 3 | — | | 5,000,000 | 3,558,901 |
| 4 | — | | 5,000,000 | 3,177,590 |
| 5 | — | | 5,000,000 | 2,837,134 |
| 6 | — | | 5,000,000 | 2,533,156 |
| 7 | — | | 5,000,000 | 2,261,746 |
| 8 | — | | 5,000,000 | 2,019,416 |
| Total | | 8,928,571 | | 35,552,483 |

United States Congress has no trouble adding extra zeroes to penalties. If the firm continued to pollute, it is not too hard to imagine the penalty being raised in the future to some much larger figure. There have been actual cases similar to this scenario, where within three years the penalty was raised to much larger values, providing a much different cost-benefit ratio. Benefits are often difficult to forecast.

Information systems projects typically involve benefits that are difficult to measure in terms of concrete monetary benefits. This vastly complicates sound management, because cost-benefit analysis will not always accurately reflect project benefits. Hinton and Kaye cited cost and benefit intangibility, hidden outcomes involved in information technology investment, and the changing nature of information technology systems as important issues.

*Intangible factors*—Both costs and benefits tend to have intangible features. The tangible costs and benefits tend to be historical, backed by data or solid price quotations from vendors. But many of the benefits are expected in the future, and are very difficult to measure. These include expected increases in market share, improved customer service, and better corporate image. These would all have a significant impact on the corporation's bottom line, but guessing exactly what that impact would be is challenging at best.

*Hidden outcomes*—Other aspects of information technology projects often involve unexpected results. Information technology projects can impact organizational power. New projects can change the power specific groups may have held in the past, which can have negative impact on the teamwork of the organization. Information technology also includes components of the organization's communications network. Often different elements of the organization can adopt projects that impact the organizational communications network without this impact being considered. This can result in duplication of efforts, or development of barriers between groups within the organization. Computers can make work more productive and more attractive, but they also can change work roles to emphasize skills in which specific employees have no training, making them feel less productive.

Failure to identify the impact of projects often is not noticed until project implementation. At that stage, the problems created are more difficult to deal with. It is important to consider the systems aspects of projects, and try to predict how the project will change how people do their jobs. Thorough user involvement can make project impact more obvious, as well as easier to reconcile and convince users of the project's benefits.

*The changing nature of information technology*—There are many excellent applications of computer technology to aid businesses, but a major problem is that technology is highly dynamic. Some information systems projects take years to implement. This can, and often has, resulted in installation of a new system after it is outdated by even newer technology. A common rule of thumb is a maximum of 9 months between management approval and project component construction in information systems projects.

## II. SELECTION PRACTICE

Hinton and Kaye (1996) surveyed 50 members of a professional organization whose members were responsible for appraising key organizational investments. Methods used for information technology projects were appraised, and compared to projects in other areas (operations, marketing, and training). Treatment of a project as a capital investment involves cost-benefit analysis to establish profitability. Treatment as a revenue-related project does not require cost-benefit analysis, as the project is expected to foster key organizational goals, and the benefits are recognized as being difficult to measure accurately. Projects involving investment in operations and information technology were usually treated with some form of cost-benefit analysis. Investments in training and marketing were usually treated as revenue-related projects, and thus cost-benefit analysis was for the most part foregone.

Net present value masks some of the true value of information technology proposals. On the other hand, some projects that have low impact on corporate performance often appear attractive in cost-benefit analyses. This is because cost-benefit analysis emphasizes those features most easily measured. The value of information technology projects is in making organizations more competitive, increasing customer satisfaction, and operating more effectively. These are the sometimes intangible strategic benefits that are often disregarded because they were not measurable.

Cost-benefit analysis should consider costs over the entire life cycle of the project. Life cycle costs are roughly four times development costs for most information systems projects. But these long-range costs are much less predictable, and therefore often not included in cost-benefit analyses.

Information systems projects involve significant investment on the part of firms. Efficient management of these investments is critical to firm success. However, there is a great deal of difficulty in accurately assessing the costs and benefits involved in most information technology projects. Many companies simply disregard important intangible factors since they involve high levels of uncertainty and even speculation. But there are many important intangible factors involved in assessing the worth of information technology project proposals. Value analysis, or multicriteria analysis offers means of considering such factors.

## III. PROJECT EVALUATION TECHNIQUES

A number of methods exist to evaluate project proposals, either from the perspective of selecting the best option available, designing an ideal option, or rank-ordering options. Cabral-Cardoso and Payne (1996) surveyed research and development decision makers in the United Kingdom about their use of formal selection methods in project selection decisions. The study, based on 152 samples, asked each decision maker to identify familiarity based on use for 18 different methods. Table V lists the most commonly cited methods in the survey (methods rank ordered by percentage use). Economic and financial analyses include payback (determining the expected time until investment is recovered), and cost-benefit analysis. Net present value and internal rate of return consider the time value of money, appropriate when projects are lengthy (3 years or more).

**Table V** Use of Formal Selection Methods

| Method | % Who Used (U.K., 152 samples) |
|---|---|
| Economic and financial | |
| Payback analysis | 68 |
| Cost-benefit analysis | 63 |
| NPV/IRR | 40 |
| Multifactor techniques | |
| Checklist | 38 |
| Project profile | 26 |
| Scoring/rating models | 26 |
| Multicriteria decision models | 11 |
| Other | |
| Goal programming | 18 |
| Expert systems | 6 |

Adapted from Cabral-Cardoso, C., and Payne, R. L. (1996). Instrumental supportive use of formal selection methods in R&D project selection. *IEEE Transactions on Engineering Management,* Vol. 43, No. 4, 402–415.

Checklists describe criteria of importance and their minimum acceptable levels of requirement. Screening methods are a variant of checklists, eliminating projects that do not have minimum estimated performance on specific measures. Project profiles describe the performance of each project on criteria so that the decision maker can see each project's strengths and weaknesses. Scoring and rating models are a simple form of multicriteria analysis where measures are obtained on each criterion of importance, and combined in some fashion. Multicriteria decision models are in general more formal than scoring and rating models, but operate on essentially the same principle—identify factors that are important, measure how well each project does on each factor, and combine these into some value score that can be used for ranking.

The last two types of methods are more specialized. Mathematical programming provides optimal solutions for a portfolio of projects, on any specific measure, subject to constraints, such as budget limits. Expert systems are usually sets of rules that implement a checklist approach in a thorough, systematic manner.

We will now review some of the more popular alternative methods to cost-benefit analysis in information systems project selection.

### A. Screening

*Screening* is a process that is very useful in cutting down the dimensions of the decision problem. The way in which screening operates can vary widely in details, but essentially involves identifying those factors that are important, establishing a minimum level of importance, and eliminating those projects that fail on any one of these minimum standards. Obviously, if the standards are set too high, the decision problem disappears as no projects survive the screening. This is appropriate if the minimum standards reflect what management demands in return for their investment.

To demonstrate screening, assume that 100 information systems project proposals are received this month. All of the projects were evidently worthwhile in someone's mind, but management must consider budgets and other resource limitations. Assume that the criteria and minimum performance levels required are as given in Table VI. If any of the 100 proposed projects failed to meet all four of these standards, they would be rejected preemptively. This reduces the number of proposed projects for more detailed analysis. This approach can be implemented

**Table VI** Screening

| | |
|---|---|
| Expected return on investment | At least 30% |
| Qualified project team leadership | Available |
| Company has expertise in this area | Either company is experienced in this work, or company desires to gain this experience |
| Project completion time | Within 48 months |

by checklists, which give clearly defined standards on those areas of importance to management.

Screening is good at quickly weeding out those projects with unacceptable features. The negative side of screening is that trade-offs between very good features and these unacceptable features are disregarded. The willingness of decision makers to accept lower return on investment (ROI) for projects with strategic importance is disregarded. For those projects for which such trade-offs are not important, screening is a very efficient way to reduce the number of proposals to a more manageable number.

In the prior section, we gave a list of risk factors for information system projects. These could be implemented as a *checklist* by management specifying minimum acceptable measures that can be used to screen individual projects. Not all risk elements might apply for a given organization's checklist. An example checklist is given in Table VII. Checklists ensure implementation of policy limits. Checklists are a way to implement screening from the perspective of features management feels are important. The next step in analysis is to more directly compare alternative project proposals.

The intent of a *project profile* is to display how the project proposal compares to standards, as well as how the project compares to other proposals. Profiles have a benefit over screening limits, because poor performance on one factor can be compensated for by strong performance on another factor. For instance, match with company strategic programs can be an important factor. There could be other project proposals that contribute nothing to the firm's strategic program, yet have an outstanding cost improvement for administrative work. This would be reflected in very strong performance on return on investment. Conversely, another project may have a slightly negative return on investment calculation, but may involve entering a new field in which the firm wants to gain experience.

To demonstrate project profiles, assume a firm has a number of information projects proposed. This is generally a large list, because of the many beneficial things information technology can do for organizations. In Table VIII there is a short list of eight proposals, measured on resources used, as well as benefits expected. A profile displays the characteristics of individual projects. Estimated cost is needed to determine if available budget can support a project. The same is true for other scarce resources, such as systems analysts in this case. A tabular form is given here. Graphical displays and ratios are often valuable to give a measure with which relative performance can be measured. For measures such as NPV/cost ratio, cut-off levels can be used to screen out projects. For instance, a positive return on estimated cost in net present terms might be desired. Projects A265, B837, and C592 are below this limit,

**Table VII** Checklist

| Factor | Minimum standard |
|---|---|
| Project manager ability | Qualified manager available |
| Experience with this type of application | Have experience, or application in a strategically key new technology |
| Experience with the programming environment | Personnel with experience can be obtained |
| Experience with the language or system used | Experienced personnel can be obtained |
| Familiarity with modern programming practices | If not, training is available |
| Availability of critical equipment, software, and programming language | Each critical component available |
| Completeness of project team (are all team members assigned?) | Key personnel identified and agree to work, support personnel widely available |

**Table VIII** Project Profiles

| Project identifier | Estimated cost | Systems analysts | Cash flow this period | NPV/cost ratio | Key to strategy |
|---|---|---|---|---|---|
| A265 | 868,000 | 5 | 100,000 | 0.63 | Yes |
| A801 | 721,000 | 5 | −190,000 | 1.22 | No |
| A921 | 528,000 | 4 | 360,000 | 1.86 | No |
| B622 | 962,000 | 6 | −52,000 | 2.55 | No |
| B837 | 752,000 | 5 | −200,000 | 0.48 | Yes |
| C219 | 649,000 | 5 | 170,000 | 1.12 | Yes |
| C512 | 758,000 | 5 | −320,000 | 1.58 | Yes |
| C592 | 887,000 | 6 | −300,000 | 0.62 | No |

and might be screened out. However, the first two of these projects are listed as key to the organization's strategy, and management might be willing to accept lower return for the potential for advancing organizational strategy.

## B. Value Analysis

Keen (1981) proposed value analysis as an alternative to cost-benefit analysis in the evaluation of proposed information system projects. These projects, clearly attractive to business firms, suffer in that their benefits are often heavily intangible. For instance, decision support systems are meant to provide decision makers with more complete information for decision making. But what is the exact dollar value of improved decision making? We all expect the success of firms to be closely tied to effective decision making, but there is no rational, accurate measure of making better decisions.

Value analysis was presented as a way to separate the benefits measured in intangible terms from costs, which are expected to be more accurately measurable. Those tangible benefits as well as costs can be dealt with in net present terms, which would provide a price tag for proposed projects. The value of the benefits would be descriptive, with the intent of showing the decision maker accurate descriptions of what they were getting, along with the net present price. The decision would then be converted to a shopping decision. Many of us buy automobiles, despite the fact that the net present cost of owning an automobile is negative. Automobiles provide many intangible benefits, such as driving a good looking car, the ability to speed through the countryside (safely, of course), and obtaining a quality means of transportation. The dollar value of these intangible benefits is a matter of willingness to pay, which can be identified in monetary terms by observing the purchasing behavior of individuals. This measurement requires some effort, and is different for each individual.

In terms of our pollution abatement project, the intangible values can be identified by criterion, or measure of value to the decision maker, as shown in Table IX. Value analysis would consist of presenting the decision maker with the intangible comparisons in performance, and placing the decision in the context of whether or not the decision maker thought the improvements provided by the new system were worth about $27 million. If, in the decision maker's

**Table IX** Value Analysis

| | Old system | New system |
|---|---|---|
| Working conditions with respect to safety | Risky | Very safe |
| Impact on market share | Vulnerable on quality | High quality |
| Capital equipment | Deteriorating | In good condition |
| Legal risk | High | Minimal |
| Net present cost relative to current | 0 | $26,624,000 |

judgment, these intangible benefits were clearly worth $27 million or more, the new system should be installed. On the other hand, if the decision maker is unwilling to pay $27 million for these improvements, the old system should be retained.

Taking value analysis one more step, to quantify these intangible benefits in terms of value (not in terms of dollars), takes us to multicriteria analysis.

## C. Multiple Objectives

Profit has long been viewed as the determining objective of a business. However, as society becomes more complex, and as the competitive environment develops, businesses are finding that they need to consider multiple objectives. While short-run profit remains important, long-run factors such as market maintenance, product quality, and development of productive capacity often conflict with measurable short-run profit.

### 1. Conflicts

Conflicts are inherent in most interesting decisions. In business, *profit* is a valuable concentration point for many decision makers because it has the apparent advantage of providing a measure of worth. Minimizing *risk* becomes a second dimension for decision making. There are cash flow needs which become important in some circumstances. Businesses need *developed markets* to survive. The impact of advertising expenditure is often very difficult to forecast. Yet decision makers must consider advertising impact. *Capital replenishment* is another decision factor which requires consideration of trade-offs. The greatest short-run profit will normally be obtained by delaying reinvestment in capital equipment. Many United States companies have been known to cut back capital investment in order to appear reasonably profitable to investors. *Labor* policies can also have impact upon long-range profit. In the short run, profit will generally be improved by holding the line on wage rates, and risking a high labor turnover. There are costs which are not obvious, however, in such a policy. First, there is training expense involved with a high turnover environment. The experience of the members of an organization can be one of its most valuable assets. Secondly, it is difficult for employees to maintain a positive attitude when their experience is that short-run profit is always placed ahead of employee welfare. And innovative ideas are probably best found from those people who are involved with the grass roots of an organization—the work force.

This variety of objectives presents decision makers with the need to balance conflicting objectives. We will present the simple multi-attribute rating technique (SMART), an easy to use method to aid selection decisions with multiple objectives.

### 2. Multicriteria Analysis

Multicriteria analysis considers benefits on a variety of scales without directly converting them to some common scale such as dollars. The method (there are many variants of multicriteria analysis) is not at all perfect, but it does provide a way to demonstrate to decision makers the relative positive and negative features of alternatives, and gives a way to quantify the preferences of decision makers.

Perhaps the easiest application of multicriteria analysis is the simple multi-attribute rating theory (SMART), which identifies the relative importance of criteria in terms of weights, and measures the relative performance of each alternative on each criterion in terms of scores. We will first explain scores.

#### a. Scores

Scores in SMART can be used to convert performances (subjective or objective) to a zero-one scale, where zero represents the worst acceptable performance level in the mind of the decision maker, and one represents the ideal, or possibly the best performance desired. Note that these ratings are subjective, a function of individual preference. Scores for the criteria given in the prior section for the pollution abatement decision could be as given in Table X. The safety score was assigned assuming that the new system would provide working conditions as ideal as possible. The score for the old system is an evaluation expressing the decision maker's judgment of just how unsafe the old system was. A rating of 0.3 indicates that it is pretty

**Table X** SMART Scores of Alternatives on Criteria

| | Old system | New system |
|---|---|---|
| Working conditions with respect to safety | 0.3 | 1.0 |
| Impact on market share | 0.5 | 1.0 |
| Capital equipment | 0.4 | 1.0 |
| Legal risk | 0.0 | 1.0 |
| Net present cost | 1.0 | 0.3 |

**Table XI** Criteria Anchors and Ranking

| | Worst measure | Best measure |
|---|---|---|
| Impact on market share | Poor quality product | High quality |
| Legal risk | High | Minimal |
| Net present cost | Worst expected | 0 |
| Working conditions with respect to safety | Very risky | Very safe |
| Capital equipment | Need replacement now | In mint condition |

bad. With respect to market share, the new system would provide as good an impact on market share as any system the decision maker could imagine. The old system would involve significant risk of losing market share, rated at 0.5. Obtaining the new system would place the company in an excellent position with respect to capital equipment, while the old system would be much worse (rated at 0.4). The legal risk of the current system is as bad as the decision maker could imagine, while the proposed system would be as good as could be imagined. Finally, the cost of the old system is much better, and in fact lower than any imaginable alternative, while the new system's net present cost of $27 million is rated as bad (numerically, 0.3 on a 0–1 scale).

### b. Weights

The next phase of the analysis ties these ratings together into an overall value function by obtaining the relative weight of each criterion. In order to give the decision maker a reference about what exactly is being compared, the relative range between best and worst on each scale for each criterion should be explained.

There are many methods to determine these weights. In SMART, the process begins with rank-ordering the four criteria, after consideration of anchors of worst and best measures. A possible ranking for a specific decision maker might be as in Table XI. Two estimates of weights can be obtained. The first assigns the least important criterion 10 points, and assesses the relative importance of each of the other criteria on that basis. Table XII demonstrates this estimation of relative weights. These add to 125. The first estimate of weights would divide each relative importance by 125, yielding the values in Table XIII. The total will of course add to 1.0. The implication is that impact on market share (over the range of values considered) is four times as important as maintaining capital equipment, and about 1.5 times as important as net present cost. The second estimate of weights is obtained by looking at relative importance from the opposite perspective. The most important criterion is assigned 100 points, and the others evaluated on that basis. This is supposed to be an independent check of the prior estimate, although relative order should be maintained in both assessments. Table XIV gives an example of these assessments. This total is 280. The second estimate of weights is therefore obtained by dividing each value by 280 as in Table XV. The next step is to compromise if necessary between these two estimates. The last criterion can be used to make sure that the sum of compromise weights adds to 1.00.

### c. Value Score

The next step of the SMART method is to obtain value scores for each alternative by multiplying each score on each criterion for an alternative by that cri-

**Table XII** Assessment of Weight Preferences

| | Worst measure | Best measure | Assigned value |
|---|---|---|---|
| Capital equipment | Need replacement now | In mint condition | 10 |
| Working conditions with respect to safety | Very risky | Very safe | 15 |
| Net present cost | Worst expected | 0 | 25 |
| Legal risk | High | Minimal | 35 |
| Impact on market share | Poor quality product | High quality | 40 |

**Table XIII** Estimation of Relative Criteria Weights

| | | |
|---|---|---|
| Capital equipment | 10/125 | = 0.08 |
| Working conditions with respect to safety | 15/125 | = 0.12 |
| Net present cost | 25/125 | = 0.20 |
| Legal risk | 35/125 | = 0.28 |
| Impact on market share | 40/125 | = 0.32 |

terion's weight, and adding these products by alternative, as in Table XVI. This value score provides a relative score that can be used to select (take the alternative with the highest value score), or to rank order (by value score). In this case, the new system would be indicated as better fitting the preferences of the decision maker.

### d. Other Multiple Criteria Methods

Note that there are many other approaches implementing roughly the same idea. The best known is multi-attribute utility theory, which uses more sophisticated (but not necessarily more accurate) methods to obtain both scores and weights. The analytic hierarchy process is another well-known approach.

## IV. SUMMARY

Cost-benefit analysis (with net present value used if the time dimension is present) is the ideal approach from the theoretical perspective, but has a number of limitations. It is very difficult to measure benefits, and also difficult to measure some aspects of costs accurately. One view of dealing with this problem is to measure more accurately. Economists have developed ways to estimate

**Table XIV** Check Assignment of Relative Criteria Weights

| | Worst measure | Best measure | Assigned value |
|---|---|---|---|
| Impact on market share | Poor quality product | High quality | 100 |
| Legal risk | High | Minimal | 80 |
| Net present cost | Worst expected | 0 | 50 |
| Working conditions with respect to safety | Very risky | Very safe | 30 |
| Capital equipment | Need replacement now | In mint condition | 20 |

**Table XV** Estimation of Check Weights

| | Based on best | | Based on worst | Compromise |
|---|---|---|---|---|
| Impact on market share | 100/280 | = 0.357 | 0.320 | 0.34 |
| Legal risk | 80/280 | = 0.287 | 0.280 | 0.28 |
| Net present cost | 50/200 | = 0.179 | 0.200 | 0.19 |
| Working conditions with respect to safety | 30/200 | = 0.107 | 0.120 | 0.11 |
| Capital equipment | 20/200 | = 0.071 | 0.080 | 0.08 |

**Table XVI** Calculation of SMART Value Scores

| Criterion | Weight | Old system | New system |
|---|---|---|---|
| Working conditions with respect to safety | 0.11 | × 0.3 = 0.033 | × 1.0 = 0.110 |
| Impact on market share | 0.34 | × 0.5 = 0.170 | × 1.0 = 0.340 |
| Capital equipment | 0.08 | × 0.4 = 0.032 | × 1.0 = 0.080 |
| Legal risk | 0.28 | × 0.0 = 0.000 | × 1.0 = 0.028 |
| Net present cost | 0.19 | × 1.0 = 0.190 | × 0.3 = 0.057 |
| Value score by alternative (sum) | | 0.425 | 0.867 |

the value of a life, and the value of scenic beauty. However, these measures are difficult to sell to everybody.

Screening provides a way to simplify the decision problem by focusing on those projects that are acceptable on all measures. Profiles provide information that display trade-offs on different measures of importance.

Value analysis isolates intangible benefits from those benefits and costs that are more accurately measurable in monetary terms, and relies upon decision-maker judgment to come to a more informed decision. The SMART method, one of a family of multicriteria decision analysis techniques, provides a way to quantify these intangible factors to allow decision makers trade-off values.

Cost-benefit provides an ideal way to proceed if there are no intangible factors (or at least no important intangible factors). However, usually such factors are present. Intermediate approaches, such as payback analysis and value analysis, exist to deal with some cases. More complex cases are better supported by multicriteria analysis. In cases of constraints, such as budgets, it is sometimes appropriate to optimize over some objective. Linear programming provides a means of generating the best portfolio of funded projects subject to constraint limits given that accurate measures of project performance are available.

## ACKNOWLEDGMENT

This chapter draws heavily from Olson, D. L. (2001). *Introduction to information systems project management.* Boston: Irwin/McGraw-Hill. With permission.

## SEE ALSO THE FOLLOWING ARTICLES

Accounting • Control and Auditing • Operations Management • Procurement • Systems Analysis • Value Chain Analysis

## BIBLIOGRAPHY

Cabral-Cardoso, C., and Payne, R. L. (1996). Instrumental and supportive use of formal selection methods in R&D project selection. *IEEE Transactions on Engineering Management,* Vol. 43, No. 4, 402–410.

Edwards, W. (1977). How to use multiattribute utility measurement for social decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-7, No. 5, 326–340.

Hinton, M., and Kaye, R. (November 1996). Investing in information technology: A lottery? *Management Accounting* Vol. 74, No. 10, 52.

Keen, P. G. W. (1981). Value analysis: Justifying decision support systems. *MIS Quarterly,* Vol. 5, No. 1, 1–16.

Keeney, R. L., and Raiffa, H. (1976). *Decisions with multiple objectives: Preferences and value tradeoffs.* New York: John Wiley & Sons.

Nas, T. F. (1996). *Cost-benefit analysis: Theory and application.* Thousand Oaks, CA: Sage Publications.

Olson, D. L. (1996). *Decision aids for selection problems.* New York: Springer-Verlag.

Olson, D. L. (2001). *Introduction to information systems project management.* Boston: Irwin/McGraw-Hill.

Saaty, T. L. (1977). A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology,* Vol. 15, 234–281.

# Crime, Use of Computers in

**M. E. Kabay**
*Norwich University*

## GLOSSARY

**authenticity** Validity, conformance, and genuineness of information.

**availability** Timely accessibility of information for a specific purpose.

**confidentiality** Limited observation and disclosure of knowledge.

**data diddling** Unauthorized modification of data.

**denial of service** Prevention of availability due to resource saturation or resource destruction.

**eavesdropping** Unauthorized interception of communications.

**integrity** Completeness, wholeness, and readability of information; quality of being unchanged from a prior state.

**logic bomb** Unauthorized, harmful executable code whose actions are triggered by a logical condition such as the presence or absence of specific data or by a particular time or date.

**malware** Contraction of “malicious software”; executable code intended by its writer to violate information security of its victims. Examples include viruses, worms, logic bombs, Trojan Horses, and denial-of-service programs.

**penetration** Unauthorized access to resources through violation of access-control restrictions.

**possession** Holding, control, and ability to use information.

**social engineering** The use of deceit to persuade other human beings to help an attacker violate information security restrictions.

**Trojan Horse** Software having undocumented and unauthorized functions in addition to or instead of expected useful functions.

**utility** Usefulness of information for a purpose.

**virus** Self-replicating executable code that inserts unauthorized instructions into other executable codes.

**worm** Self-replicating executable code that passes copies of itself through computer communications networks.

This article reviews the important types of **CRIMES** involving computers and networks. Computers and computer networks are tools for obtaining, storing, manipulating, and transmitting information. Like any other tool, they can be used for social good or for social ill. Criminals have used every technological innovation in history as the basis for new or variant crimes, and the criminal subculture has been active in turning computers and networks towards its ends. Computers and networks play a role in crime both as mediating instruments of crime and, in contrast, as the objects or targets of crime.

## I. THE FOUNDATIONS OF INFORMATION SECURITY

The classic definition of information security was developed in the 1970s: Data security [involves] the protection of information from unauthorized or accidental modification, destruction, and disclosure. The "classic triad" of information security names confidentiality, integrity, and availability. To these three, the respected security expert Donn B. Parker has added possession, authenticity, and utility.

## A. Basic Concepts

*Protection* means reducing the likelihood and severity of damage. Another way of putting this is that information security strives to reduce risks. It is not possible in practice to provide perfect prevention of security violations. Common sense suggests that the degree of protection must match the value of the data.

*Information* is protected by caring for its form, content, and storage medium.

*Unauthorized* means forbidden or undocumented. The very concept of authorization implies classification: there must be some definition of which data are to be protected and at what level.

*Modification* means changes of any kind. The ultimate modification is destruction. However, small but significant changes in data cause more trouble than destruction. For example, the damage caused by a vandal who damages a Web site by adding pornography and vile language can be spotted at once and can be removed quickly. In contrast, some kinds of malicious software can make small random changes (e.g., in spreadsheets) that can accumulate for months. Backup copies of the corrupted files may make it impossible to recover valid versions of these files.

*Disclosure* means allowing people to see or use data. The critical element is authorization: permission by a *data owner* for selected others to have access to these data.

*Confidentiality* is a wider concept than disclosure. For example, certain files may be confidential; the data owner may impose operating system controls to restrict access to the data in the files. Nevertheless, it may be possible for an unauthorized person to see the names of these files or find out how often they are accessed. Changing a file's security status may be a breach of confidentiality; for example, copying data from a secure file to an unsecured file is a breach of confidentiality.

*Possession* means control over information. For example, when thieves copy proprietary software without authorization, they are breaching the owner's possession of the software. Such *counterfeit* software represents a breach of possession or control. Similarly, if someone obtains an unauthorized copy of a confidential document, there is a breach of possession or control even before anyone actually looks at the document because the owner no longer determines when the data will be disclosed to unauthorized people.

*Integrity* refers to internal consistency. A database is termed structurally corrupt when its internal pointers or indexes no longer correspond to the actual records they point to. For example, if the next record in a group is in position 123 but the index pointer refers to position 234, the structure lacks integrity. Surreptitiously using a disk editor to bypass security and alter pointers in such a data structure would impair integrity even if all the data records were left intact. Logical corruption occurs when data are inconsistent with each other or with system constraints. For example, if the summary field in an order header contains a total of $5678 for all items purchased but the actual sum of the costs is $6789 then the data structure is logically corrupt; it lacks integrity.

*Authenticity* refers to correspondence between data and what the data represent: accordance with reality, correctness. A typical example of impaired authenticity is electronic mail sent using a false name—or worse, someone else's name.

*Availability* means that data can be used in a timely fashion; the data are convenient or handy. If a server crashes, the data on its disks are no longer available; but if a mirror disk is at hand, the data may still be available.

*Utility* refers to the usefulness of data for specific purposes. Even if the information is still intact, it may have been transformed into a less useful form. For example, unauthorized encryption of a firm's source code for production program is a breach of utility. In a formal sense, the data were authentic, accurate, and available—they just were not useful.

## B. Threats to Security

Enterprise systems are faced with two kinds of threat: people and disasters. People include managers, employees, service personnel, temporary workers, suppliers, clients, thieves, liars, and frauds. Disasters include fire, flood, earthquake, civil disturbance, and war.

The difficulty in describing the risk of facing these threats is that we lack proper statistical information about how often different types of damage occur. In statistical work, this difficulty is known as the problem of ascertainment. Most organizations are reluctant to admit, let alone publicize, successful attacks on their information systems.

The second part of the ascertainment problem is that even if people were reporting all the computer crimes and accidents they knew about, we would still not know about the crimes and accidents that have not yet been discovered.

Keeping in mind that all statistics about computer crimes are problematic, the information security field has arrived at a shaky consensus about the origins of damage to computer systems and networks. In brief,

- Perhaps as much as half of the damage is due to errors and omissions by authorized users of the systems.
- Fire and water damage and problems resulting from poor electrical power account for perhaps a quarter of the problems.
- Authorized but dishonest or disgruntled employees are a significant source of difficulties.
- Malicious software and outside attacks were thought to account for a small portion of the threat to systems before the explosive growth in Internet usage in the early 1990s; however, by the turn of the millennium, both malicious software and outsiders posed a much greater source of danger, perhaps approximating the threat from angry and dishonest insiders.

Figure 1 shows the rough guesses about damage to computer systems before and after the explosion of Internet usage that occurred around 1993. Note that the edges of the categories are deliberately made fuzzy to remind the reader of the uncertainty of these estimates. The categories are

- E&O. Errors and omissions; due to lack of training, poor motivation, or poor supervision
- Fire, water. Arson, accident, sabotage; water damage often accompanies fire damage
- Dishonest. Employees
- Disgruntled. Employees
- Outsider. Contractors, visitors, strangers
- Virus. Self-replicating code that integrates into executable code
- Worms. Self-replicating code that propagates through networks
- Trojans. Software with undocumented and unauthorized functions
- DoS. Denial of service attacks

## II. THE LEGAL FOUNDATIONS

This section reviews some of the key laws that govern the use of computers and networks and which criminalize specific acts.

### A. United States Computer-Crime Laws

The most advanced set of laws criminalizing particular unlawful behavior involving computers and networks has been legislated in the United States. The *Computer Fraud and Abuse Act of 1986* (18 USC §1030) focuses primarily on protecting "government-interest" computers, including federal, state, county, and municipal systems; financial and medical institutions; and computers used by contractors supplying such institutions. Specifically, the Act prohibits the use of "a program, information, code, or command" with intent to damage, cause damage to, or deny access to a

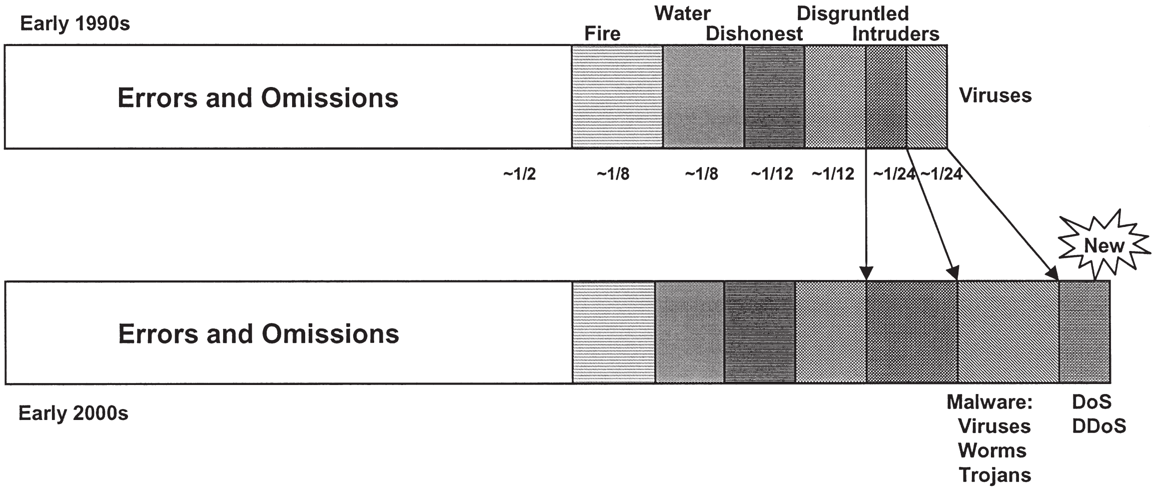


**Figure 1** Rough guesses about the sources of damage to information systems. Copyright 2002, M. E. Kabay.

computer system or network. In addition, the Act specifically prohibits even unintentional damage if the perpetrator demonstrates reckless disregard of the risks of causing such damage.

Another law governing interstate electronic communications has been used in prosecutions of computer crimes: 18 USC §1343, dealing with wire fraud. Wire fraud requires the following elements: (a) a scheme to defraud by means of false pretenses; (b) knowing and willful participation with intent to defraud; (c) the use of interstate wire communications in furtherance of the scheme.

The Electronic Communications Privacy Act of 1986 (18 USC §1367 and others), generally known as the ECPA, assigns fines and prison sentences for anyone convicted of unauthorized interception and disclosure of electronic communications such as phone calls through land lines or mobile systems and e-mail. In addition, the ECPA specifically prohibits making use of an unlawfully overheard electronic communication if the interceptor knows that the message was unlawfully obtained. On the other hand, providers of electronic messaging systems, including employers, are permitted to intercept messages on their own systems in the course of their normal operations; naturally, they are authorized to transmit messages to other communications providers as part of the normal course of transmission to the ultimate recipient. The ECPA also prohibits access to stored messages, not just those in transit.

United States law also criminalizes the use of interstate communications for the transmission of threats, in kidnappings, and in extortion (18 USC §2518). Another form of prohibited speech is everything associated with child pornography: making, sending, publishing, or storing images of children engaged in sexually explicit conduct (18 USC §2251).

The Communications Decency Act of 1996 (47 USC §223) was a highly controversial statute prohibiting anyone using interstate or communications from transmitting obscene or indecent materials when they know that the recipient is under 18 years of age—regardless of who initiated the communications. In June 1997, in a stinging rebuke to proponents of censorship, the United States Supreme Court issued its ruling on the Communications Decency Act, finding that it violated First Amendment protection of free speech. The unanimous opinion stated that the effort to protect children from sexually explicit material went too far because it also would keep such material from adults who have a right to see it.

In addition to federal laws, the United States has a tapestry of state laws applying to computer crimes. States differ widely in the availability of computer-crime laws and in their definitions and penalties.

## B. Criminal Law and Civil Law

Another area of legal constraints originates in civil law. Issues of copyright, trademark, defamation, privacy, anonymity and pseudonymity, duty of care, and digital signatures are too complex for this article, which focuses on the relatively simple concepts of unauthorized access to or interference with computer systems and networks (see Bibliography).

## C. International Developments

Few countries have kept up with the United States in their legislation concerning computer crimes. However, there have been recent developments bringing hope to the targets and victims of computer criminals. The following sections give a few examples of legislation to illustrate the kinds of issues and penalties being developed around the world in cyberlaw.

### 1. Canada

Canadian law (section 342.1) specifies, “Every one who, fraudulently and without color of right, (a) obtains, directly or indirectly, any computer service, (b) by means of an electromagnetic, acoustic, mechanical or any other device, intercepts or causes to be intercepted, directly or indirectly, any function of a computer system, or (c) uses or causes to be used, directly or indirectly, a computer system with intent to commit an offence under paragraph (a) or (b) or an offence under section 430 in relation to data or a computer system is guilty of an indictable offence and liable to imprisonment for a term not exceeding ten years, or is guilty of an offence punishable on summary conviction.”

In addition, Canadian law addresses “mischief” pertaining to computer systems (section 430.1): “Every one commits mischief who wilfully (a) destroys or alters data; (b) renders data meaningless, useless or ineffective; (c) obstructs, interrupts or interferes with the lawful use of data; or (d) obstructs, interrupts or interferes with any person in the lawful use of data or denies access to data to any person who is entitled to access thereto.”

On January 1, 2001, Canada’s Personal Information Protection and Electronic Documents Act took effect. The law defined statutory obligations for protecting privacy, among other security-related topics.

### 2. United Kingdom

In Britain, the Computer Misuse Bill of 1990 defines unauthorized access to computer material (including equipment and data), stipulates that there be intent to commit or facilitate commission of further offenses, and specifically addresses the issue of unauthorized modification of data. The law states that there is no need to prove that the defendant was aiming to harm any particular program or data, any particular kind of program or data, or indeed programs or data held in any particular computer. Penalties are limited to a maximum of 5 years in prison and various levels of fines.

### 3. Germany

German law (section 202a) defines "data spying" as unauthorized access to other people's data and comes down hard on "violation of private secrets" (section 203) which include in particular data held by physicians, dentists, veterinarians, pharmacists, psychologists, lawyers, patent agents, notaries public, defense counsel, certified public accountants, sworn auditors, tax advisors, auditors, marriage/family/educational/youth/addiction counselors, social workers, insurance companies, and several other categories of data owners. Violation of this provision can be punished by fines or imprisonment of up to 1 year.

Section 204 specifically identifies industrial espionage by augmenting the possible penalties to a maximum of 2 years in prison.

Section 263a increases the penalties yet again for anyone convicted of computer fraud: "Anybody who, with a view to procuring himself of a third person any unlawful property advantage, causes prejudice to the property of another by influencing the result of a data proceeding activity through improper program design, through the use of incorrect or incomplete data, through the unauthorized use of data, or otherwise through any unauthorized interference with the transaction, shall be sentenced to imprisonment not exceeding five years or to a fine."

Other sections of German law explicitly deal with forgery, deception by unauthorized modification of data, and computer sabotage.

### 4. Italy

Law number 547 dating to 1993 established Article 615.5 of the Penal Code: "Spreading of programs aimed at damaging or interrupting a computer system. Anyone who spreads, transmits, or delivers a computer program, whether written by himself or by someone else, aimed at or having the effect of damaging a computer or telecommunication system, the programs or data contained in or pertaining to it, or interrupting in full or in part or disrupting its operation is punished with the imprisonment for a term of up to two years and a fine of up to It. L. 20,000,000."

### 5. Switzerland

Article 144bis, in force since 1995, stipulates, "Anyone, who without authorization deletes, modifies, or renders useless electronically or similarly saved or transmitted data, will, if a complaint is filed, be punished with the imprisonment for a term of up to 3 years or a fine of up to 40,000 Swiss francs. If the person charged has caused a considerable damage, the imprisonment will be for a term of up to 5 years."

As for malicious software, "Anyone, who creates, imports, distributes, promotes, offers, or circulates in any way programs, that he/she knows or has to presume to be used for purposes according to the item above, or gives instructions to create such programs, will be punished with the imprisonment for a term of up to 3 years or a fine of up to 40,000 Swiss francs. If the person charged acted for gain, the imprisonment will be for a term of up to 5 years."

### 6. Other Countries

For a comprehensive and frequently updated review of computer crime laws in 37 countries, see Stein Schjolberg's review, "The Legal Framework: Unauthorized Access to Computer Systems—Penal Legislation in 37 Countries." The Web address is http://www.mossbyrett.of.no/info/legal.html. The 37 countries covered are:

Argentina
Australia
Austria
Belgium
Brazil
Canada
Chile
China
Czech Republic
Denmark
Egypt
Finland
France
Germany
Greece
Hungary
Iceland
Ireland
Israel
Italy
Japan
Luxembourg
The Netherlands
New Zealand
Norway
Poland
Portugal
Romania
Singapore
South Africa
Spain
Sweden
Switzerland
Tunisia
Turkey
United Kingdom
United States

## D. Jurisdictional Problems

Cyberspace crime poses a jurisdictional problem because the perpetrator of a crime can reside in one country, act through computers and networks in several other countries, and cause harm to computer systems in yet other countries. Trying to investigate and prosecute crimes that are carried out in milliseconds when international cooperation can take days and weeks means that many computer crimes simply go unpunished.

The most irritating aspect of computer crime investigations and prosecutions is the jurisdictional quagmire resulting from incomplete and inconsistent laws. In international law, no one may legally be extradited from one country to face prosecution in another country unless both counties involved have dual criminality. That is, an offense must be similar in law and at the same level of criminality (misdemeanor, felony) before extradition can be considered by courts of law.

A good example of the frustration felt by law enforcement officials and victims of computer crime occurred in the year 2000, when a worldwide infestation by the e-mail-enabled worm *Love Bug* caused damage and lost productivity estimated in the hundreds of millions of dollars. The putative originator of the worm was a computer programming student in Manila, The Philippines. Even though the alleged perpetrator came close to admitting his responsibility for the infection—and was lionized by the local press—there were no applicable laws in The Philippines under which he could be prosecuted locally. As a result, he was never extradited to the United States for prosecution.

## III. CLASSIFICATION OF BREACHES OF INFORMATION SECURITY

The study of computer crime has not reached the state of academic rigor characteristic of a mature field. Classification of computer crimes remains relatively primitive. However, there are two ways of referring to computer crimes that are sometimes used to organize discussions. Many authors provide lists of computer crimes, but there is rarely any obvious underlying principle for the sequence of crimes in their lists.

### A. Levels of Information Warfare

One approach to organizing the types of computer crime is based on the work of Winn Schwartau, a controversial author and speaker who was active during the 1990s in warning of the danger of an "electronic Pearl Harbor" and succeeded in bringing electronic attack methods and countermeasures to public attention. Schwartau points out in his book *Information Warfare* that there are three obvious levels of target in electronically mediated conflict: individuals, corporations and other organizations, and countries. He refers to these classes as Interpersonal, Intercorporate, and International Information Warfare. This schema permits a crude but useful level of organization for discussions of crime and warfare directed at and mediated through information technology.

### B. John D. Howard's Analysis

In his 1997 doctoral dissertation, John D. Howard presents a thorough analysis of computer incidents.

Howard starts by defining the following elements of a computer security event:

- Attacker
- Tool
- Vulnerability
- Action
- Target
- Unauthorized result
- Objective

Security events may involve more than one factor from each of the elements; in that sense, the analysis is not a taxonomy because it cannot be used to assign any given crime to a single class. Nonetheless, Howard's work is most helpful in thinking about computer crime.

The attackers include

- Hackers
- Spies
- Terrorists
- Corporate raiders
- Professional criminals
- Vandals
- Voyeurs

The tools available to computer criminals include

- Physical attack
- Information exchange
- User command
- Script or program
- Autonomous agent
- Toolkit
- Distributed tool
- Data tap

The vulnerabilities that can be exploited by an attacker include

- Design
- Implementation
- Configuration

Attackers can use their tools on specific vulnerabilities by taking the following *actions:*

- Probe
- Scan
- Flood
- Authenticate
- Bypass
- Spoof
- Read
- Copy
- Steal
- Modify
- Delete

The specific targets addressed by these actions include

- Account
- Process
- Data
- Component
- Computer
- Network
- Internetwork

The unauthorized results include

- Increased access
- Disclosure of information
- Corruption of information
- Denial of service
- Theft of resources

The objectives of all this effort include

- Challenge, status, thrill
- Political gain
- Financial gain
- Damage

## IV. CRIMES WHERE COMPUTERS AND NETWORKS ARE TOOLS ONLY

This article makes a distinction between computer crimes that use computers and networks as tools versus those where the computers and networks are the primary targets of the crime as well as being tools. We start with computers and networks as tools.

### A. Fraud

One of the most common forms of computer crime is data diddling—illegal or unauthorized data alteration. These changes can occur before and during data input or before output. Data diddling cases have included banks, payrolls, inventory, credit records, school transcripts, and virtually any other form of data storage known. In most of these cases, the purpose was to defraud victims by using the modified data to misrepresent reality and thereby to trick the victims into granting or allowing gain to the perpetrators.

#### 1. The Equity Funding Fraud

Perhaps the most notorious case of computer-mediated fraud through data diddling was the *Equity Funding Fraud,* a case of organized data diddling on a scale unparalleled to date which took place from 1969 through 1972.

The case began with computer problems at the Equity Funding Corporation of America, a publicly traded and highly successful firm with a bright idea. The idea was that investors would buy insurance policies from the company and also invest in mutual funds at the same time, with profits to be redistributed to clients and to stockholders. Through the late 1960s, Equity's shares rose dizzyingly in price; there were news magazine stories about this wunderkind of the Los Angeles business community.

The computer problems occurred just before the close of the financial year. An annual report was about to be printed, yet the final figures simply could not be extracted from the mainframe. In despair, the head of data processing told the president the bad news; the report would have to be delayed. The president ordered him to make up the bottom line to show about $10,000,000 in profits and calculate the other figures so it would come out that way. The DP chief obliged, rationalizing it with the thought that it was just a temporary expedient and could be put to rights later in the real financial books.

The expected profit didn't materialize, and some months later, the head of DP was in trouble again. The books were not going to balance: Where were the large inflows of cash from investors that the company had counted on? The executives at Equity manufactured false insurance policies which would make the company look good to investors. They inserted false information about nonexistent policies and identified the fraudulent records with special customer codes to exclude then from audit listings, thus tricking a lackadaisical auditor who saw only records which had corresponding paper files for real policyholders.

In time, Equity's corporate staff decided to sell the policies to other insurance companies via the redistribution system known as re-insurance, which spreads the risk of insurance policies across cooperating groups of insurers. The imaginary policies brought in large amounts of real cash. When it came time to start paying real money to the re-insurers for the policies in the names of fake people, the criminals "killed" the imaginary holders of the fake policies. Equity naturally demanded real money for the imaginary beneficiaries of the ghostly policy holders. Re-insurers poured cash into Equity—over a million dollars for these false deaths.

By the spring of 1971, the executives were churning out between 20,000 and 50,000 fake policies per year; by 1972, 64,000 of the companies 97,000 policies were fraudulent. The face value of these invented people's insurance policies totaled $2.1 billion out of a total of $3.2 billion. About 25% ($185M) of the company's total assets ($737M) reported in 1971 were imaginary.

As has often happened in cases of conspiracy, an angry computer operator who had to work too much overtime reported the fraud to the Securities and Exchange Commission. Although the crooked managers tried to erase incriminating computer tapes, they were arrested, tried, and condemned to prison terms.

## 2. Vladimir Levin

In February 1998, Vladimir Levin was convicted to three years in prison by a court in New York City. Levin masterminded a major conspiracy in 1994 in which the gang illegally transferred $12M in assets from Citibank to a number of international bank accounts. The crime was spotted after the first $400,000 was stolen in July 1994 and Citibank cooperated with the FBI and Interpol to track down the criminals. Levin was also ordered to pay back $240,000, the amount he actually managed to withdraw before he was arrested. This case illustrates the international, boundary-crossing nature of today's computer-mediated crime.

## 3. Salamis

A particular kind of computer fraud is called the *salami* technique. In the salami technique, criminals steal money or resources a bit at a time. Two different etymologies are circulating about the origins of this term. One school of security specialists claims that it refers to slicing the data thin—like a salami. Others argue that it means building up a significant object or amount from tiny scraps—like a salami.

The classic story about a salami attack is the "collect-the-roundoff" trick. In this scam, a programmer modifies the arithmetic routines such as interest computations. Typically, the calculations are carried out to several decimal places beyond the customary two or three kept for financial records. For example, when currency is in dollars, the roundoff goes up to the nearest penny about half the time and down the rest of the time. If the programmer arranges to collect these discarded fractions of pennies in a separate account, a sizable fund can grow with no warning to the financial institution.

More daring salamis slice off larger amounts. The security literature includes case studies in which an embezzler removed $0.20 to $0.30 from hundreds of accounts two or three times a year. These thefts were not discovered or reported until an audit found them: most victims wouldn't bother finding the reasons for such small discrepancies.

In another scam, two programmers made their payroll program increase the federal tax-withholding amounts by a few cents per pay period for hundreds of fellow employees. The excess payments were credited to the programmers' withholding accounts instead of to the victims' accounts. At income-tax time the following year, the thieves received fat refunds from the Internal Revenue Service.

In January 1993, four executives of a Value Rent-a-Car franchise in Florida were charged with defrauding at least 47,000 customers using a salami technique. The defendants modified a computer billing program to add five extra gallons to the actual gas tank capacity of their vehicles. From 1988 through 1991, every customer who returned a car without topping it off ended up paying inflated rates for an inflated total of gasoline. The thefts ranged from $2 to $15 per customer—rather thick slices of salami but nonetheless difficult for most victims to detect.

In 1998, Los Angeles, district attorneys charged four men with fraud for allegedly installing computer chips in gasoline pumps that cheated consumers by overstating the amounts pumped. The problem came to light when an increasing number of consumers charged that they had been sold more gasoline than the capacity of their gas tanks. However, the fraud was difficult to prove initially because the perpetrators programmed the chips to deliver exactly the right amount of gasoline when asked for 5- and 10-gallon amounts, which were the standard volumes used by inspectors.

### 4. Stock Fraud

Fraud artists have used letters and newspapers to trick victims into giving away money for nothing; naturally, today's confidence tricksters use e-mail and the World Wide Web for similar purposes.

One of the more popular scams is the *pump-and-dump* stock fraud. The perpetrators use e-mail or the Web to stimulate or manipulate specific stocks; depending on when and how they buy the stocks, the crooks can make a profit either by raising the stock price or by lowering it. For example, a former employee of online press release distributor Internet Wire was arrested in August 2000 and charged with securities and wire fraud in connection with the distribution of a phony press release that sent a tech company's stock price plummeting the week before. Shares of Emulex, a maker of fiber-optic equipment, lost up to 60% of their value, most of it during one 15-minute freefall, after some financial news services, including Dow Jones and Bloomberg, ran stories based on the release. The suspect netted profits of $240,000.

## B. Counterfeits of Documents and Money

Creating false documents long predates the use of computers; however, digital scanners, digital-image editing programs, and high-resolution color printers have made forgeries easy. People have created convincing counterfeit money, sent authentic-looking faxes leading to the premature release of prisoners, and used impressive but false letters of recommendation—complete with digitized logos of prestigious institutions copied from Web sites—to get jobs for which they were unqualified.

One of the more ingenious forgeries occurred in the 1970s, when automatic processing of checks and deposits was still relatively new. A young man in Washington, DC, printed his own account's routing numbers in magnetic ink at the bottom of the deposit slips he stole from a bank. He replaced the blank deposit slips in the public areas of the bank with the doctored ones. All the slips with magnetic ink were automatically sorted and processed, diverting $250,000 of other people's money into the criminal's bank account, from which the thief withdrew $100,000 and disappeared.

Credit card numbers include check-digits that are computed using special algorithms to help prevent creation of authentic-looking account numbers. Unfortunately, programs for creating such authentic credit card accounts, complete with check digits, are widely available in the computer underground. Even children have taken to forging credit card numbers. For example, a 16-year-old Australian from Brisbane started defrauding businesses using stolen and forged credit card numbers just after leaving school. By 1997, he had stolen $100,000 in goods and services. In October 1997, he pleaded guilty to 294 counts of fraud.

## C. Extortion

Computer data can be held for ransom. For example, in an early case dating to 1971, two reels of magnetic tape belonging to a branch of the Bank of America were stolen at Los Angeles International Airport. The thieves demanded money for their return. The owners ignored the threat of destruction because they had adequate backup copies.

In the 1980s and 1990s, rumors persistently circulated in the financial community that banks and other institutions were giving in to extortion. For example, the June 3, 1996, issue of the *London Times* reported that hackers had been paid 400 million pounds sterling in extortion money to keep quiet about having electronically invaded banks, brokerage firms, and investment houses in London and New York with logic bombs (programs with harmful effects that could be launched as a result of specific conditions such as a given date or time). According to the article, banks chose to give in to the blackmail over concerns that publicity about such attacks could damage consumer confidence in the security of their systems.

In September 1999, the *Sunday Times* of London reported that British banks were being attacked by criminal hackers attempting to extort money from them. The extortion demands were said to start in the millions and then run down into the hundreds of thousands of pounds. Mark Rasch, a former attorney for computer crime at the United States Department of Justice and later legal counsel for Global Integrity, said, "There have been a number of cases in the UK where hackers have threatened to shut down the trading floors in financial institutions. . . . The three I know of (in London) happened in the space of three months last year one after the other. . . . In one case, the trading floor was shut down and a ransom paid." The International Chamber of Commerce (ICC) confirmed it had received several reports of attempted extortion.

There was a case of attempted extortion directed at a retail Web site in December 1999. A 19-year-old

Russian criminal hacker calling himself Maxus broke into the Web site of CD Universe and stole the credit card information of 300,000 of the firm's customers. When the company refused his $100,000 ransom, he posted 25,000 of the accounts on a Web site (Maxus Credit Card Pipeline). After investigation showed that the stolen card numbers were in fact being used fraudulently, 300,000 people had to be warned to change their card numbers.

In January 2000, information also came to light that VISA International had been hacked by an extortionist who demanded $10M for the return of stolen information—information that VISA spokesperson Chris McLaughlin described as worthless and posing no threat to VISA or to its customers. The extortion was investigated by police but no arrests were made.

## D. Slamming

*Slamming* is the fraudulent, unsolicited switching of long-distance services to another long-distance carrier; the practice has caused consternation among victims confronted with larger phone bills than they expected from their normal carrier. In mid-December 1996, Connecticut's Department of Public Utility Control (DPUC) was slammed by a firm called Wiltel, which converted 6 of its 14 lines to its service without authorization.

By July 1997, the United States Federal Trade Commission was overwhelmed with over 16,000 complaints from enraged customers whose long-distance telephone service had been switched without their permission. For example, the Fletcher Companies engaged in systematic slamming and the United States Federal Communications Commission (FCC), responding to over 1400 complaints, fined the group of companies $5M in April 1998. In June 2000, long-distance company WorldCom, Inc. agreed to pay $3.5M to settle an inquiry by the Federal Communications Commission into 2900 complaints from persons charging that WorldCom telemarketers illegally switched them from other phone service carriers. WorldCom president Bernard J. Ebbers said the slamming incidents were perpetrated by a few sales employees who were subsequently fired.

## E. Industrial Espionage

Teenage hackers who deface government sites or steal credit card numbers attract a lot of attention, but experts say the real problem of cybercrime is corporate-sponsored proprietary information theft committed by professionals who rarely get caught. According to a report from the American Society for Industrial Security in September 2000, Fortune 1000 companies sustained losses of more than $45 billion in 1999 from thefts of proprietary information, and a survey by the Computer Security Institute in 2000 indicated over half of 600 companies polled said they suspected their competitors were a likely source of cyberattack.

In 1995, San Jose, CA, prosecutors announced indictments in a case of industrial espionage in Silicon Valley. Two executives of the defunct Semiconductor Spares, Inc. were charged with stealing over 500 technical drawings from Lam Research Corp.

In 1996, Britain's Davy International initiated a lawsuit over industrial espionage against the Austrian firm VA Technologie AG. In another case of alleged industrial espionage that came to light in 1996, the American subsidiary of Boehringer Mannheim Corp., a pharmaceutical firm based in Germany, accused Lifescan, Inc., the diabetes-products division of Johnson & Johnson, of encouraging industrial espionage by presenting "Inspector Clouseau" and "Columbo" awards to employees who got the most information about their competitor, regardless of ethics.

In June 1997, two citizens of Taiwan were arrested after allegedly trying to bribe a Bristol–Myers Squibb Co. scientist into turning over technological secrets for the manufacture of Taxol, a drug used to fight ovarian cancer.

In 1998, Pixar, makers of the recent animated movie "Toy Story," filed suit for a restraining order barring persons unknown from spreading stolen information about the salaries of their 400 employees. The report was widely circulated on the Net and damaged the company's ability to hire and retain employees (because competitors could outbid Pixar easily and inexpensively).

In a settlement of one of the few documented cases of industrial espionage involving intercepted e-mail, the Alibris company paid a $250K fine in 1999 for the firm it acquired in 1998. That company, Interloc, admitted intercepting and copying 4000 e-mail messages sent to Amazon.com through its own ISP, Valinet. Prosecutors said that the e-mail was intercepted to gain a competitive advantage against Amazon in Interloc's own book business. The managers of Interloc steadfastly denied any wrongful intention but failed to explain why they copied the e-mail.

In June 2000, Microsoft complained that various organizations supporting Microsoft in its antitrust battle with the United States government had been victimized by industrial espionage agents who attempted to steal documents from trash bins.

Echelon, an international surveillance network, was in the news in the late 1990s. Echelon, which is jointly operated by the U.S., U.K., Australia, Canada, and New Zealand, is capable of intercepting phone, fax, and e-mail signals around the world and is intended to gather intelligence regarding terrorist and other threats to the U.S. and its allies. In 1997, the *Covert Action Quarterly,* an intelligence newsletter, reported, "Unlike many of the electronic spy systems developed during the Cold War, Echelon is designed primarily for non-military targets: governments, organizations, businesses, and individuals in virtually every country. It potentially affects every person communicating between (and sometimes within) countries anywhere in the world." In July 2000, the European Parliament renewed its attack on Echelon by forming a temporary committee to investigate whether the spy network was used for commercial espionage against European businesses. The parliament said the committee would also determine Echelon's legality. Later in 2000, a Green Party member of the European Parliament filed criminal charges in Germany against Echelon.

## F. Gambling

One of the more lucrative scams focuses on bilking credulous gamblers by offering games of chance and betting on sports and other events via the Internet. Interstate gambling is illegal in the United States, but the operators of the gambling sites have been setting up their servers in offshore locations, free of U.S. law. The likelihood that any of the games of chance are in fact programmed to be honestly conducted is unknown. In one embarrassing incident in 1998, an analyst discovered that nobody who chose the digit "9" as part of their bet ever won the Arizona Lottery's new Pick 3 game—because the algorithm was incapable of generating a 9 in the winning three-digit numbers. Observers noted that the risk of accidental or deliberate distortions of probability distributions might be even higher in software written by unknown persons working for unknown private organizations in offshore locations. If gambling is a tax on people with a limited understanding of probability, offline gambling seems like a tax on people with limited reasoning powers.

## G. Auctions

Auctions have always been a risky way to buy goods, since dishonest sellers can engage shills to pretend to bid the price of an item up beyond its value. The risk is higher when goods have no intrinsic value but depend solely on demand for determination of the price. When there is no visual contact or screening of the participants in the group bidding for an item, however, the risk is much greater.

Another aspect of online auctions is the possibility of buying stolen or illegal goods. For example, in September 1999, someone put up a human kidney for sale through the online auction house eBay and received bids of up to $5.8M. The auction service canceled the sale because selling human organs is a Federal felony with up to $250,000 in fines and at least 5 years in jail. Other offers—some of which may have been pranks—included an offer to sell a human baby; prices (possibly also from pranksters) reached over $100,000 before eBay interrupted the (illegal) sale.

Online auctions have become the most serious source of complaints to the Internet Fraud Complaint Center, a project of the FBI and the Department of Justice. In November 2000, the Center opened and began receiving more than 1000 complaints a day. However, the online auction industry denies that fraud is a serious problem, and eBay says that only 1 of every 40,000 listings has resulted in a confirmed case of fraudulent activity. Complaints about Internet fraud can be reported to http://www.ifccbi.gov.

## H. Pornography

Some monitors think that pornography is the single largest money-making use of the Internet and the World Wide Web. Pornography is governed by different standards in different countries, but all countries ban the creation, distribution, and storage of child pornography. Many pornographers use tricks such as registering their domains with misleading names; a well-known example is http://www.whitehouse.com, which plays on novices' ignorance of the naming standards (U.S. government agencies have domain names ending in .gov, not .com). Other tricks include using misspellings. At one time, for example, a pornographer registered several misspellings of "microsoft.com"; people were astonished at what they would see appearing on screen after typing, say, http://www.micosoft.com. Trademark owners have been successful in stopping this obvious abuse of their trademark through civil litigation, but the pornographers keep coming up with alternatives.

## I. Stalking and Assault

Some of the worst abuses of the new communications media have involved lies by pedophiles. These sexual predators have successfully used e-mail and especially children's chat rooms to misrepresent themselves to naïve children as if they were in the same age range. The Internet Crime Forum in the U.K. reported in December 2000 that they estimate 20% of the children online have been approached by pedophiles. Pedophiles have exacerbated conflicts between their victims and their parents, lured youngsters into concealing their communications, persuaded them to send pornographic videos of themselves, and even convinced a few to travel without parental approval for meetings with their new "friends." In January 2001, for example, a 32-year-old man was charged with raping a 14-year-old upstate New York girl he met in an Internet chat room and lured to a hotel room in Albany, NY.

### J. Libel, Misrepresentation, and Harassment

The ease with which anyone can forge the identifying information used in e-mail or use pseudonyms on discussion groups has resulted in many instances of libel, distortion, misrepresentation, and harassment. For example, criminals sent out thousands of racist, hateful e-mail messages in the name of a Texas university professor who subsequently needed police protection for his home and family. Another criminal posted a victim's phone number in chat rooms catering to phone-sex enthusiasts and described the young woman in question as a prostitute. She had to change her phone number to escape hundreds of salacious callers a day. Another kind of harassment is unsolicited commercial e-mail (often called "spam," much to the disgust of the trademark owner for the luncheon meat called *Spam*). Spammers often use anonymous e-mail identities to flood the Net with millions of unwanted advertising messages, much of it fraudulent. Some jurisdictions (e.g., Washington, Virginia, and Massachusetts) have criminalized the use of forged headers in such e-mail. Many observers predict that unsolicited commercial e-mail will eventually be regulated as unsolicited facsimile (fax) messages were in the 1980s.

A different kind of junk e-mail is hoaxes and hoax virus warnings. These nuisances spread through the ill will of pranksters who write or modify the hoaxes and, unfortunately, through the good will of credulous novices who cannot recognize the nonsense they are obediently forwarding to everyone they know. Pathognomonic signs of a hoax include:

- Absence of a specific date, name of contact, or originating organization's Web site
- Absence of a valid digital signature
- Improbably catastrophic effects or consequences of a supposed danger
- Use of exclamation marks, ALL CAPS TEXT, and presence of misspellings
- Claims that anyone can monitor exactly how many e-mails are sent with copies of the message
- Instructions to send the message to "everyone you know"

### K. Theft of Intellectual Property

Electronic communications are ideal for sharing files of all kinds; unfortunately, some people share other people's property. In 1999 and 2000, concern grew in the recording industry over the widespread pirating of music tracks through a variety of networks such as MP3.com, Napster, Gnutella, and others. At some universities, traffic in unauthorized copies of songs (and later videos) grew so frantic that available bandwidth was exhausted, leading to prohibitions on such transfers and stringent filtering at the firewalls. After extensive negotiations, several copyright-violation lawsuits, and considerable debate among people with divergent views on the ownership of commercial music and video, several facilitating companies in the U.S. agreed to cooperate with the entertainment industry to provide access to their products at reasonable cost.

## V. WHERE COMPUTERS, NETWORKS, AND SOFTWARE ARE THE TARGETS AS WELL AS TOOLS

In a sense, any attack on a computer is an attack on its users. However, this section focuses on types of crime where interference with the computing equipment and communications networks are themselves prime targets, not just incidental mechanisms in the crime.

### A. Denial of Service and Jamming

Saturating resources without falling afoul of security restrictions has been a common attack method for decades. However, such *denial of service* (DoS) attacks have grown rapidly in frequency and severity in recent years. Factors contributing to such harassment techniques include

- The explosive growth of Internet access by individuals, including children, in the 1990s
- The growing number of sites online
- Faster modems
- Widespread distribution of attack scripts
- A subculture of criminal hacking
- Easy anonymity on the Net

*E-mail bombing* is a popular method; for example, in one case a victim received 25,000 identical e-mail messages containing the single word "IDIOT." *Subscription-list bombing* involves subscribing victims to hundreds of list servers; in an early case, the criminal calling itself "Johnny [x]chaotic" harassed several dozen recipients with thousands of postings from these unwanted subscriptions. This technique is harder to use today because list servers typically now ask for a written confirmation of all subscription requests.

Another kind of DoS often occurs by mistake: *mailstorms* occur when an autoresponder sends mail to another autoresponder, which sends mail back to the originating autoresponder. Mailstorms can generate thousands of messages very quickly, causing mailboxes to reach their limits and even crashing susceptible systems. Such feedback loops can be exploited by an attacker who forges a REPLY-TO address in an e-mail message designed to spark such a storm. Mailstorms are greatly amplified when a list server can be tricked into communicating with an autoresponder.

Many other types of DoS attacks use attributes of TCP/IP. Some involve sending malformed datagrams (packets) that crash recipient processes (e.g., Ping of Death); others send bad data to a process (e.g., buffer-overflow attacks).

Towards the middle of 1999, security agencies noticed that a new generation of DoS attacks were brewing: *Distributed DoS* (DDoS). In these attacks, criminals use automatic scanning software to identify systems with known vulnerabilities and install *slave* (also known as *zombie*) programs that initiate concealed *(stealth)* processes *(daemons)* on the victimized machines. These zombies wait for encrypted instructions from a *master* program controlled by the criminals; at a specific time, hundreds or thousands of zombies can be ordered to use their host-machine's resources to send an overwhelming flood of packets to the ultimate victim machines. Such attacks materialized in February 2000, when major Web sites such as eBay, Amazon, and other high-profile systems were swamped with so much spurious traffic that they were unable to service legitimate users. Damages were estimated in the tens of millions of dollars.

## B. Penetration

The classic computer crime is penetration of a security perimeter. Such penetration has become a hobby with a subculture of criminal hackers, but it can also be part of a more serious effort to obtain information illicitly. The popular press frequently includes reports of such penetrations; perhaps one of the most spectacular recent cases in terms of publicity occurred in October 2000, when Microsoft reported that criminal hackers appeared to have entered their production systems and made copies of valuable source codes for the latest versions of its flagship MS-Windows and MS-Office products.

Most penetration occurs through exploitation of known security vulnerabilities. Although patches are known and available for new vulnerabilities within hours or days, many overworked, untrained, or careless system administrators fail to install these patches. All studies of known vulnerabilities have the same result: a majority (two-thirds and up) of all Net-connected systems have old, unpatched vulnerabilities that can be penetrated even by children *(script-kiddies)* using automated tools *(exploits, scripts)* they barely understand.

Another class of attacks involves *social engineering*, which is the hacker phrase for lying, cheating, dissimulation, impersonation, intimidation, seduction, and extortion. Criminals such as the notorious Kevin Mitnick use such techniques in persuading employees to betray user identification and authentication codes that can then be used for surreptitious access to systems.

So many Web sites are vandalized by the criminals who penetrate their inadequate security perimeters that the incidents now barely make the news. Archives of copies *(mirrors)* of the vandalized pages are available on the Web; e.g., http://www.antionline.com. Most of the vandalized pages are not suitable for viewing by children due to the presence of foul language, bad grammar, and lots of pornographic images; ironically, it is thought that most of the vandalism is *by* children, many of whose parents are delighted that their unsupervised offspring are ensconced in front of a computer "keeping out of trouble."

An important point about all penetrations is that, contrary to criminal-hacker cant, all penetrations are harmful. Criminal propaganda claims that unauthorized entry is harmless as long as no data are modified; some go further and argue even against unauthorized disclosure of confidential data. However, operations staff know that when intruders break into any system, they destroy the basis for trust of that violated system. All data and all software must be validated after every penetration; such work is tedious, difficult, and expensive.

## C. Covert Breaches of Confidentiality

Even without breaching the security perimeter in an obvious fashion, criminals can intercept confidential communications. For example, in August 1997, three New Jersey businessmen were arrested and charged with illegally intercepting and selling messages sent via a paging service to senior New York City officials such as the mayor, top police officers, and leaders of the fire department. Interception of domestic cordless telephones is an easy method for collecting information that can be used for blackmail or for sale to unscrupulous buyers. Many wireless mobile phones still use no encryption and their signals can be intercepted by commonly available equipment (with minor modifications) costing a few hundred dollars. Land-lines are easy to tap at the point-of-presence of the telephone company, at the neighborhood distribution cabinet, or—in office buildings—at the usually unlocked junction panels in basements or corridor walls.

Another form of electronic eavesdropping involves the use of *spyware*. Some software is written to allow automatic transmission of information from a user's system to specified sites on the Internet. A typical and harmless example is the registration process of many products; the user has a choice on whether to transmit information or not, and if so, how (by modem, Internet connection, fax, or mail). Spyware, in contrast, by definition conceals its transmissions. Users with firewalls that monitor inbound and outbound TCP/IP communications may be surprised by occasional requests for outbound transmission from processes they know nothing about. For example, Comet Systems cute cartoon cursors were downloaded by millions of people, many of them children. However, the free software turned out to be a Trojan: the modified programs initiated TCP/IP communications through the users' Internet connections and reported on which sites were being visited by each copy of the programs when the users went to any of 60,000 sites providing links to the cursor programs. Their purpose was to gather statistics about Web usage patterns. Company officials argued that there were no links between the serial numbers and any identifying information about the users. Privacy advocates argued that the reporting function ought to have been overt and optional.

## D. Viruses, Worms, and Trojans

Disregarding DNA, which is the ultimate self-reproducing information-storage structure, self-reproducing computer programs and processes have been around since the Bell Labs scientists started playing "Core Wars" on company mainframes in the 1960s.

### 1. Early Viruses

Hobbyists in the 1980s had more scope for their experiments because the operating systems of personal computers lacked a security kernel and therefore allowed any process to access any part of memory. Apple II microcomputer users invented computer viruses in the early 1980s such as Festering Hate, Cyberaids, and Elk Cloner. In 1983, Fred Cohen, then a student, created a self-replicating program for a VAX 11/750 mainframe at the University of Southern California. His thesis advisor, Len Adelman, suggested calling it a virus. Cohen demonstrated the virus to a security class. Cohen continued his work on viruses for several years; his Ph.D. thesis presented a mathematical description of the formal properties of viruses. He also defined viruses neatly and simply as "a computer program that can infect other computer programs by modifying them to include a (possibly evolved) copy of itself."

On October 22, 1987, a virus apparently written by two brothers in Lahore, Pakistan, was reported to the Academic Computer Center of the University of Delaware in Newark. This virus destroyed the data on several hundred diskettes at the University of Delaware and also at the University of Pittsburgh School of Business. It destroyed the graduate thesis of at least one student.

In November 1987, students at Lehigh University in Bethlehem, Pennsylvania, began complaining to the staff at the computer center that they were getting bad diskettes. At one point, 30 students returned diskettes in a single day. It turned out that there was a virus adding itself to the COMMAND.COM file on the DOS system diskettes. When the Lehigh staff examined the virus, they discovered that it was programmed to copy itself 4 times after each infection. On the 4th replication for any given copy, the virus would destroy the file allocation table of the diskette or hard disk, making the data unrecoverable (at that time, there were no utilities available for reconstituting files easily once the pointers from cluster to cluster on the disk had been lost). Several hundred students lost their data.

Until 1995, there were two main virus vectors and therefore types: *boot-sector* viruses and *file-infectors*. There were a few thousand distinct kinds of viruses (defined by signature strings of specific recognizable executable code) and industry surveys suggested that the rate of infection (measured in terms of numbers of PCs infected) was rising tenfold per year. Viruses were restricted to single platforms: MS-DOS, MS-Windows, and the Apple Macintosh operating system. UNIX and other operating systems with real security features were largely unaffected.

In August 1995, everything changed. Reports appeared of a new form of harmful self-replicating code: macro-language viruses. The first instance, dubbed "winword.concept" by antivirus specialists, contained no harmful payload: it merely contained text explaining that it was an illustration of the concept of *macro viruses.* Within the next few years, macro viruses came to dominate the lists of virus types. By January of 2001, there were over 56,000 viruses in antivirus laboratories, of which more than half were macro viruses. However, in the wild, almost all infections were by macro viruses. In the *2000 Annual Virus Prevalence Survey* run by ICSA Labs, there were no significant reports of boot-sector or file-infector viruses in the population studied.

The dominance of macro viruses is due to their cross-platform capability. Microsoft decided to ignore warnings by security specialists and incorporated extensive macro capabilities into its MS-Office products—products that run under a number of different operating systems. The default state allows automatic execution of such macros without direct user intervention, leading to the situation we face today. The problem has been exacerbated in the final years of the 1990s because Microsoft also decided to incorporate automatic execution of *any* executable attachment to e-mail received in its MS-Outlook products.

## 2. Worms

### a. Early Worms

In December 1987, a German student released a self-reproducing program that exploited electronic mail networks on the ARPANET and BITNET networks. This program would display the request, "Please run me. Don't read me." While the victim ran the program, it displayed a Christmas tree on screen; at the same time, it used the victim's e-mail directory and automatically sent itself to everyone on the list. Because this rogue program did not embed itself into other programs, experts call it the Christmas-Tree Worm.

Unfortunately, this worm had no mechanism for remembering where it had come from. Since most people to whom we write include our names in their address list, the worm usually mailed itself back to the computer system from which it had originated as well as to all the other computer systems named in the victim's directory. This reflection from victim to infector is reminiscent of an uncontrolled nuclear chain reaction. The greater the number of cross references among e-mail address directories, the worse would be the growth of the worm.

The original version of this worm worked only on IBM VM/VMS mainframe computers; luckily, there weren't very many of them on the ARPANET and BITNET networks. However, a source-code version of the worm was installed into the IBM internal e-mail network and recompiled. Because of the extensive cross-references in the e-mail system, where many employees corresponded with hundreds of other employees, the worm reproduced explosively. According to Phillips, the network was clogged for 3 hours before IBM experts identified the problem, wrote an eradicator, and eliminated the worm.

### b. The Morris Worm of 1988

The first worm that garnered worldwide attention was a self-reproducing program launched at 17:00 EST on November 2, 1988, by Robert T. Morris, a student at Cornell University in Ithaca, New York. In addition to sending itself to all the computers attached to each infected system, the worm superinfected its hosts just like the Christmas-Tree Worm had done, leading to slowdowns in overall processing speed. By the next morning, the Internet was so severely affected by the multitudes of copies of the worm that some systems administrators began cutting their networks out of the Internet. The Defense Communications Agency isolated its Milnet and Arpanet networks from each other around 11:30 on November 3.

By late November 4, a comprehensive set of patches was posted on the Internet to defend systems against the Worm. That evening, the author of the Worm was identified. By November 8, the Internet seemed to be back to normal. A group of concerned computer scientists met at the National Computer Security Center to study the incident and think about preventing recurrences of such attacks. The affected systems were no more than 5% of the hosts on the Internet, but the incident alerted administrators to the unorganized nature of this worldwide network. The incident contributed to the establishment of the Computer Emergency Response Team Coordination Center at the Software Engineering Institute of Carnegie-Mellon University, whose valuable Web site is http://www.cert.org.

In 1990, Morris was found guilty under the Computer Fraud and Abuse Act of 1986. The maximum penalties included 5 years in prison, a $250,000 fine, and restitution costs. Morris was ordered to perform 400 hours of community service, sentenced to 3-years probation, and required to pay $10,000 in fines. He was expelled from Cornell University. The Supreme Court of the United States upheld the decision by declining to hear the appeal launched by his attorneys.

#### c. The Melissa Worm

On Friday, March 26, 1999, the CERT-CC received initial reports of a fast-spreading new MS-Word macro virus. *Melissa* was written to infect such documents; once loaded, it used the victim's MAPI-standard e-mail address book to send copies of itself to the first 50 people on the list. The virus attached an infected document to an e-mail message with the subject line "Subject: Important Message From <name>" where <name> is that of the inadvertent sender. The e-mail message read, "Here is that document you asked for . . . don't show anyone else ;-)" and included an MS-Word file as an infected attachment. The original infected document "list.doc" was a compilation of URLs for pornographic Web sites. However, as the virus spread it was capable of sending any other infected document created by the victim.

Because of this high replication rate, the virus spread faster than any previous virus in history. On many corporate systems, the rapid rate of internal replication saturated e-mail servers with outbound automated junk e-mail. Initial estimates were in the range of 100,000 downed systems. Antivirus companies rallied immediately and updates for all the standard products were available within hours of the first notices from CERT-CC.

The Melissa macro virus was quickly followed by the PAPA MS-Excel macro virus with similar properties.

#### d. The Love Bug

In May 2000, the I LOVE YOU ("Love Bug") computer worm struck computers all over the world, starting in Asia, then Europe. The malicious software spread as an e-mail attachment, sending itself to all the recipients in standard e-mail address books. Within days, new variants appeared; for example, one variation used a subject line purporting that the carrier message contained a joke. These worms not only spread via e-mail, they also destroyed files on the infected systems.

Within a week, Philippine authorities detained several young people for questioning after identifying the computer used to launch the worm. On May 11, Filipino computer science student Onel de Guzman of AMA Computer College in Manila told authorities that he may accidentally have launched the Love Bug but he did not take responsibility for creating it, saying in Tagalog, "It is one of the questions we would rather leave for the future." All suspects were released without prosecution because of the absence of laws in their country that would criminalize their alleged actions.

### 3. Trojans

#### a. Early Trojans

Helpful volunteers in the early 1980s distributed a great deal of useful software for free; such *freeware* became a blind for malefactors who wrote harmful programs but described them as useful utilities. In March 1988, users noticed a supposed improvement to the well-known antivirus program Flu-Shot-3. Flu-Shot-4 was a Trojan, however, and it destroyed critical areas of hard disks and floppy disks. One of the interesting aspects of this Trojan was that it was an early user of the *stealth technique* of self-modifying code: the harmful assembler instructions were generated only when the program was run, making it harder for conventional antivirus signature scanner programs to identify it.

Other famous early Trojans included the supposed keyboard driver KEYBGR.COM which displayed a smiley face that moved randomly around on screen, and the 12-Tricks Trojan, which was advertised as a hard-disk diagnostic program but actually caused a wide range of damage such as garbling print output and reformatting hard disks. A particularly notorious Trojan was the PC Cyborg or AIDS Trojan, which claimed to be an AIDS information program but actually used a simple monoalphabetic character substitution code to scramble the names of all files and directories as well as using up all free space on disk and issuing fake error messages for all DOS commands.

#### b. The Moldovan Pornography Scam

In late 1996, viewers of pornographic pictures on the http://www.sexygirls.com site were in for a surprise when they got their next phone bills. Victims who downloaded a "special viewer" were actually installing a Trojan Horse program that silently disconnected their connection to their normal ISP and reconnected them (with the modem speaker turned off) to a number in Moldova in central Europe. The long-distance charges then ratcheted up until the user disconnected the session—sometimes hours later, even when the victims switched to other, perhaps less prurient, sites. AT&T antifraud staff spotted the problem because of unusually high volume of traffic to Moldova, not usually a destination for many U.S. phone calls. A federal judge in New York City ordered the scam shut down. In November 1997, the U.S. Federal Trade Commission won $2.74M from the bandits to refund to the cheated customers.

### c. Back Orifice

In July 1998, *The Cult of the Dead Cow* (cDc, a long-running group supporting criminal hacking activities) announced BackOrifice (BO), a tool for analyzing and compromising MS-Windows security and named as a spoof on the *Back Office* product from Microsoft. The author, a hacker with the L0PHT group (http://www.10pht.com), described the software as follows: "The main legitimate purposes for BO are remote tech support aid, employee monitoring and remote administering [of a Windows network]." However, added the cDc press release, "Wink. Not that Back Orifice won't be used by overworked sysadmins, but hey, we're all adults here. Back Orifice is going to be made available to anyone who takes the time to download it [read, a lot of bored teenagers]." The product featured image and data capture from any Windows system on a compromised network, an HTTP server allowing unrestricted I/O to and from workstations, a packet sniffer, a keystroke monitor, and software for easy manipulations of the victims' Internet connections. BO's description qualified it as a Trojan that allowed infection of other applications and used stealth techniques to erase its own visibility once loaded into memory. Security experts pointed out that the key vulnerability allowing BO to contaminate a network was the initial step—running a corrupted application that would load the parasitic code into memory. Users should not download software from unknown sites or execute attachments to e-mail without assurance of their legitimacy. All the major firms offering anti-malicious-code software issued additions to their signature files to identify the Trojan code.

About 15,000 copies of BO were distributed to Internet Relay Chat users by a malefactor who touted a "useful" file *(nfo.zip)* that was actually a Trojan dropper for BackOrifice.

In July 1999, cDc released BackOrifice 2K (BO2K), usually installed illegally on victim machines through a contaminated vector program that has been thereby transformed into a Trojan Horse dropper. BO2K allowed complete remote control and monitoring of the infected PCs. BO2K was noteworthy because it attacked WindowsNT workstations and servers and thus had even more serious implications for information security. Antivirus companies worked feverishly immediately after the release of the tool to update their virus-signature files. A criminal hacker calling himself Deth Veggie insisted that the CDC was involved in guerilla quality assurance—their penetration tools, he argued, would force Microsoft to repair the "fundamentally broken" Windows operating systems. Security specialists disagreed, saying that writing and releasing such tools was definitely malicious and was primarily damaging innocent users.

## E. Logic Bombs

A logic bomb is a program which has deliberately been written or modified to produce results when certain conditions are met that are unexpected and unauthorized by legitimate users or owners of the software. Logic bombs may be within stand-alone programs or they may be part of worms or viruses. An example of a logic bomb is any program which mysteriously stops working 3 months after, say, its programmer's name has disappeared from the corporate salary database.

In 1985, a disgruntled computer security officer at an insurance brokerage firm in Texas set up a complex series of Job Control Language (JCL) and RPG programs described later as "trip wires and time bombs." For example, a routine data retrieval function was modified to cause the IBM System/38 midrange computer to power down. Another routine was programmed to erase random sections of main memory, change its own name, and reset itself to execute a month later.

In 1988, a software firm contracted with an Oklahoma trucking firm to write an application system. The two parties disagreed over the quality of the work and the client withheld payment, demanding that certain bugs be fixed. The vendor threatened to detonate a logic bomb which had been implanted in the programs some time before the dispute unless the client paid its invoices. The client petitioned the court for an injunction to prevent the detonation and won its case on the following grounds:

- The bomb was a surprise—there was no prior agreement by the client to such a device.
- The potential damage to the client was far greater than the damage to the vendor.
- The client would probably win its case denying that it owed the vendor any additional payments.

In public discussions among computer programmers and consultants, some have openly admitted installing such logic bombs in their customers' systems as a tool for extorting payment.

In 1998, a network administrator for Omega Engineering was convicted of activating a digital time bomb that destroyed the company's most critical manufacturing software programs. The company claimed more than $10 million in damages and lost productivity.

## F. Sabotage

The quintessential sabotage story concerns the National Farmers Union Service Corporation of Denver, where a Burroughs B3500 computer suffered 56 disk head crashes in 2 years starting in 1970. Down time was as long as 24 hours per crash, with an average of 8 hours per incident. Technicians guessed that the crashes were due to bad power; the company spent $500,000 upgrading their power. The crashes continued.

The investigators began wondering about sabotage; all the crashes had occurred at night—specifically during a trusty operator's shift, old helpful Albert. Management installed a closed-circuit TV (CCTV) camera in the computer room—without informing Albert. Film of the next crash showed good old Albert opening up a disk cabinet and poking his car key into the read/write head solenoid, shorting it out, and causing the 57th head crash.

Psychologists determined that Albert had been ignored and isolated for years in his endless night shift. When the first head crashes occurred spontaneously, he had been surprised and excited by the arrival of the repair crew. He had felt useful, bustling about, telling them what had happened. When the crashes had become less frequent, he had involuntarily, and almost unconsciously, recreated the friendly atmosphere of a crisis team. He had destroyed disk drives because he needed company.

Many other cases of sabotage involve disgruntled employees or ex-employees.

However, other cases do involve outsiders. For example, in the late 1980s, a New Jersey magazine publisher's voice mail system was corrupted by a 14-year old boy and his 17-year old cousin, both residents of Staten Island. The younger child had ordered a subscription to a magazine dedicated to Nintendo games and never received the colorful $5 poster he had been promised. In retaliation, the children entered the company's voice mail, cracked the maintenance account codes, and took over the system. They erased customer messages, changed employees' answering messages, and generally wreaked havoc. Their actions resulted in lost revenue, loss of good will, loss of customers, expenses for time and materials from the switch vendor, and wasted time and effort by the publisher's technical staff. Total costs were estimated by the victim at $2.1M.

We have already seen that Web-site defacement, a form of sabotage, is so common that it no longer warrants much news coverage.

## G. Counterfeit Software

All over the world, opportunistic criminals make illegal copies of copyrighted software. The problem is particularly serious throughout Asia, where some countries have more than 99% of all software in pirated form; however, counterfeit software is big business even in the U.S. For example, in June 2000, Pennsylvania State Police cracked a global software piracy operation involving at least $22M in counterfeit Microsoft software. Police collected over 8000 copies of Windows 98, Microsoft Office, and Windows NT and more than 25,000 counterfeit end-user license agreements. Authorities pointed out the following warning signs of counterfeit software:

- Impossibly low prices
- Unwillingness of companies or individuals to verify their identity or contact information
- Online distributors with inadequate descriptions of return and warranty policies
- Nonstandard packaging such as a CD in a jewel case but with no documentation or authentication marks

An unfortunate side-effect of the ease with which ordinary users can copy software—including even burning their own CD-ROMs—is that many adults and especially children have no clear conception that there is anything wrong with making copies of software for their friends and even for sale. In the U.S., however, penalties for copyright violations can reach as high as fines of $250,000 *per title* and up to 5 years in prison.

## SEE ALSO THE FOLLOWING ARTICLES

Computer Viruses • Copyright Laws • Electronic Payment Systems • Encryption • Ethical Issues • Firewalls • Forensics • Law Firms • Privacy • Security Issues and Measures • Software Piracy • Year 2000 (Y2K) Bug Problems

## BIBLIOGRAPHY

Bosworth, S., and Kabay, M. E., eds. (2002). *Computer security handbook,* 4th ed. New York: Wiley.

Cavazos, E., and Morin, G. (1996). *Cyberspace and the law: Your rights and duties in the on-line world.* Cambridge, MA: MIT Press.

Fialka, J. J. (1997). *War by other means: Economic espionage in America.* New York: Norton.

Fraser, B. (Ed.) (1997). Site security handbook, RFC2196 (Network Working Group). Available at http://www.cis.ohio-state.edu/htbin/rfc/rfc2196.html.

Freedman, D. H., Mann, C. C. (1997). *@Large: The strange case of the world's biggest Internet invasion.* New York: Simon & Schuster.

Howard, J. D. (1997). *An analysis of security incidents on the Internet 1989–1995.* Ph.D. Thesis, Department of Engineering and Public Policy, Carnegie Institute of Technology, Carnegie Mellon University. Available at http://www.cert.org/research/JHThesis/Start.html.

Icove, D., Seger, K., and VonStorch, W. (1995). *Computer crime: A crime fighter's handbook.* Sebastopol, CA: O'Reilly & Associates.

Lessig, L., Post, D., and Volokh, E. (1997). Cyberspace law for non-lawyers. Published via e-mail, http://www.ssrn.com/update/lsn/cyberspace/csl_lessons.html.

Littman, J. (1996). *The fugitive game: Online with Kevin Mitnick—The inside story of the great cyberchase.* Boston: Little, Brown and Company.

Parker, D. B. (1998). *Fighting computer crime: A new framework for protecting information.* New York: Wiley.

Power, R. (2000). *Tangled web: Tales of digital crime from the shadows of cyberspace.* Indianapolis: Que.

Schwartau, W. (1996). *Information warfare,* 2nd ed. New York: Thunder's Mouth Press.

Shimomura, T., and Markoff, J. (1996). *Takedown: The pursuit and capture of Kevin Mitnick, America's most wanted computer outlaw—By the man who did it.* New York: Hyperion.

Slatalla, M., and Quittner, J. (1995). *Masters of deception: The gang that ruled cyberspace.* New York: HarperCollins.

Smith, G. (1994). *The virus creation labs: A journey into the underground.* Tuscon, AZ: American Eagle Publications.

Sterling, B. (1992). *The hacker crackdown: Law and disorder on the electronic frontier.* New York: Bantam Doubleday Dell.

Stoll, C. (1989). *The cuckoo's egg: Tracking a spy through the maze of computer espionage.* New York: Simon & Schuster.

Tipton, H. F., and Krause, M. (Eds.) (2000). *Information security management handbook,* 4th ed. Boca Raton, FL: Auerbach.

# Cybernetics

**Asterios G. Kefalas**
*University of Georgia*

> Cybernetics, or control and communication in the animal and machine.
>
> The full importance of an epoch-making idea is often not perceived in the generation it is made.
>
> *Alfred Marshall, 1920*

## GLOSSARY

**adaptability** The ability of a system to learn and to alter its internal structure and operations in response to changes in the external environment.

**attributes** Properties of objects or relationships that manifest the way something is known, observed, or introduced in a process.

**black box** A technique that considers a system only in terms of inputs and outputs and whose internal mechanisms are unknown or unknowable.

**complexity** The property of a system resulting from the interaction of four main determinants: the number of elements, their attributes, the number of interactions among the elements, and the degree of organization of the elements.

**cybernetics** The science of control and communication in the animal and machine.

**cybernetic systems** Systems that are characterized by extreme complexity, probabilism, and self-regulation.

**entropy** A measure of the degree of disorder; a natural tendency of systems to fall into a state of disorder; the loss in quality of energy accompanying all transformation processes.

**environmental scanning** The process of acquiring information about the external environment.

**feedback** The reintroduction into a system of a portion of the system's output known as the error ($e$), which is the difference between the goal and actual performance. If it minimizes the error then it is called negative feedback. If it amplifies the error then it is positive feedback. Negative feedback ensures control; positive feedback accounts for growth.

**goal-seeking systems** Systems whose behavior is determined by the degree of accomplishment of the goal.

**homeostasis** The maintenance of a given (same) state of a system.

**information** The element that tells matter and energy what to become or what form to take. It is measured in bits. One bit of information is the amount of uncertainty removed once one of two equally probable alternatives (e.g., heads or tails) has been specified.

**probabilism** A doctrine that asserts that statements about the behavior of exceedingly complex systems must be accompanied by a probability, a number between 0 and 1.

**uncertainty** The inability to make a statement about the future state of a system. It is measured as complexity or disorder by using Boltzmann's formula, $S = k \log W$.

## I. INTRODUCTION

When, in the middle of the 1940s, Norbert Wiener coined the term *cybernetics,* he had no idea that, half a century later, this word would have become so popular, ubiquitous, pervasive, and revolutionary. By the same token, few cyberpunks realize today that the world in which they now live was once a pure vision expressed in a number of mathematical equations littering the blackboard in Room 2-224 at MIT. Yet today there is hardly any human being over 3 years old who does not know or use the first half of this word. In the early 1970s science fiction writers produced some popular television serials, such as *The Cyborgs* and *The Six Million Dollar Man,* that dramatized the basic idea of cybernetics: that humans and machines can be combined to create a perfectly controlled system that will always achieve its purpose. Today the word *cyber* has become the first part of numerous combination words such as *cyberspace, cybermall,* and *cybercrime.* In this article we attempt to take the mystery out of cybernetics by explaining its origins and meaning, its main building blocks, its main principles and applications, and their importance for the business community and society in general.

## II. BRIEF HISTORY OF CYBERNETICS

In the twilight years of the 19th century inquisitive scholars began to suspect that nature could no longer be seen as matter and energy alone. In addition, and perhaps more importantly, they realized that nature's secrets could only be unlocked by adding to the existing laws of physics and chemistry a new theory. This theory could provide a set of principles and tools that would enable humans to understand that which "informs" the material world to do what it is doing and to become what it is becoming. This is the *theory of information.* Thus, nature must be interpreted as matter, energy, and information. Aristotle established the foundations of a theory of information as a change agent, i.e., as that which makes the possible actual. According to Campbell, Aristotle proposed four types of causes responsible for bringing about changes in the world.

Cause number one is *matter,* without which nothing would happen at all. Cause number two is the *form* implicit in the thing which changes. The form is its meaning, determining what kind of thing it is, "what can be said" about it. Cause number three is the *efficient cause,* an active agent of change, a mover, a force or power that brings about what matter only makes possible. Finally, cause number four is the *end* or *purpose* that a thing naturally tends to approach when it changes.

Cybernetics is about understanding and managing these four forces of change. A quick review of its more recent historical antecedents must start with the efforts of late 19th-century thinkers who questioned the adequacy of Newtonian laws as an all-explaining set of rules for life on earth. One of the first phenomena that attracted the attention of physicists was the transformation of energy from one type into another. Since Newton's mechanical explanation of the movements of the heavenly bodies, once thought to be guided by God, focused on achieving equilibria states rather than on changes in the states, transformation processes were of no interest. When scientists directed their searchlights away from heavenly bodies and toward earthly objects they noticed that these objects were in a constant state of change. Even though the amount of energy in the universe does not change (i.e., energy is indestructible), it can be converted into another type of energy at a fixed rate of exchange. In this exchange the amount of energy is constant; however, its quality may change. Rudolf Clausius identified this loss of quality of energy as *entropy* and expressed it as a fraction: heat divided by temperature. Clausius declared in his second law of thermodynamics that every transfer of energy is accompanied by a net irreversible increase in entropy. Clausius summed up his conclusions in the famous couplet:

> The energy of the universe is a constant. The entropy of the universe tends to a maximum.

Thus humanity once again was confronted with a similar question that was answered in a definite way by Sir Isaac Newton more than 100 years earlier. Instead of asking the question that it was asked back then, "Why do things fall to the ground?," physicists now ask: "Why does energy suffer this curious one-way decay?" Could it be possible to answer this question with the precision and elegance of Newton's laws? Ludwig von Boltzmann's (1844–1906) efforts to create a theory of atomism in the late 1800s provided a measure for this decaying process of entropy maximization, and Norbert Wiener in late 1940 provided its therapy—a theory of control and communication that uses information to arrest entropy that prevents a system from performing useful work.

In a paper presented at the Imperial Academy of Sciences, Boltzmann told his colleagues that

> Precisely those forms of energy that we wish to realize in practice are however always improbable. For example, we desire that a body move as a whole; this requires all its molecules to have the same speed and the same direction. If we view molecules as independent individuals, this is however the most improbable case conceivable. It is well known how difficult it is to bring even a moderately large number of individuals to do exactly the same thing in exactly the same manner.

Boltzmann defined this improbable order as complexity and provided a set of mathematical tools to measure it. His basic idea was that the degree of complexity will somehow be related to the number of independent individuals in an entity. Unable to convince his fellow scientists of the importance and necessity of focusing on the atomic level of the material world Boltzmann took his life. After considerable work and some 20 years later, Max Planck and Werner Heisenberg provided the formula

$$S = k \log W \tag{1}$$

where $S$ is complexity, $k$ is Boltzmann's universal constant, and $W$ is the number of alternatives (or the number of ways in which the parts of a system can be arranged).

The entropy $S$ reaches a maximum when all the parts of the system are so thoroughly mixed up that there is no reason to expect the system to be in one state or another. When that happens our knowledge of the state of the system is at a minimum. In sum, Boltzmann told us that the higher the entropy, the less information we can have about the system.

Despite these great developments, a "real" theory of information was not developed until the early 1940s, until the ideas of entropy, uncertainty, and information were related and became the building blocks of a new science. Two very different figures played a pivotal role in this new science. The first one was Claude Shannon. Shannon was unassuming, slow to push his ideas before the public, seemingly indifferent to applause and recognition, and a famously unprolific writer. The second figure was Norbert Wiener (1894–1964). Wiener was almost the opposite of Shannon: a florid and eccentric character, a blower of fanfares for his own considerable accomplishments. Wiener would walk into some of his colleagues' rooms, puffing at a cigar, and say: "Information is entropy." Although many others sensed the connection between entropy and information it was clearly Wiener whose "mathematical carpentry" became the essential building blocks of the new and emerging science of cybernetics.

Cybernetics dates from the time of Plato, who, in his *Republic,* used the term *kybernetike* (a Greek term meaning "the art of steersmanship") both in the literal sense of piloting a vessel and in the metaphorical sense of piloting the ship of state, that is, the art of government. From this Greek root was derived the Latin word *gubernator,* which, too, possessed the dual interpretation, although its predominant meaning was that of a political pilot. From the Latin, the English word *governor* is derived. It was not until Watt termed his mechanical regulator a "governor" that the metaphorical sense gave way to the literal mechanical sense. It was this event that in 1947 provided the motivation for Norbert Wiener to coin the term *cybernetics* for designating a field of studies that would have universal application. With this the term has now come full circle.

In more recent times the science of cybernetics has been much abused by writers who equated it exclusively with electronic computers, automation, operations research, and a host of other tools. In his classic text, Wiener defines cybernetics as the *science of control and communication in the animal and machine.* It is quite evident that Wiener intended cybernetics to be concerned with universal principles applicable not only to engineering systems but also to living systems. In Wiener's 1954 words:

> In giving the definition of Cybernetics in the original book, I classed communication and control together. Why did I do this? When I communicate with another person, I impart a message to him, and when he communicates back with me he returns a related message which contains information primarily accessible to him and not to me. When I control the actions of another person, I communicate a message to him, and although this message is in the imperative mood, the technique of communication does not differ from that of a message of fact. Furthermore, if my control is to be effective I must take cognizance of any messages from him which may indicate that the order is understood and has been obeyed. . . . When I give an order to a machine, the situation is not essentially different from that which arises when I give an order to a person. In other words, as far as my consciousness goes, I am aware of the order that has gone out and of the signal of compliance that has come back. . . . Thus the theory of control in engineering, whether human or animal or mechanical, is a chapter in the theory of messages.

This view of control can be profitably applied at the theoretical level of any system and to diverse disciplines in both large and small systems. As Wiener states, "It is the purpose of Cybernetics to develop a language and techniques that will enable us indeed to attack the problem of control and communication in general, but also to find the proper repertory of ideas and techniques to classify their particular manifestations under certain concepts."

To better understand the true meaning and usefulness of cybernetics, let us take a closer look at the definition given by Wiener. Four main concepts are included in this definition, which can be arranged into two sets: (1) control and communication, and (2) animal and machine. What this definition implies is that the concepts of control and communication are two sides of the same coin. The desired outcome (i.e., control) is achieved by the process of communication and the process of communication is triggered by the lack of control. Thus in systems that are always in perfect control, there is no need for communication. However, such systems do not really exist; or if they do exist they are of no interest because they will eventually fall victims to the second law of thermodynamics, the law of maximum entropy, and will die.

Finally, implicit in this definition of cybernetics is the idea that this relationship between control and communication holds for mechanical systems (i.e., machines), as well as for human systems (i.e., animals). This latter statement became the impetus for science fiction writings and television dramatizations. In addition, this statement was misinterpreted by a number of scholars and motivated Wiener to write a popular book on the subject under the title *The Human Use of Human Beings*. What people foresaw back then is what actually happened today: The human and the machine became one system, an information processing system.

## III. CYBERNETIC SYSTEMS*

Stafford Beer, a brilliant British systems scholar, is the first and only man to declare in his passport *cybernetician* as a profession. He, more than anybody else, has contributed immensely to the popularization of cybernetics and its application to management. One of his greatest contributions is his classification of cybernetic systems. In his seminal work *Cybernetics and Management,* Beer used a scheme that classifies systems in accordance with two criteria: *complexity* and *predictability*. The first criterion, complexity, refers to the structure of the system, whereas the second one, predictability, relates to its behavior. Table I provides a presentation of the possible combinations of these two criteria and their specific attributes.

With respect to the first criterion, Beer uses three degrees of complexity: simple, complex, and exceedingly complex. A simple system is one that has few components and few interrelationships; similarly, a system that is richly interconnected and highly elaborate is complex, and an exceedingly complex system is one that cannot be described in a precise and detailed fashion. The second criterion concerns the system's predictability of its behavior. A system whose parts interact in a perfectly predictable way, and consequently make its behavior predictable, is a deterministic system. On the other hand, a system whose parts do not behave in a predictable manner is a probabilistic system. In the former the observer or the manager of a system can make a definite statement about its present and future state, whereas in the latter any such statement must be qualified by the addition of a statement regarding the probability associated with the likelihood that the statement will be true.

The six categories of the two criteria, one threefold and the other twofold, are presented in Table I. Although Beer is clear in his admonition that these bands are hazy and that they represent merely bands of likelihood, still such a scheme has value since his grouping is done according to the kinds of *control* to which they are susceptible. Not all categories are of equal difficulty and of equal importance. Deterministic systems are of little interest because behavior is predetermined and because they do not include the organization as does an open system. As shown in Table I, examples of this type of system include the pulley, billiards, a typewriter, most machines in organizations, the movement of parts on an assembly line, the automatic processing of checks in a bank, and so on. In each of the above examples, the output of the system is controlled by management of the input to the system.

From simple deterministic systems one moves to complex deterministic ones, the singular difference being the degree of complexity involved. The computer is illustrative of this class of system in that it is much more complex than the previously mentioned systems but still operates in a perfectly predictable manner. The point made earlier that the band separating the categories is hazy is demonstrated by the fact that to a computer specialist the computer may not be complex. In a similar manner, the automobile engine is complex for many, but again for a mechanic it is a simple deterministic system. In all of the above examples there is only a single state of nature for the system, which is determined by the structural arrangement of the elements composing it. If these are in the proper configuration, the system will operate in a predetermined pattern.

If one were to introduce a second state of nature into each of the above systems, they would become probabilistic. As seen from Table I, probabilistic systems can range from the simplest games of chance, such as the flipping of a coin, in which only two possible states can exist, to the organization, in which multiple states are possible. As the complexity of a probabilistic system and the number of states of nature increase, prediction and control of systems behavior become extremely difficult. Thus, while in deterministic systems control of the inputs will provide prediction of the outputs, in probabilistic systems con-

*Sections III through VI draw heavily from *Management Systems: Conceptual Considerations,* P. P. Schoderbek, C. G. Schoderbek, and A. G. Kefalas, by permission of the senior author.

**Table I** Classification of Systems Based on Susceptibility to Control

| Predictability/Complexity | Simple | Complex | Exceedingly complex |
|---|---|---|---|
| Deterministic (one state of nature) | Pulley<br>Billiards<br>Typewriter | Computer<br>Planetary system | Empty set |
| Type of control required | Control of inputs | Control of inputs | Control of inputs |
| Probabilistic (many states of nature) | Quality control<br>Machine breakdowns<br>Games of chance | Inventory levels<br>All conditional behavior<br>Sales | Firm<br>Humans<br>Economy |
| Type of control required | Statistical | Operations research | Cybernetic |

*Source:* Adapted from Stafford Beer, *Cybernetics and Management,* Science Edition (New York: John Wiley, 1964), p. 18.

trol of the inputs will provide only a *range of possible outputs.* This is the case with the last category of exceedingly complex, probabilistic systems which includes the firm, the individual, and the economy, all of which can exhibit variable states of nature. The firm, being composed of multiple subsystems, interacts with other external systems such as the government, competitors, unions, suppliers, and banks. The interaction of the various internal departments and components of an organization and its external subsystem is so intricate and dynamic that the system is impossible to define in detail.

What, then, is of concern for cybernetics is a system whose structure is very complex and whose behavior is probabilistic. As noted in Table I, simple probabilistic systems are controlled through statistical methods, whereas complex probabilistic systems are dealt with through more sophisticated methods. These tools serve adequately in dealing with systems exhibiting a measure of complexity, but in treating exceedingly complex systems, which lack definability, they are deficient. Highly complex systems will not yield to the traditional analytical approach because of the morass of indefinable detail; yet these too must be controlled. Ironically, the same complexity that makes these systems indescribable in details provides the key to their controllability. This is Beer's third characteristic of cybernetic systems, and that is *self-regulation.* The self-regulatory feature of cybernetic systems is essential if systems are to maintain their structure.

## IV. UNDERSTANDING AND MANAGING CYBERNETIC SYSTEMS

The domain of cybernetics is systems that are exceedingly complex, probabilistic, and self-regulating. Understanding these kinds of systems requires a new definition of understanding. There is a story of two physicists taking a walk on a beach in Cape Cod. One physicist asked the other, "I understand that you are teaching this new stuff about relativity theory. Do you really understand relativity theory?" The other physicist looked at him and replied, "Yes I do teach it, but to understand relativity theory one must change the definition of understanding." The new understanding is a process that involves the three Cs: conviction, convenience, and convention. When at the beginning of the 1900s physicists were confronted with the inadequacy of the conventional language to describe the structure and behavior of the atom, they invented a brand new set of concepts. The pioneers of this vocabulary creation process had to first *c*onvince their colleagues of the appropriateness of the vocabulary. Subsequent users found the use of this vocabulary *c*onvenient and finally its frequent usage became a *c*onvention. Some 50 years later computer scientists went through the same three Cs vocabulary building process. Today there is no serious dispute as to the meaning of bits, bytes, megabytes, gigabytes, terabytes, and all of the other words we need to communicate. Nobody really *understands,* in the conventional meaning of the word, what a gigabyte is, yet all of us "know" that a gigabyte is more than a megabyte.

Table II presents the main characteristics of cybernetic systems along with the tools for dealing with each of these characteristics. For example, the technique employed when dealing with extreme complexity is

**Table II** Characteristics and Tools for Analysis of Cybernetic Systems

| Characteristics of a system | Tools for analysis |
|---|---|
| Extreme complexity | Black box |
| Probabilism | Information theory |
| Self-regulation | Feedback principle |

that of the black box. There can be but little doubt that only a few of the systems encountered in the workaday world are of the deterministic type. Most are probabilistic in their behavior. Any system operating within a margin of error is probabilistic and therefore must be treated statistically. Information theory is the tool used to deal with probabilism. Finally, self-regulation requires that control must operate from within. The tool for dealing with self-regulation is negative feedback that utilizes the error or the difference between the goal and the actual performance as the means of control.

## A. Complexity and the Black Box

An explanation of the term *complexity* can be approached from many different viewpoints. From the mathematical viewpoint, complexity can best be understood as a statistical concept. More precisely, complexity can best be explained in terms of the probability of a system being in a specific state at a given time. From a nonquantitative viewpoint, complexity can be defined as the quality or property of a system that is the outcome of the combined interaction of four main determinants. These four determinants are (1) the *number of elements* comprising the system, (2) the *attributes* of the specified elements of the system, (3) the *number of interactions* among the specified elements of the system, and (4) the *degree of organization* inherent in the system; that is, the existence or lack of predetermined rules and regulations that guide the interactions of the elements or specify the attributes of the system's elements.

Most attempts at measuring the complexity of a given system usually concentrate on two criteria: the number of elements and the number of interactions among the elements. This is especially true in classical statistics situations. This kind of measure of complexity is very superficial and, to some extent, misleading. Confining oneself to these two dimensions of complexity will lead one to classify a car engine as a very complex system. There are indeed a large number of elements and an equally large number of interactions among all the parts of a car engine. By the same token, one would be inclined to classify a two-person interaction as a very simple system, because only two elements and only two possible interactions are involved.

If one were to incorporate the other two determinants of complexity, namely, the attributes of the elements and the degree of organization, then one would arrive at a different conclusion. Concerning the example of a car engine, one would observe that the interactions must obey certain rules and follow a certain sequence. One would also observe that the attributes of the system's elements are predetermined. By using all four criteria of complexity, one must conclude that the car engine is, in fact, a very simple system.

One of the merits of the black box technique (Fig. 1) is that it provides the best antidote against the tendency of the investigator to oversimplify a complex phenomenon by breaking it into smaller parts. The black box technique for dealing with complexity represents a selection procedure based on a series of dichotomies. In other words, the investigator of a complex situation manipulates the inputs to the black box and classifies the outputs into certain distinct classes

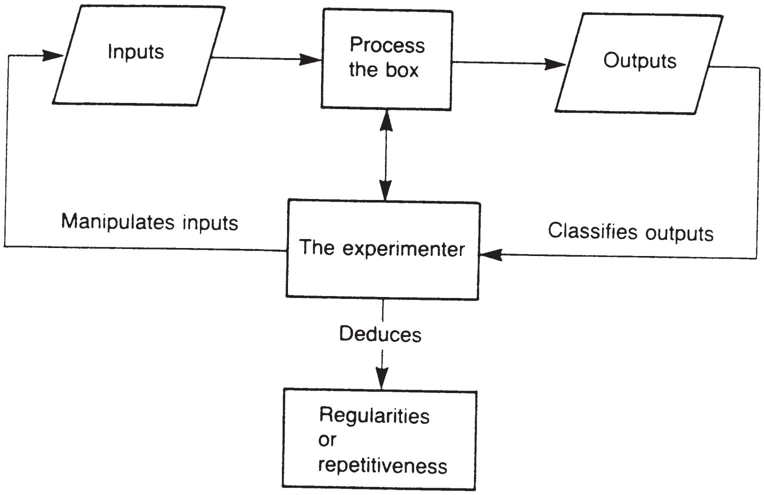


**Figure 1** The black box technique.

based on the degree of similarity of the outputs. The investigator then converts each class into a "many-to-one" transformation. The black box technique involves the following sequential steps: (1) input manipulation, (2) output classification, and, finally, (3) many-to-one transformations. The input manipulations over an extended number of trials reveal (in the output classification as recorded in the protocol) certain similarities or repetitiveness. These similarities are in turn converted into legitimate many-to-one transformations that act as implicit control devices; these many-to-one transformations account for the reduction in the system's variety without unnecessary simplifications.

## B. Probabilism and Information Theory

Information theory is a body of organized knowledge that is concerned primarily with the abstract logical nature of information, its mathematical measure, capacity of communication channel, noise, coding, speed, and accuracy of transmission, and secondarily with the meaning of the transmitted information.

### 1. The Abstract Logical Nature of Information

Information is a statistical concept, which denotes a change in probabilities indicated to the receiver as the result of actual selection among possible message states by the sender at the opposite end of the communications channel. It is evident from this definition of information that the concept refers to the mechanism of selecting, choosing, or narrowing of the range of possible alternatives about which the user of that mechanism is ignorant or uncertain as to the correctness of his choice. The fewer the number of alternatives the selector is left with after the selection process has begun, the more informed he is and, therefore, the less uncertain. In general, one could define information as the antidote for uncertainty.

### 2. Measurement of Information

The basic unit of measurement of information is the bit—a contraction of "binary digit." A source is said to have received 1 bit of information when the probability of one of two equally probable alternatives has been specified. In other words, if the receiver knows that he is confronted with two equally probable alternatives, such as the tossing of a coin when only two alternatives—heads (H) or tails (T)—are possible, and is told the existence of the one alternative—thereby removing all uncertainty—then he is said to have received 1 bit of information. Alternatively, the number of bits can be found by determining the number of times that the message or the uncertainty would have to be halved in order to achieve certainty. Mathematically, the number of bits is determined by the following formula (its resemblance to Boltzmann's formula for complexity is obvious):

$$H = -pi \log 2pi \qquad (2)$$

where $H$ is the average amount of information in bits, $pi$ is the number of alternatives, and log 2 is the binary logarithm. Note that the minus sign is needed to ensure positive amounts of information (positive $H$). Any probability is a number less than or equal to 1, and the logarithm numbers less than 1 are themselves negative.

For equally probable alternatives, the first term on the right-hand side of the equation *(pi)* is reduced to 1 (since the logarithm of 2 to the base 2 is unity), and $H$ is found by looking up $pi$ in the log 2 $pi$ table. A simple example will illustrate the point. Figure 2 depicts a situation of total uncertainty. Let's assume that a coin has been hidden under one of the 64 cells of the matrix. There is no further information available than that there are 64 alternatives *(pi)* and that each cell has an equal probability of hiding the coin. The question is how much information does one need to discover the coin? We can determine the number of bits from Eq. (2). Thus,

$$H = -pi \log 2\ pi = 1 * \log 2\ 64 = 1 * 6$$
$$= 6 \text{ bits of information}$$

Alternatively, in practical terms, one can determine the number of bits via a series of binary (yes or no) questions and by doing a series of dichotomies or by halving the 64 alternatives. As Fig. 2 shows, the answer again will be 6 bits of information, which in reality represent the answers to six yes/no questions. It is important for the question to be asked in a binary manner, leaving no choice or free will to the subject. In other words, no matter what answer the subject gives, information must be created that will remove half of the uncertainty or eliminate half of the alternatives. For example, one should not ask the question "Is the coin in cell 1–1 or not?" The answer to this type of question depends on the willingness and honesty of the subject. The question "Is the coin in cell 1–1?" will provide information no matter what the answer is. If the answer is yes, then of course all uncertainty is removed. However, if the answer is no, then that

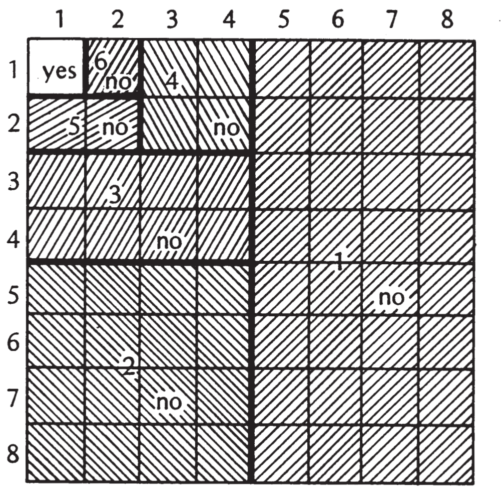


**Figure 2** Determining the number of bits by halving the alternatives.

alternative is no longer part of the total uncertainty and can be eliminated from the total uncertainty.

Following this logic one can find the specific cell by halving the possible alternatives. As can be seen from Fig. 2, this process involves successive reductions of uncertainty in a descending order of magnitude. The first question eliminates 32 possible cells resulting in 1 bit of information. The second question halves the remaining 32 alternatives and removes another 16 alternatives, and so on until the last (sixth) question involving two alternatives, of which one is eliminated leaving no uncertainty at all.

### 3. Meaning of Information

It must be said at the outset that the originators of information theory, Shannon and Weaver, who wrote the original work, *The Mathematical Theory of Communication,* in the late 1940s, did not intend to deal with the problem of the meaning of transmitted information. In Weaver's own words, relative to the broad subject of communication, there seem to be problems at three levels. Thus, it is reasonable to ask, seriously:

*Level A:* How accurately can the symbols of communication be transmitted? (The technical problem.)
*Level B:* How precisely do the transmitted symbols convey the desired meaning? (The semantic problem.)
*Level C:* How effectively does the received meaning affect conduct in the desired way? (The effectiveness problem.)

Although the mathematical theory of communication applies only to level A, Weaver noted that the theory of level A is, at least to a significant degree, also a theory of levels B and C. No real attempt was made, however, to develop a theory of the meaning of information by either Shannon or Weaver. Weaver felt that such a theory was feasible and that a communication system must be extended to include a semantic receiver and a second receiver whose function will be to match the statistical semantic characteristics of the message to the statistical semantic capacities of the totality of receivers, or of that subset of receivers that constitutes the audience one wishes to affect. Shannon, on the other hand, considered the entire semantic aspect of communication irrelevant to the engineering problem.

## C. Self-Regulation and Negative Feedback

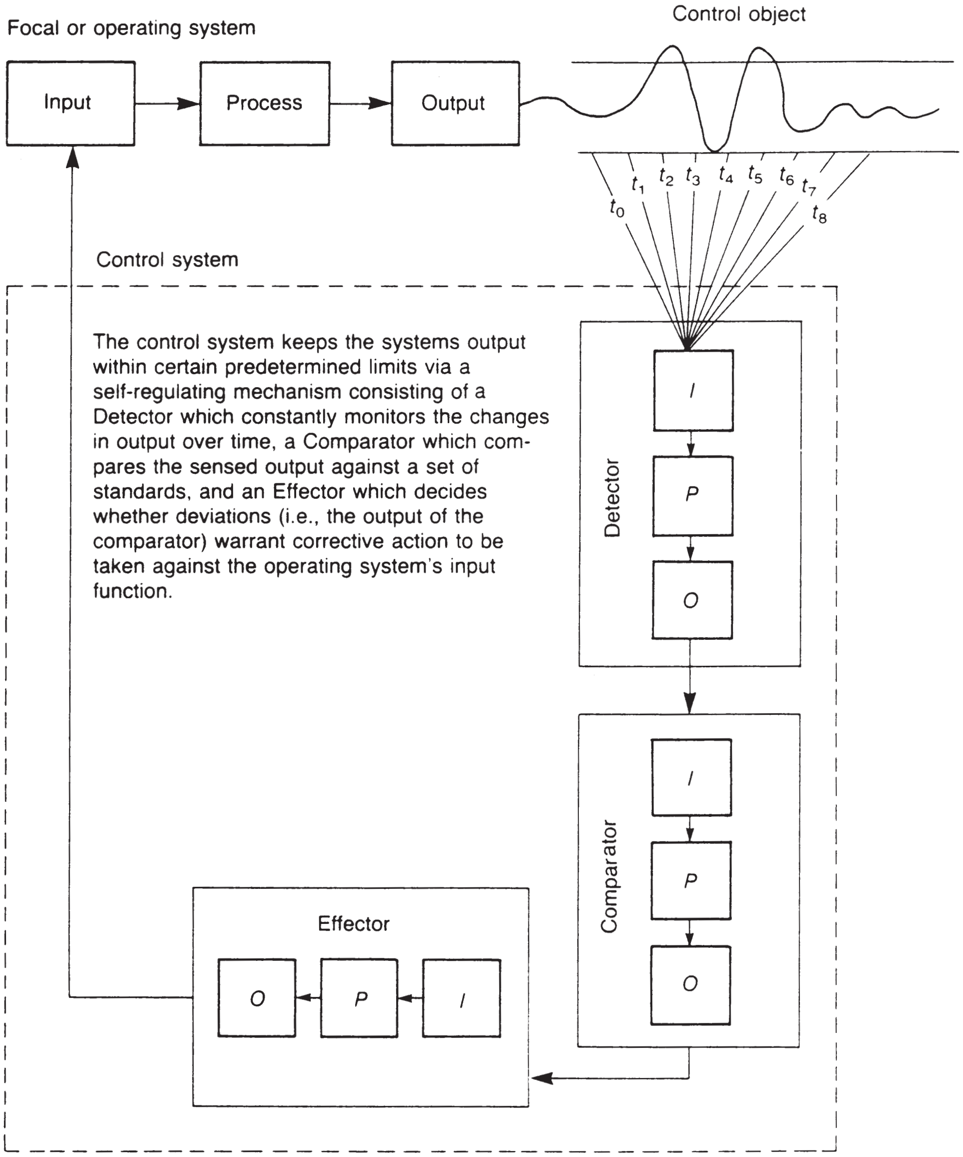
Self-regulating mechanisms are subsystems whose main function is to keep some variables of the focal or operating system within predetermined limits. They consist of four basic elements, which are themselves subsystems:

1. A control object, or the variable to be controlled
2. A detector, or scanning subsystem
3. A comparator
4. An effector, or action-taking subsystem

These four basic subsystems of the control system, along with their functional interrelationships and their relationship with the operating system, are depicted in Fig. 3. The broken lines enclose the area of the system under consideration, here the control system of the organization. It too has its inputs, processes, and outputs with the outputs of one system or subsystem serving as the serial inputs of another. In Fig. 3 a time element has been associated with the outputs. The various points in time are indicated by $t_0$ through $t_8$. If the system depicted were a production system, it would be evident from the figure that some of these timed outputs have exceeded the upper control limit. These control objects are fed into the detector, then to the comparator, and finally to the effector before being fed back into the overall system. What one sees at the bottom portion of the figure is a blown-up view of the feedback control element of a system or the implicit controller that is activated by the existence of an error, that is, a difference between the standard and the actual output of the system.

### 1. Control Object

A control object is the variable of the system's behavior chosen for monitoring and control. The



The control system keeps the systems output within certain predetermined limits via a self-regulating mechanism consisting of a Detector which constantly monitors the changes in output over time, a Comparator which compares the sensed output against a set of standards, and an Effector which decides whether deviations (i.e., the output of the comparator) warrant corrective action to be taken against the operating system's input function.

**Figure 3** Major elements of a control system.

choice of control object is the most important consideration when studying and designing a control system. Variations in the states of the control object (i.e., its behavior) become the stimuli that trigger the functioning of the system. The control system keeps the system's output within certain predetermined limits via a self-regulating mechanism consisting of a *detector* that constantly monitors the changes in output over time. The output of the detector is sent to a *comparator,* which compares the sensed output against a set of standards. The *effector* decides whether deviations (i.e., the output of the comparator) warrant corrective action to be taken against the operating system's input function. Without these variations, the system has no reason for existence. Because, in reality, we never have a perfect match between desired and actual outcome, variations will always exist; ergo, the need for control.

From the foregoing it should be clear that great care and much thought ought to be given to "what must be controlled." Let it suffice to point out the rather obvious observation that the control object must be chosen from the system's output variables. Well-balanced quantitative and qualitative attributes of the system's output should provide the best choice of control variables. Focusing on controlling system output does not necessarily imply an *ex post facto* account of system behavior. Feedback control systems can just as easily function as anticipatory mechanisms rather than *ex post facto* corrective devices.

### 2. Detector

The function of the detector or scanning system will be explained in greater detail in the next section. It will suffice to say here that scanning systems feed on information. In other words, the detector operates on the principle of selective acquisition, evaluation, and transmission of information. As such, a detector system is another name for a management information system (MIS). Frequency, capacity, efficiency, accuracy, and cost of detector devices are some of the important aspects with which an administrator must reckon.

### 3. Comparator

The output of the scanning system constitutes the energizing input of the comparator. Its function is to compare the magnitude of the control object against the predetermined standard or norm. The results of this comparison are then tabulated in a chronological and ascending or descending order of magnitude of the difference between actual performance and the standard. This protocol of deviations becomes the input to the activating system.

Note that if there are significant differences between the output and the goal, the system is said to be "out of control." This could mean that the goal formulated is unrealistic, unsuitable for the system's capabilities. Either the goal itself must be changed, or the design characteristics of the system must be altered. For example, if the production goal cannot be met, the goal must be altered or, if kept, either more people must be put on the production line or more equipment employed.

### 4. Effector

The effector is a true decision maker. It evaluates alternative courses of corrective action in light of the significance of the deviations transmitted by the comparator. On the basis of this comparison, the system's output is classified as being in control or out of control. Once the status of the system's output is determined to be out of control, then the benefits of bringing it under control are compared with the estimated cost of implementing the proposed corrective action(s).

These corrective measures might take the form of examining the accuracy of the detector and of the comparator, the feasibility of the goal being pursued, or the optimal combination of the inputs of the focal system, that is, the efficiency of the *process* of the operating system. In other words, the output of the activating system can be a corrective action that is aimed at investigating the controllability of the operating system or the controllability of the controller itself.

The ultimate results of the operation of the detector–comparator–effector process are always negative feedback, i.e., a transmission of a message to the input portion of the system that will inform it to do the opposite of what it has been doing. The end result of the efficiency of the implicit contoller is error minimization, i.e., its function is to minimize the difference between the goal and the actual performance. The system's ability to do so will depend on its ability to achieve a stable state

## V. STABILITY AND INSTABILITY OF CYBERNETIC SYSTEMS

A cybernetic system can take either of two states: (1) It can be stable or (2) it can be unstable. Both states are necessary for system survival. Although stability is the ultimate long-run goal of the system, short-run instability is necessary for system adaptation and learning. The system, in other words, pursues a long-run stability via short-run changes in its behavior manifested in its output's deviations from a standard. It is this continuous change of states (a churning) that guarantees the long-term ultrastability of a system.

Let us briefly explore the nature of stability and instability as well as some of the reasons for instability. In general terms, stability is defined as the tendency of a system to return to its original position after a disturbance is removed. In our systems nomenclature, stability is the state of the system's control object which exhibits at time $t_1$ a return to the initial state $t_0$, after an input disturbance has been removed. Were the system's control object not able to return to or recover the initial state, then the system's behavior would exhibit instability. The input disturbance may be initiated by the feedback loop, or it may be direct input from the system's environment. The particular behavior pattern that the system will exhibit is dependent on the quality of the feedback control system (detector, comparator, and effector) in terms of sensitivity and accuracy of the detector and comparator as well as the time required to transmit the error message from the detector to the effector. Oversensitive and very swift feedback control systems may contribute as much to instability as do inert and sluggish ones.

Time delay is the most important factor for instability of social systems such as business enterprises and governments. Although the application of information technology such as MIS and electronic data processing (EDP) has made considerable progress toward accelerating the transmission of information from the detector to the effector, as well as expediting the comparison and evaluation of information inside the comparator, still, the impact of the corrective action on the control object's behavior is felt after a considerable time lag. The object of the implicit controller is to minimize and/or eliminate the information float that causes this time lag.

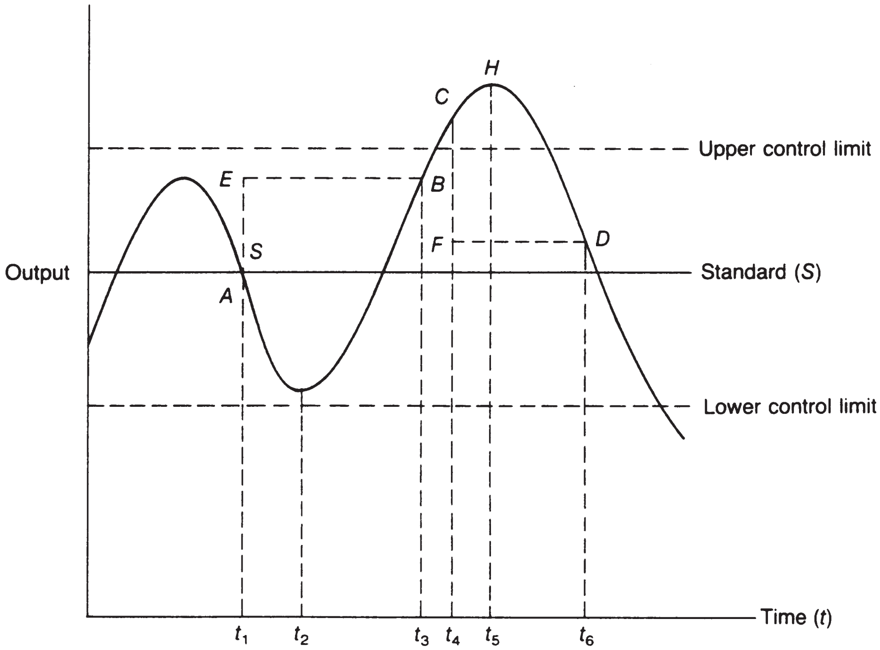
Continuous oscillations of the kind exhibited in Fig. 4 are the result of two characteristics of feedback systems: (1) The time delays in response at some frequency add up to half a period of oscillation and (2) the feedback effect is sufficiently large at this frequency.

Figure 4 demonstrates the behavior of the system's output, which is controlled by a feedback system characterized by a one-half-cycle time delay. When a time delay of that magnitude exists, the impact of the corrective action designed to counteract the deviation comes at a time when this deviation is of a considerably different magnitude, although it has the same direction. This causes the system to overcorrect. In Fig. 4 a deviation of the magnitude equal to *SA* is detected at time $t_1$. At time $t_2$, new inputs are added to bring the output back to the standard (*S*). The impact of this corrective action on the system's output is not felt until $t_3$. By that time the actual system's output is at point *C* (i.e., after a time lag equal to $t_4 - t_3$. The detector senses this new deviation and initiates new corrective action. Because of the one-half-cycle time lag, the actual system's output has oscillated above the upper limits at point *H*. New corrective action initiated, aimed at bringing the output back to the standard, will be felt at time $t_4$.

This basic principle of time delay and its impact on the system's control behavior is illustrated very clearly in Tustin's diagrams, which appear in Fig. 5. It might seem from the above brief description of the function of the basic elements of the control system that the task is a formidable one when measured in terms of cost or time. In conventional control systems, this might indeed be the case. In cybernetic control systems, however, this is definitely not true. The reason, of course, is that this task is performed as part of the normal operation of the system and requires no extra effort; i.e., it is built-in or is implicit in the system's design.

## VI. SOME PRINCIPLES OF IMPLICIT CONTROLS OR HOMEOSTATS

The basic tenets of cybernetics could be summarized in a few principles. The principles governing cybernetic systems are universal and simple. These principles as formulated by Beer are discussed next.



*A:* *Point where direction of output is recognized.*
*B:* *Corrective input is added.*
*C:* *Error is noted—directive to remove resources.*
*D:* *Resources are removed.*
*EB and FD:* *Information time lag.*

**Figure 4** Control object's behavior. [Adapted from Johnson, R. A., Kast, F. E., and Rosenzweig J. E (1967). *The Theory and Management of Systems,* 2nd ed. New York: McGraw Hill, 89.]

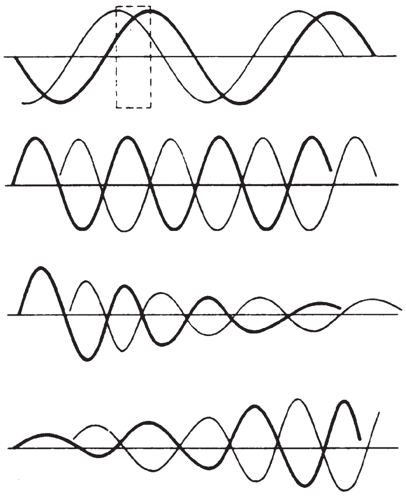


**Figure 5** Oscillations in feedback systems. Oscillation is inherent in all feedback systems. The drawing at top shows that when a regular oscillation is introduced into the input of a system (lighter line), it is followed somewhat later by a corresponding variation in the output of the system. The dotted rectangle indicates the lag that will prevail between equivalent phases of the input and the output curves. In the three drawings below, the input is assumed to be a feedback from the output. The first of the three shows a state of stable oscillation, which results when the feedback signal (thinner line) is opposite in phase to the disturbance of a system and calls for corrective action equal in amplitude. The oscillation is damped and may be made to disappear when, as in the next drawing, the feedback is less than the output. Unstable oscillation is caused by a feedback signal that induces corrective action greater than the error and thus amplifies the original disturbance. [Adapted from Tustin, Arnold (1955). Feedback. In a *Scientific American* book, *Automatic Control.* New York: Simon & Schuster, 20–21.]

## A. Control Principle I

Implicit controllers depend for their success on two vital tricks. The first is the *continuous and automatic comparison* of some behavioral characteristic of the system against a standard. The second is the *continuous and automatic feedback* of corrective action.

Thus, according to control principle 1, implicit controllers are engaged in both detector and comparison activities, as well as in corrective action. This is, of course, common to all control systems. However, what is unique in the case of implicit controllers is the prerequisite that these functions be *continuous and automatic.* That is to say, detecting, comparing, and correcting activities are not initiated periodically, nor are they imposed on the control system from outside; rather they are executed from within in a perpetual manner. The entire control function is built in or embedded into the operating system.

Implicit controllers can be found both in machines and in humans. It is possible to program machines to easily self-correct. Many functions of the human body are self-correcting. The maintenance of body temperature, blood count, the pH of electrolytes, body-fluid retention, etc., are all controlled continuously and automatically. Social organizations, on the other hand, are typically neither able nor willing because of costs to have continuous and automatic control.

## B. Control Principle II

In implicit governors, *control is synonymous with communication.* Control is achieved as a result of transmission of information. Thus, to be in control is to communicate. Or, in Norbert Wiener's original words, "Control . . . is nothing but the sending of messages which effectively change the behavior of the recipient." This is indeed the most basic and universal principle of cybernetics. The realization that control and communication are two sides of the same coin motivated Wiener to use them as the subtitle of his classic pioneering work *Cybernetics.* This is where one reads: *to control is to communicate and vice versa!*

It is evident from the above principles of control (I and II) that the system whose behavior is subject to this type of control becomes literally a slave to its own purpose. Because every deviation from standard behavior is autonomously and automatically communicated (through the sequential activities of the detector, comparator, and effector), the more frequently out-of-control situations occur, the more frequently communication takes place and consequently the more corrective action is taken. It is for this reason that implicit controllers are also referred to in the literature as *servomechanisms* (*servo* means slave) or *homeostats* (same state). These systems become slaves to their own purpose or goal, which is the minimization of the error (i.e., the difference between the goal and the actual performance).

This observation allows us to formulate another basic principle of cybernetic control, originally conceived by S. Beer.

## C. Control Principle III

In implicit controllers, variables are brought back into control *in the act of* and *by the act of* going out of control. This principle follows directly from our explanation of the basic structure of the control system. Recall that what triggers the detector subsystem is the existence and magnitude of the deviation between the goal and actual performance, i.e., the actual output of the operating or focal system. In addition, the system must be autonomous; i.e., must have the authority to inform the focal system regarding the magnitude of the deviation without the intervention of another decision-making authority. It follows that the more frequently deviations occur, the more frequent will be the communication between the detector and the comparator. In addition, the more frequent and more substantial the magnitude of the deviation, the more likely it is that corrective action will be initiated and executed.

From the foregoing discussion of the basic principles of cybernetic control, the following question is inescapable: Given the unique nature of a cybernetic system, what kinds of demands do these control principles impose on goal-directed systems? The most important demand facing the system is that it be an adaptive learning system. In other words, the function of the implicit controller demands that the operating system eventually learn that being in control is as necessary a condition for its survival as its growth capabilities. In addition, cybernetics teaches us that for systems to be in control they must be capable and free to go out of control. Thus, systems that claim to be always in control must not be free or must be lying or both.

## VII. EPILOGUE

For most people the 20th century was clearly the age of the material world. The intellectual accomplishments of the previous eras were swiftly converted into robust scientific theories that were in turn converted into great technologies. The industrial revolution, which started in Great Britain in late 18th century, reached its zenith in the last quarter of the 20th century. The main object of scientific and technological activities was the manipulation of disorganized mass and its conversion into organized objects.

Humanity experienced tremendous riches accompanied by an increase in leisure time. Humans were able to live comfortably by working less and less. Machines did all difficult tasks while humans enjoyed managing the machines. By the end of the 20th century humans lived easier, safer, better, and longer. Scientists taught engineers how to deal with the smallest object—the atom—and engineers designed machines that made atoms do what humans wanted. The end result has been an unprecedented economic wealth creation.

The 21st century is ushering in the so-called "New Economy." This economy is essentially a process of creating wealth not by converting one kind of matter into another, but by manipulating data and creating information. The myriad of wealth-creating systems is nothing else but communication devices that do not directly handle matter, but instead facilitate the movements of bits of information. Human effort is now focusing on the third characteristic of nature: information. Information tells the other two, matter and energy, what form to take. The New Economy operates not in places but in spaces. The totality of these spaces is known as *Cyberspace*. Trillions of bits of information travel in this space at a speed approaching the speed of light. Although Alan Turing's (1912–1954) dream to create a computing machine that could operate at the speed of thought has not yet become reality, Bill Gates's "Business @ the Speed of Thought" proposal seems to look more and more real. Indeed, today there are more people in the developed—and increasingly also in the developing—world making a living pushing bytes than there are those who push atoms. Cyberspace has become the new market. Cybernetics is the science that explains the behavior of systems that operate in cyberspace.

As in the middle of the 20th century, so it is in the beginning of the 21st century. Wiener's issue of man and machine is gaining momentum. Unlike in the 20th century, this time human imagination regarding the symbiosis between the animal and machine has become the subject of serious academic research and experimentation. The January 2000 issue of *Wired,* which carries the special focus of "Augment or Bust," features Cyborg 1.0, Kevin Warwick's outline of his plan to become one with his computer. Warwick and his colleagues in the Department of Cybernetics at Reading University in the United Kingdom have been working since 1988 on various cybernetic projects. In August 1998, Warwick had a silicon chip implanted in his arm allowing the computer to monitor his movements through the halls and offices. The implant communicates via radio waves with a network of antennas, which in turn transmit the signals to a computer programmed to respond to his actions. In Warwick's words, "The aim of this experiment was to determine whether information could be transmitted to and from an implant. Not only did we succeed, but the trial demonstrated how the principles behind cybernetics could perform in real-life applications."

## ACKNOWLEDGMENTS

This article is dedicated to the Schoderbek brothers, Peter, my doctoral advisor, coauthor, friend, and

lifelong mentor, and to Father Charles, friend, coauthor, and great mentor to both Peter and myself.

## SEE ALSO THE FOLLOWING ARTICLES

Automata Theory • Data, Information, and Knowledge • Decision Theory • Future of Information Systems • Game Theory • Information Theory • Systems Science • Uncertainty

## BIBLIOGRAPHY

Ashby, W. R. (1960). *Design for a brain*. London: Chapman and Hall.

Ashby, W. R. (1963). *Introduction to cybernetics.* New York: John Wiley & Sons.

Beer, S. (1964). *Cybernetics and management.* New York: John Wiley & Sons.

Beer, S. (1972). *Brain of the firm: A development in management cybernetics.* New York: McGraw-Hill.

Buckley, W. (Ed.) (1968). *Modern system research for the behavioral scientist: A sourcebook.* Chicago: Aldine Publishing Co.

Campbell, J. (1982). *Grammatical man: Information, entropy, language, and life.* New York: Simon and Schuster.

De La Metrie, J. O. (1961). *Man a machine.* La Salle, IL: Open Court Publishing Co.

Dechert, C. R. (ed.) (1966). *The social impact of cybernetics.* New York: Simon and Schuster.

Foerster, H. von (Ed.) (1953), *Cybernetics.* New York: Josiah Macy.

Kefalas, A. G. (1977). Organizational communication: A systems approach. *Readings in interpersonal communication,* R. C. Huseman, C. W. Logue, and D. L. Freshley (Eds.). Boston: Holbrook Press.

Klir, G., and Valachi, M. (1967). *Cybernetic modeling.* New York: Van Nostrand.

MacKay, D. (1969). *Information, mechanism and meaning.* Cambridge, MA: The MIT Press.

Maltz, M. (1970). *Psycho-cybernetics and self-fulfillment.* New York: Bantam Books.

Pattee, H. (1973). *Hierarchy theory: The challenge of complex systems.* New York: George Braziller.

Porter, A. (1969). *Cybernetics simplified.* New York: Barnes & Noble.

Schoderbek, P., Kefalas, A. G., and Schoderbek, C. G. (1990). *Management systems: Conceptual considerations,* 4th ed. Homewood, IL: Irwin.

Scientific American (1966). *Information: A comprehensive review of the extraordinary technology.* San Francisco: W. H. Freeman and Co.

Shannon, E. C., and Weaver, W. (1964). *The mathematical theory of communication.* Urbana: University of Illinois Press.

Simon, H. A. (1970). *The science of the artificial.* Cambridge, MA: The MIT Press.

Tustin, A. (1955). Feedback. *Scientific American: automatic control.* New York: Simon & Schuster.

Wiener, N. (1948). *Cybernetics, or control and communication in the animal and machine.* New York: John Wiley & Sons.

Wiener, N. (1954). *The human use of human beings: Cybernetics and society.* Garden City, NY: Doubleday-Anchor.

# Database Administration

**Ming Wang**
*California State University, Los Angeles*

## GLOSSARY

**database administrator (DBA)** DBA refers to the person who is a database administrator and is responsible for the control of the centralized and shared database.

**database recovery** The process restores the database to a correct state after a failure. It protects the database from inconsistencies and data loss.

**log file** A special file maintained by a DBMS contains information about all updates to a database.

**transaction** A basic logical unit consists of an action or series of actions, carried out by a single user or application program, which accesses or changes the contents of the database. Transactions can terminate successfully (commit) or unsuccessfully (abort). Aborted transactions must be rolled back to the database original state.

**DATABASE ADMINISTRATION** is a critical activity for any organization. It involves logical and physical database designs as well as technical issues like authorization, security enforcement, monitoring, performance tuning, backup and recovery.

The database administrator (DBA) is the person responsible for the design, control, and administration of the database. The DBA must manage the information system (IS) as the database is analyzed, designed, and implemented. The person also interacts with and provides support for end users. The DBA must have a broad technical background with a sound understanding of hardware architectures and database life cycles. The technical aspects of the DBA's job are as follows:

1. Database management system (DBMS) selection and configuration
2. Database development management
3. Database administration routines
4. Monitoring and tuning
5. Backup and recovery

## I. DATABASE MANAGEMENT SYSTEM (DBMS) SELECTION AND INSTALLATION

One of the DBA's first and most important responsibilities is the selection of the DBMS. The selected DBMS must have features needed by the organization and be properly interfaced with existing software and hardware in the organization. In today's Internet environment, the DBA must also work with web server administrators for the Internet connectivity.

The first step is to determine a company's needs. To match DBMS capacity to the organization's needs, the DBA should check the following features of a DBMS before he makes a decision: type of data model, storage capacity, application development tools, security, Internet connectivity, integrity, performance, concurrency control, backup and recovery procedures, database administration tools, portability and standards, hardware/ software configuration, data dictionary, vendor training and support, available third-party tools, and software acquisition cost. The DBA must supervise the installation of all software and hardware and have a thorough understanding of the configuration, startup procedures, and installation procedures. The installation procedures include details such as the location of backup and transaction log files, network configuration information, and physical storage details.

## II. DATABASE DEVELOPMENT MANAGEMENT

The DBA should be involved in every phase of database development, from database planning through analysis, design and implementation stages, testing, and modification. Strong design and data modeling skills are essential for the DBA.

### A. Conceptual and Logical Design

The DBA schedules and coordinates the database designers for data design and modeling. He determines the standards and procedures of the development and ensures that the database modeling and design activities are performed within this framework. The result of testing database and applications must meet the predefined standards.

The entity-relationship (ER) diagram is popularly used for conceptual design. The ER diagram illustrates the relationships that have been identified among the entities. ER diagrams are critical for providing an understanding of the global view of the database. They also help to ensure consistency in definitions across the enterprise, to identify the application interface, and to make up applications.

The conceptual design is further refined and mapped onto a logical data model. The logical data model can be a relational data model, hierarchical data model, network data model, or object-oriented data model. Mapping is the process of applying a set of technical rules to convert the conceptual design to a logical design. Mapping from an ER diagram to a relational database will convert an entity to a relation with a primary key and its associated attributes. Conceptual design and logical database design are iterative processes. They are critical to the overall success of the system.

### B. Application Design

Application design proceeds in parallel with conceptual design in a DBMS-independent way. When a database system is being designed, the designers should be aware of the transactions/applications that will run on the database. An important part of database design is to specify the functional characteristics of these transactions early in the design process. This ensures that the database will include all the information required by these transactions.

The DBA provides database transaction design and quality control services to the application programmers. Such support services include reviewing the database application design to ensure that transactions are correct, efficient, and compliant with database integrity and standards.

### C. Physical Design

Database administration is typically responsible for physical database design and much of database implementation. Physical design is the process of choosing specific structures and access paths for database files to achieve good performance for the various database applications. Each DBMS provides a variety of options for file organization and access paths. These include various types of indexing and clustering of related records on disk blocks. Once a specific DBMS is selected, the physical design process is restricted to choosing the most appropriate structure for the database files from the options offered by that DBMS. One of the advantages of relational database is that users are able to access relations and rows without specifying where and how the rows are stored. The internal storage representation for relations should be transparent to users in a relational database.

### D. Relational Database Implementation for a Target DBMS

#### 1. Creating Tables and Loading Data

After the logical and physical designs are completed, the DBA can implement the database systems with the selected DBMS. Data definition language (DDL) and storage definition language (SDL) are used to create database tables and data storage files. The database can then be loaded with the data. If data is to be converted from earlier file processing systems, conversion routines may be needed to reformat the data for loading into the new database.

#### 2. Defining Integrity Constraints

The constraints in the database help to ensure the referential integrity of the data in the database. They provide assurance that all of the references within the database are valid and all constraints have been met. The reason to define constraints is to make the DBMS do most of the work in maintaining integrity of the database. The more constraints you add to a table definition, the less work you have to do in applications to maintain the data. On the other hand, the more constraints there are in the table, the longer it takes to update the data. The two most important constraints specified in a relational database are the primary key and foreign key constraints. They enforce database integrity.

The primary key of the table is the column or a group of columns that makes every row in that table unique. A foreign key constraint is used to specify the nature of the relationship between tables. A foreign key from one table references a primary key that has been previously defined elsewhere in the database. For example, the Department table has a primary key of `DeptNo.` The Employee table contains `DeptNo` column links to the one in the Department table. By specifying `Employee.DeptNo` as a foreign key to `Dept.DeptNo`, you guarantee that no `DeptNo` values can be entered into the Employee table unless those values already exist in the Department table.

The design of other constraints is dependent on the choice of the DBMS. Some DBMS's provide more facilities than others for defining other constraints. Oracle provides three additional constraints—Not Null, Check, and Unique.

1. The Null constraint is a column constraint that is used to allow or disallow the absence of a value in a specific column. The Not Null constraint prevents the entry of NULL data for a column on which the NOT NULL constraint is defined.
2. The Check constraint is a column constraint that is used to ensure that only specific values will be allowed for storage in a column. Check constraints verify data in a column against a set of constants defined to be valid data.
3. The Unique constraint is a column constraint used to ensure unique values in the column. It prevents duplicate values from appearing in a column for two or more rows.

### 3. Creating Indexes

An index is an object used to speed up the retrieval of rows by using a pointer. It reduces disk I/O by using the path access method to locate data quickly. The DBA can create indexes to support database performance and improve on database applications. The key attribute used to retrieve data frequently requires an index. The following guidelines will indicate when the DBA needs to create an index or not create an index.

Create an index on:

1. The primary key unless some DBMS are automatically created
2. Any attribute that is heavily used as a secondary key
3. A foreign key if it is frequently accessed
4. An attribute that contains a wide range of values
5. An attribute that contains a large number of null values
6. A big table where most queries are expected to retrieve less than 2–4% of the rows

Don't create indexes on:

1. Small relations except for its primary key; it may be more efficient to search the whole relation than to store an additional index structure
2. An attribute or relation that is frequently updated
3. An attribute if the query will retrieve a significant proportion of the rows in the relation
4. An attribute that consists of long character strings
5. An attribute that is not often used as a condition in the query

### 4. Creating Views

A view is a virtual relation that does not actually exist in the database but is produced upon the request. Views are used to display data from one or more tables that may be inappropriate or too complex for users to access. The content of a view is defined as a query on one or more base relations. Views are dynamic, meaning that changes made to base relations that affect view attributes are immediately reflected in the view. Therefore a view is a dynamic result of relational operations on the base relations. Advantages of views are as follows:

1. Making complex queries easy
2. Restricting direct access to data in the database
3. Allowing data independence
4. Presenting different views of the same data

Oracle's data dictionary is composed of views. It stores all data about the Oracle database in tables, but disallows direct access to the tables instead of providing views through which the user can only select data. Normally the views are created using SQL statements.

## E. Testing Databases and Applications

Before an application comes on-line, the DBA must test and evaluate the database and all applications.

Testing usually starts by loading the tested database. Such a database contains test data for the applications. The purpose is to check the data definition and integrity rules of the database and application programs. Application programmers are responsible for program unit testing. DBA and system administrators are responsible for system integration testing. End users are involved with functional testing.

## III. DATABASE ADMINISTRATION ROUTINES

### A. Database Security and Authorization

The DBA is the central authority for managing the authorizations for access to the data in the database. The DBA's responsibilities include granting privileges to users who need to use the system. The DBA has a super account in the DBMS, called a system account, which provides powerful capabilities not available to other users. The DBA must specify the account structure for individual accounts, users, or user groups. This should include the ownership of all objects in the application and the clearly defined privileges for that user. A privilege is a right to execute a particular type of SQL statement or a right to access another user's object.

#### 1. User Authentication

Authentication is a mechanism that determines whether a user is who he claims to be. Whenever a person or a group of persons needs to access a database system, the DBA will create a new account and password for the user. When the user logs in to the DBMS, the DBMS checks that account number and password. Application programs are also considered as users and required to supply passwords.

Some new DBMSs have password expiration and account locking features. Password expiration means that the DBA can set a password to have a lifetime in the DBMS. Once a time period passes, the user must change his or her password or be unable to access the database. Account locking will lock an account when users attempt to log into the database unsuccessfully on several attempts.

The DBA keeps track of database users and their accounts and passwords in a table or a file with two fields—account name and password. The password is encrypted. The DBA cannot read the user's password, but the DBA can reassign the user's password by overwriting the older password. The table is maintained by the DBMS. Whenever a new account is created, a new record is inserted into a table. When account is canceled, the corresponding record must be deleted from the table.

If there are many users and privileges governing database usage, using roles can improve the management of granting privileges to users. A role is a named set of privileges that can be given to users and other roles. A DBA, or any other user with the `CREATE ROLE` privilege, can create roles, assign various privileges to roles, and grant roles to users or to other roles. This is especially true when groups of users require the same privileges. In Oracle, a role can be created with a statement as follows:

```
CREATE ROLE Student;
```

After you create a role, you can assign privileges to it

```
GRANT SELECT, INSERT, UPDATE, DELETE
on Employee, Department to Student;
```

#### 2. Object Level Granting

Anything created in the database such as a table, column, view, and procedure is called an object. Privilege at the object level specifies for each user the individual tables on which each type of command can be applied. Some privileges also refer to individual columns.

Each object is under the control of an owner account, which is the account that was used when the relation was created in the first place. The account owner controls the granting and revoking of object privileges. The owner can pass privileges on any of the owned tables to other users by granting privileges to their accounts. The DBA does not own all the objects, but the DBA can assign an owner to a whole database schema by creating the schema and associating an authorization identifier with that schema.

#### 3. Account Level Granting

The privileges at the account include the following: create privilege, alter privilege, select, insert, delete, and update privileges. Account level privilege statements are not defined as part of SQL as standard. Instead, they are defined by the individual DBMS. In Oracle, an account owner specifies the particular privileges on the objects in his or her account as follows:

```
GRANT INSERT, DELETE ON EMPLOYEE, DE-
PARTMENT TO User1;
```

Whenever the owner of an object grants a privilege on the object to another account, the privilege can be given to that account with or without grant option. If the grant Option is given, the other account can also grant that privilege on the object to other accounts. This is called propagation of privileges.

```
GRANT INSERT, DELETE ON EMPLOYEE, DE-
PARTMENT TO User1
WITH GRANT OPTION;
```

The clause WITH GRANT OPTION means the grantee can propagate the privilege to User2's account by using GRANT.

## B. Auditing

Database auditing means to monitor database activity to uncover suspicious or inappropriate use against the database. An enterprise database system must be able to keep track of all operations on the database that are applied by a certain user during each log-in session, which consists of the sequence of database interactions that a user performs from the time of logging in to the time of logging off. When a user logs in, the DBMS can record the user's account name and the client machine from which the user logged in. All the operations from that client machine are ascribed to the user's account until the user logs off. It is very important to keep track of updated operations on the database.

Database auditing consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period. When an illegal or unauthorized operation is found, the DBA can determine the account number used to perform this operation. Database audits are particularly important for sensitive database that are updated by many transactions and users, such as an airline database that is updated by many sales representatives. A database log that is used mainly for security purposes is called an audit trail. In Oracle, the DBA can set audit to TRUE for AUDIT_TRAIL in order to audit the privileged operations.

A database audit is performed if any tampering with the database is suspected. The database audit generates a lot of information about database access. If the DBA tried to audit everything, the most important information might be missed. A database audit is most effective when the DBA knows what he is looking for. The best way is to set up a clear goal, and then set audit to monitor these aspects of database. Auditing the database requires a good deal of additional storage space for storing the audit data generated.

## C. Concurrency Control Techniques

Concurrency control is the process of simultaneously managing operations on the database so that they do not interfere with each other. Concurrency control is needed when multiple users are allowed to access or to update the same data in the same database simultaneously. The transaction is a central concept associating with concurrency control techniques. Transactions can terminate successfully (commit) or unsuccessfully (abort). Aborted transactions must be rolled back to the original database state. There are two basic concurrency control techniques that allow transactions to execute safely in a parallel way: locking and stamping.

### 1. Locking

Locking is a procedure used to control concurrent access to data. When one transaction is accessing the database, a lock may deny access to other transactions to prevent an incorrect result. Locking methods are the most widely used approach to ensure serializability of concurrent transactions.

Deadlock occurs when two or more transactions are waiting to access the data locked by the other transaction. Neither transaction can continue because each is waiting for a lock it cannot obtain until the other completes. Once deadlock occurs, the DBMS has to recognize that deadlock exists and break the deadlock. Unfortunately, the only way to break deadlock once it has occurred is to abort one or more changes made by the transaction so that the lock can be released.

There are two general techniques for handling deadlock: deadlock prevention and deadlock detection. Using deadlock prevention, the DBMS looks ahead to determine if a transaction would cause deadlock and never allows deadlock to occur. Using deadlock detection, the DBMS allows deadlock to occur but recognizes occurrences of deadlock and breaks them. Many systems use the detection and recovery method, since it is easier to detect deadlock than to prevent it.

### 2. Timestamping

Timestamping methods for concurrency control are quite different from locking methods. No locks are involved, and therefore there can be no deadlock. Locking methods generally prevent conflicts by making transactions wait. With timestamp methods, there is no waiting. Transactions involved in conflict are simply rolled back and restarted.

Timestamping is a unique identifier created by the DBMS that indicates the relative starting time of a transaction. Timestamps can be generated by simply using the system clock at the time the transaction started, or by incrementing a logical counter every time a new transaction starts.

With timestamping, a transaction which attempts to read or write a data item is allowed to proceed only if the last update on that data item was carried out by an older transaction. Otherwise, the transaction requesting the read/write is restarted and given a new timestamp to prevent them from being continually aborted and restarted. Timestamping ensures that transaction conflicts are serializable.

## IV. MONITORING AND TUNING

After a database is developed and is in operation, actual use of the applications, transactions, queries, and views reveals factors and problem areas that may not have been accounted for during the initial physical design. As a matter of fact, database tuning continues for the life of the database, as long as performance problems are identified.

Most DBMSs include a monitoring utility to collect performance statistics, which are kept in a system catalog or data dictionary for analysis. These include statistics on storage, the number of times a particular query or transaction is executed in an interval time, I/O performance of files, locking or logging related statistics, counts of file pages, and frequency of index usage.

Identified performance problems can be tuned. Based on the statistics, some queries or transactions may be rewritten for better performance. We may invert to logical design, normalize or denormalize tables, and make adjustments to the database. The dividing line between physical design and tuning is very thin. As database system requirements change, it often becomes necessary to add or remove existing tables, to change primary access methods by dropping old indexes and constructing new ones, to set appropriate physical DBMS parameters, to change configurations of devices, and even to change operating system parameters.

The DBA is responsible for guaranteeing services and for ensuring the reliability of the system. The goals of tuning are as follows:

1. To make applications run faster
2. To lower the response time of transactions
3. To improve the overall throughput of transactions

### A. Tuning Database Design

If a given database does not meet the expected objectives, we may revert to the logical database design, and make adjustments to the logical schema to a new set of physical tables and indexes. The ultimate goal of normalization is to assign the logically related attributes into tables, to minimize redundancy, and to avoid update anomalies. The trade-off for this is to generate extra processing overhead in the database. In this case, the DBA should consider denormalization. The process of combining the normalized tables (which may be in BCNF or 4NF) into weaker forms is called denormalization (i.e., to add attributes from one table to attributes in another table in order to answer queries more efficiently).

### B. Tuning Indexes

Indexes may be tuned for the following reasons:

1. Some queries without an index may take too long to run.
2. Some indexes may not be needed.
3. Some indexes may be causing excessive overhead because the index is on an attribute that changes frequently.

To diagnose the above problem, the DBA needs to use a trace tool in the DBMS to ask the system to show how a query was executed and what operations were performed in what order. Based on the tuning analysis, he may drop, add, or rebuild some indexes.

### C. Tuning Queries

Query performance is dependent upon appropriate selection of indexes; indexes may have to be tuned after analyzing queries that give poor performance in SQL. A well-designed application may still experience performance problems if the SQL code is poorly constructed. Application design and SQL problems cause most of the performance problems in properly designed databases. The key to tuning SQL is to minimize the search path that database uses to find the data. Here are the two examples that indicate that a query needs to be tuned:

The first query shows that relevant indexes are not being used. The query output is supposed to display the employee's SSN for those who work under the manager with MGRSSN = ‘123456789’

```
SELECT SSN FROM EMPLOYEE
WHERE DNO IN (SELECT DNUMBER FROM
  DEPARTMENT)
WHERE MGRSSN = '123456789';
```

may not use the index on DNO in EMPLOYEE, whereas the following query may cause the index to be used.

```
SELECT SSN FROM EMPLOYEE
WHERE DNO = DNUMBER AND MGRSSN =
  '123456789';
```

The following query may take too many disk accesses. It has potential danger of searching all of inner employee table M for each row from the outer employee table E.

Get the SSN of employee with the highest salary in each department:

```
Select SSN from EMPLOYEE E
WHERE SALARY = SELECT MAX(SALARY)
FROM EMPLOYEE AS M
WHERE M.DNO = E.DNO;
```

To make it more efficient, it can be broken into two queries where the first query just computes the maximum salary in each department as follows:

```
SELECT MAX (SALARY) AS HIGHSALARY, DNO
  INTO TEMP
FROM EMPLOYEE
GROUP BY DNO;
SELECT SSN
FROM EMPLOYEE, TEMP
WHERE SALARY = HIGHSALARY AND
  EMPLOYEE.DNO = TEMP.DNO;
```

## D. Tuning Disk Utilization

How the database actually stores data also has an effect on the performance of queries. A segment is a logical storage that stores the data for a database object. Each segment contains one or more extents. An extent is a smallest storage unit containing a contiguous set of data blocks. If the data is fragmented into multiple extents, then resolving a query may cause the database to look in several physical locations for related rows. Fragmentation may slow performance when storing new records. If the free space in a data storage file is fragmented, then the database may have to dynamically combine neighboring free extents to create a single extent that is large enough to handle the new space requirements. If a segment is fragmented, the easiest way to compress its data into a single extent is to rebuild it with the proper storage parameters. This process can be automated via the Export/Import tools.

# V. BACKUP AND RECOVERY

## A. Database Backup

Backup is the process of periodically copying the database and log file onto storage media. It is always advisable for the DBA to make backup copies of the database and log file at regular intervals and to ensure that the copies are in a secure location. In the event of a failure, the backup copy and the details captured in the log file are used to restore the database to the latest possible consistent state. It is strongly recommended that the log file be stored on a disk separated from the main database files. This reduces the risk of both the database files and the log file being damaged at the same time. Two database backup techniques currently used are logical backup and physical backup.

### 1. Logical Backup

A logical backup of the database involves reading a set of database records from a table and writing them to a file. These records are read independently of their physical location. The Export utility performs this type of backup. To recover using the file generated from an export, the corresponding import utility is used. For example, Oracle's Export utility reads the database including the data dictionary, and writes the output to a binary file called an export dump file. You can export the full database, specific users, or specific tables. In practice, you can perform full database exports for all tables or for only those tables that have changed since the last export. Use Import to recover database objects.

### 2. Physical Backup and Recovery

Physical backup involves copying the files that form the database without their logical content. These backups are also referred to as file systems backups since they involve using operating system file backup commands. There are two different types of physical files backup: off-line backups and on-line backups.

**Off-line backups**—Off-line backups occur when the database has been shut down normally. The following files are backed up: data files, control files, on-line redo logs, and database configuration file such as the `init.ora` file in Oracle.

**On-line backups**—An on-line backup involves setting each tablespace into a backup state, then backing up its datafiles, and then restoring the table space to its normal state. You can use on-line backups for any database that is running in on-line backups mode. In this mode, the on-line redo logs are archived, creating a full log of all transactions within the database.

On-line backup procedures are very powerful for two reasons. First, they provide full point-in-time recovery. Second, they allow the database to remain open during the file system backup. Thus, even databases that cannot be shut down due to user requirements can still have file-system backup.

## B. Database Recovery

Database recovery is the process of restoring the database to a correct state in the event of a failure. The failure may be the result of a system crash due to hardware or software, a media failure, such as a head crash, or application software errors. It may also be the result of unintentional or intentional corruption or destruction of data by operators or users. Whatever the underlying cause of the failure, the DBMS must be able to recover from the failure and restore the database to a consistent state.

### 1. Restore the System Log File

To facilitate recovery, the DBMS must maintain a log file containing transaction records that identify the start/end of transactions and the before and after images of the write operations. In the case of a catastrophic system failure, the latest backup copy can be reloaded from the storage tape to the disk and the system can be restarted. It is necessary to restore the last backup copy of the database and reapply the update operations of committed transactions using a log file. This assumes that the log file has not been damaged.

### 2. Transaction Rollback

If the database has not been physically damaged but has become inconsistent, we don't need to use the backup copy of the database, but can restore the database to a consistent state using the before- and after-images held in the log file. For example, the system crashed while transactions were executed, then it is necessary to undo the changes (rollback) that caused the inconsistency to the original consistent state. Then redo the transactions to ensure that the updates they performed have executed correctly in the database.

# VI. TRENDS IN DATABASE ADMINISTRATION

## A. Changes

Database technology evolved from file processing systems. Before the database was invented, data was stored in different data files and data administration was essential in every data processing organization. Data administration is a high level function that is responsible for the overall management of data resources in an organization. With databases developed on a mainframe legacy system, the role of data administration still remains popular. Organizations that employ separate data administration functions often have mainframe-based databases that have been developed using established systems development methodology, including development of logical and physical models.

As client/server database technology develops, the blend of the two roles between data administration and database administration is becoming a reality. These organizations emphasize the capacity to build a database quickly, tuning it for maximum performance and being able to restore it to production quickly when problems develop. These databases are more likely to be departmental, client/server databases that are developed quickly using newer developmental approaches such as prototyping, which allow changes to be made quickly. There are no universally accepted data administration and database administration structures. Organizations vary widely in their approaches to data administration. As business practices change, roles are also changing within organizations. In organizations where there is no separate data administration function, the DBA also assumes the responsibilities of the data administrator.

## B. The Future

From the advent of shared databases, the role of database administration has been changing to become more specialized, evolving into distributed databases, server programming, and data warehouses. The ability to work with multiple databases, communication protocols, and operating systems will be highly valued. There will be more and more automatic tools for database monitoring and tuning. Traditional database tuning might be replaced by decision support systems able to tune systems by analyzing usage patterns.

## SEE ALSO THE FOLLOWING ARTICLES

Database Development Process • Database Machines • Database Systems • Data Warehousing and Data Marts • Network Database Systems • Network Environments, Managing

## BIBLIOGRAPHY

Caffrey, M., and Scherer, D. (2001). *Oracle DBA: Interactive workbook.* Englewood Cliffs, NJ: Prentice-Hall.

Connolly, T. M., and Begg, C. E. (2002) *Database systems: A practical approach to design, implementation, and management,* 3rd ed. Reading, MA: Addison-Wesley.

Date, C. J. (2000). *An introduction to database systems,* 7th edition. Reading, MA: Addison-Wesley.

Elmasri, R., and Navathe, S. B. (2000). *Fundamentals of database systems,* 3rd edition. Reading, MA: Addison-Wesley.

Koch, G., and Loney, K. (2000) *Oracle8i: The complete reference.* Berkeley, CA: Osborne McGraw-Hill.

Loney, K. (2000). *Oracle8i DBA handbook.* Berkeley, CA: Osborne McGraw-Hill.

Ramakrishman, R., and Gehrke, J. (2000). *Database management systems,* 2nd edition. Boston, MA: McGraw-Hill.

Riccardi, G. (2001). *Principles of database systems with internet and Java applications.* Reading, MA: Addison-Wesley.

Rob, P., and Coronel, C. (2001). *Database systems: Design, implementation, and management,* 4th edition. Cambridge, MA: Course Technology.

Theriault, M., Carmichael, R., and Viscusi, J. (2000). *Oracle DBA 101.* Berkeley, CA: Osborne McGraw-Hill.

# Database Development Process

**Ming Wang**
*California State University, Los Angeles*

**Russell K. Chan**
*Pioneer Hi-Bred International, Inc.*

## GLOSSARY

**data** Data are raw facts that are meaningful and can be stored in a database. Data can be characterized as atomic (e.g., bit, character, integer, floating point, or boolean), complex (e.g., records, sets, or arrays), or multimedia (e.g., text fonts, audio, image, or digital video).

**database** A database is a collection of related data and meta-data stored in secondary storage. Data in the database are persistent because once data are stored in the database, it can be retrieved, updated, or removed from the database only by an explicit request to the database management system.

**database administration** Database administration includes physical database design and implementation using a target DBMS, setting security and integrity constraints, monitoring system performance, tuning the database and database queries, and performing backups and system recovery.

**database design** Conceptual design is the high-level, abstract representation of reality. Logical design translates this representation into specifications that can be implemented on and processed by a computer system. Physical design determines the physical storage structures and access methods required for efficient access to the contents of a database from a secondary storage device.

**database language** A database language provides a description and implementation of a data model. It is also an interface between users and a database system. Typical database languages are database definition language (DDL), database manipulation language (DML), and database control language (DCL). DDL defines the database schema. DML defines database operations. DCL controls database privileges and security.

**database management system (DBMS)** A DBMS is a software package that is used to define, store, maintain, and provide controlled access to databases. Examples of relational DBMSs are Oracle, Access, FoxPro, SQLServer, DB2, Informix, Sybase, and Visual Base. Examples of object-oriented DBMSs are Jasmine, O2, Gemstone, and Object Store.

**database system** A database system involves a DBMS, databases, and database applications. The interactions among them make a database system work. The DBMS is the software that creates and manages databases. Database applications are programs that create and use the data in databases.

**data dictionary** The data dictionary defines the structure for each table in the system including all attribute names and characteristics. In other words, the data dictionary contains meta-data that describes what data are stored in the database.

**data model** An abstract, self-contained, and logical definition of a collection of concepts that can be used to describe the structure of a database. By structure of a database we mean the data, the relationships among data, and the constraints that apply to the data. Some data models include a set of basic operations for specifying retrievals and updates on the database.

**entity-relationship model (ER)** The entity-relationship model is a high-level conceptual data model developed by Chen in 1976 to facilitate database design.

The ER diagram is a diagrammatic technique for displaying an ER model. The ER diagram describes data as entity types, relationship types, and attributes and key attributes. Entity types are specific objects or things in the real world that are represented in the database.

**meta-data** Meta-data describes how data are stored in the database. It is closely related to the database management system software. It includes storage structures, logical database structures, and user views or subsets of conceptual schema.

**normalization** Normalization is a formal technique for analyzing relations based on their primary keys, functional dependencies, and other types of dependencies. The technique involves a series of rules that can be used to test individual relations so that a database can be normalized to a certain degree. When a requirement is not met, the relation violating the requirement must be decomposed into relations that individually meet the requirements of normalization.

**object-oriented database (OODB)** The object-oriented database model is based on the object-oriented data model defined by Kim in 1991. An OODB is a collection of persistent and sharable objects supporting object-oriented principles such as encapsulation, inheritance, and polymorphism.

**object-relational database (ORDB)** Object-relational database technology is an extension of relational database technology to support some object-oriented features such as encapsulation, inheritance, and reuse. The object-relational database management system (ORDBMS) is a hybrid of a relational database management system (RDBMS) and object-oriented database (OODBMS) technology. ORDB is still relational because the object data are stored in relations (tables). The technique used for ORDB is mapping objects to relations.

**relational database (RDB)** A relational database is based on the concept of a relational data model. The relational data model was proposed by E. F. Codd in 1970. In the relational data model, data and relationships are represented in tables that are composed of rows and columns. Each column has a unique name and data type. A data value is stored at the intersection of each row and column. A row in a table represents a relationship among a set of values. Tables are related to each other by defined referential integrity. Data retrieval from multiple tables is done through referential integrity.

**SQL** Structured query language (SQL) is the most widely used language for creating, manipulating, and querying relational databases and is supported by almost every relational DBMS. SQL is a non-procedural language. Instead of writing a program to retrieve data, you can retrieve data from a database with a single SQL statement.

**system catalog** The system catalog is one of the fundamentals of a DBMS. It is actually a system-created database whose tables store the user-created database characteristics and contents. It is a very detailed system data dictionary that contains meta-data that describes how data are stored in the database.

**THE DATABASE DEVELOPMENT PROCESS** is divided into two parts: database development and application development. Database development includes design (data structure and content) and implementation. Application development is a process including functionality analysis, transaction design, user interface design, and implementation. Database design and application design are parallel activities. During or after the conceptual design, the basic data model operations can be used to specify the high-level user operations identified during functional analysis. This also serves to confirm that the conceptual design meets all the identified functional requirements. Modifications to the conceptual design can be introduced if some functional requirements cannot be specified in the initial conceptual design. We must ensure that all the functionalities stated in the users' requirements specification are present in the application design. In addition, the physical database design phase, during which we choose the storage structures and access paths of database files, depends on the applications that will use these files. Obviously, these two activities strongly influence one another. Traditionally, the database development process has primarily focused on the first activity, whereas software design has focused on the second. This may be called data-driven versus process-driven design. It is now being recognized by database designers and software engineers that these two activities should proceed hand in hand, and the commercial design tools are increasingly combining them together. Figure 1 shows the interaction between the data and process during the database development process.

## I. DATABASE DEVELOPMENT

The database development process begins with enterprise modeling. Ideally every database development should fit within the enterprise architecture. This means data collection and conceptual design are based on the needs of the business and are then trans-

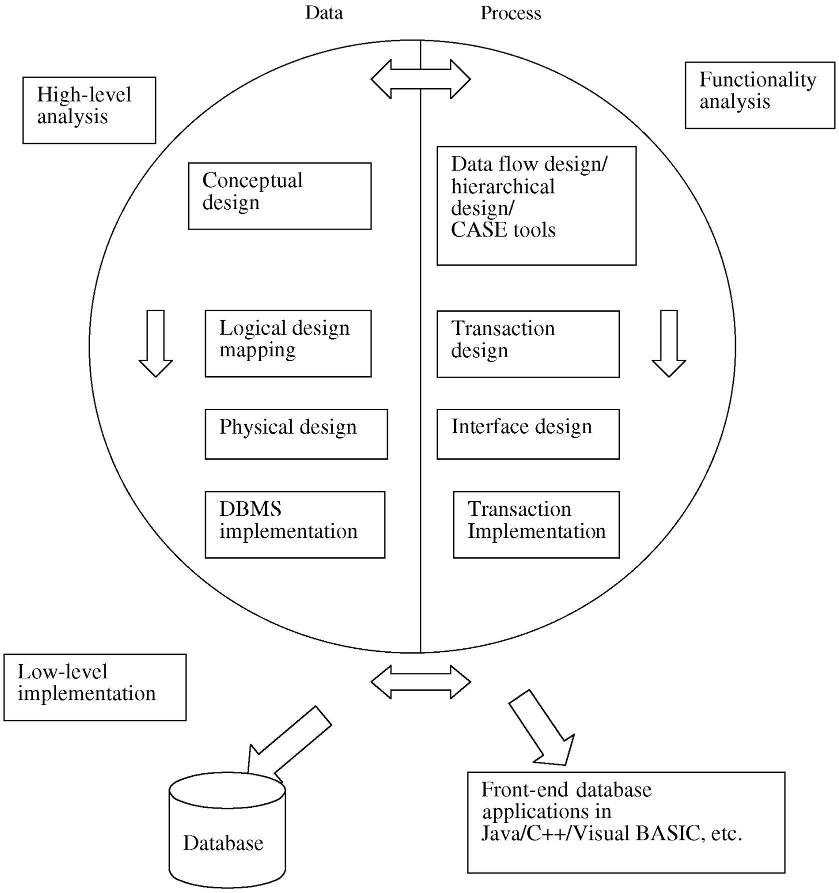


**Figure 1** Database development process diagram.

lated into logical design, physical design, and the implemented database systems. The following phases do not have to proceed strictly in sequence. In many cases we may have to modify the design from an earlier phase during a later phase.

## A. Data Collection and Analysis

Data collection and analysis is a preliminary stage of database design. The information required for database design may be gathered in the following ways:

- Interviewing prospective end-users of the database within the enterprise
- Using a questionnaire to gather information from prospective end-users
- Observing the enterprise operation
- Reviewing documentation used or generated in day-to-day work
- Analyzing and converting data from the existing system

## B. Conceptual Design

Conceptual design is the high-level, abstract representation of reality. It is a concise description of the data requirements of users and includes a detailed description of the entity types, relationships, and constraints. The conceptual design can be used as a reference to ensure that all users' data requirements are met. It enables the database designers to concentrate on specifying the properties of the data for each end-user view without being concerned with data storage details. This conceptual design is often carried out using the entity-relationship (ER) and enhanced entity-relationship (EER) diagrams.

### 1. Entity-Relationship (ER) Model

The most popular high-level data model used in database design is the entity-relationship (ER) model. The ER model uses a top-down design approach. The ER analysis begins with identification of entity types in the user's view of the enterprise and identification of relationship types between the entity types that we have identified. An entity type is a specific object or thing having an independent existence in the world. It can be either concrete or abstract. Let us consider a human resource example using the entity types Employee and Department and the relationship type, Employee Works for Department. Finally we identify attributes associated with each entity type such as (Employee ID, Last Name, and First Name) and (Department Code, Department Name, and Department Phone). Attributes are properties used to describe an entity type. A specific entity will have a value for each of its attributes; for example, a specific employee entity may have empID=987654321, empLast=Smith and empFirst=John.

#### a. Degree of Relationship

The degree of a relationship refers to the number of participating entity types in a relationship. A relationship between two entity types is called a binary type relationship. A relationship between three entity types is called a ternary type relationship. Relationship types of many degree are called *n*-ary type relationships.

Figure 2 illustrates a binary relationship using Chen's ER diagram.

#### b. Symbols

A rectangle denotes an entity type. Ovals denote attributes. A diamond indicates a relationship type with lines connecting the related entity types. Underlined attributes are keys that are associated with the entity type. Each value in the key is used to uniquely identify an instance of an entity type. Employee ID is a key identified in Employee. Department Code is a key identified in Department. An entity type may have more than one key, each of which is a candidate key. The primary key in a relational database table is chosen from one of the candidate keys.

#### c. Cardinality Ratio

The cardinality ratio defines the three types of relationships between entity types. One to many (1:M) or many to one (M:1), one to one (1:1), and many to many (M:N). The ER model uses the cardinality ratio to represent the conceptual design of a database. In Fig. 2, the cardinality ratio between the two entity types is many to one (M:1). That is, one department may have many employees and one employee must work for only one department.

#### d. Participation

A double line indicates total/mandatory participation; that is, each employee must work for one department. A single line indicates partial/optional participation; that is, each department may or may not have employees.

## 2. Enhanced Entity-Relationship (EER) Model

The enhanced entity-relationship Model (EER) supports object-oriented concepts such as inheritance. It extends the ER diagram by adding new symbols to the ER diagram. For instance, a company has some employees who are paid by hourly wage and others who are paid by salary. The hourly employees have an hourly pay rate, whereas the salaried employees have a monthly pay rate. In addition, salaried employees earn vacation and sick leave.

#### a. Superclass and Subclass

It is natural to define an entity class HourlyEmployee as a subclass of Employee. That is, an HourlyEmployee entity is an Employee and has all of its characteristics. Additionally, an HourlyEmployee has additional characteristics, such as an hourly pay rate. Similarly, salaried employees are employees who are different from hourly employees. Figure 3 is an EER diagram that depicts subclass–superclass relationship types. These relationship types are often called "is-a types" because a member of the subclass is a member of the superclass.

#### b. Symbols

The subclasses HourlyEmployee and SalaryEmployee are connected to the superclass Employee with lines connected through a circle. The d in the circle indicates disjointness, which specifies that the sub-

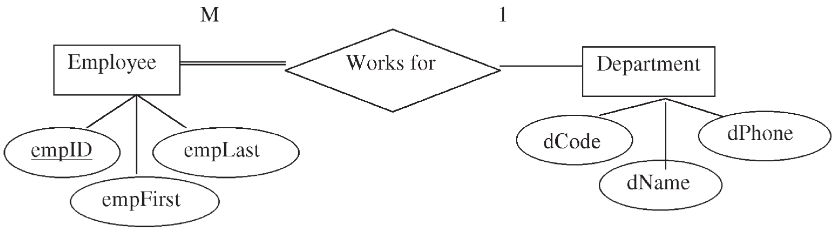


**Figure 2** Chen's ER diagram depicts the binary relationship.

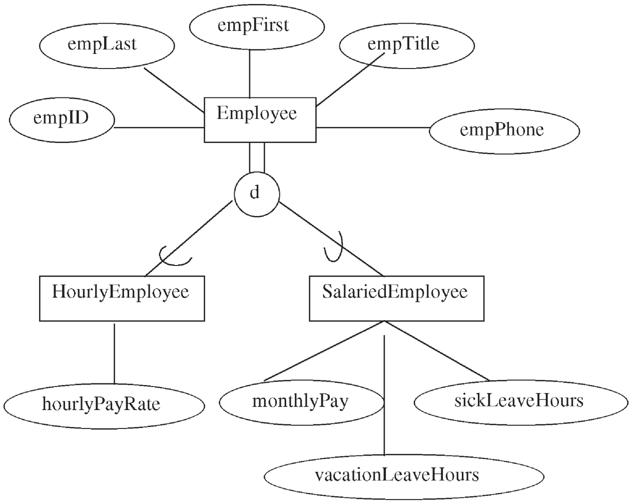


**Figure 3** EER diagram for specification.

classes of the specification must be disjoint. This means that an entity can be a member of at most one of the subclasses of the specification. An individual employee is either an hourly paid employee or salary paid employee and cannot be both. The inheritance (arch) symbols have their open sides facing the superclass. The entity class HourlyEmployee has six attributes: five inherited from Employee and one of its own. The entity class SalariedEmployee has eight attributes: five inherited from Employee and three of its own.

Class hierarchies are categorized in two ways: specification and generalization. Specification is the process of dividing a class of objects into more specialized subclasses. Generalization is the reverse process of generalizing several subclasses into a higher-level abstract superclass that includes the objects in all these classes. Specification is conceptual refinement in a top-down approach, whereas generalization is conceptual synthesis in a bottom-up approach.

## 3. Universal Modeling Language (UML) Diagram

The UML Class Diagram is a standard for object modeling in software design and has now become a new approach used for object-oriented databases and object-relational database modeling. Figure 4 describes the classes Employee and Department and the relationship between them.

In UML, multiplicities are specified in the form min..max and an asterisk(*) indicates no maximum limit on participation. Thus Fig. 4 indicates that one employee may work for one or zero department and one department must have four or more employees. UML was developed by the Object Management Group and spearheaded by Rational Software Corporation. In UML class diagrams, a class is displayed as a box (see Fig. 4) that includes three sections: the top section gives the class name, the middle section includes the attributes for individual objects of the class, and the last section includes methods that can be applied to these attributes. An entity in the ER diagram corresponds to an object in UML.

## 4. Summary of Conceptual Design

To sum up, we can conduct the conceptual design with ER, EER, or UML class diagrams in the following steps:

a. Identify entity types from each user group's view
b. Identify the important relationship types that exist between the entity types that we have identified
c. Determine the cardinality ratio and participation constraints of relationship types
d. Identify attributes associated with the entity types
e. Determine attribute domain (value range) for each attribute
f. Determine a key or keys for each entity type

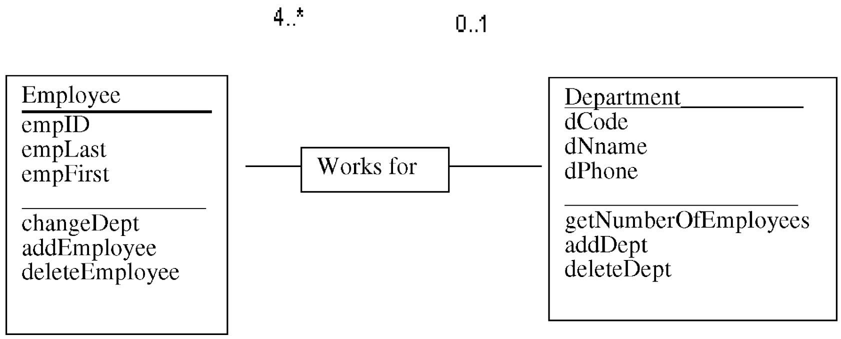


**Figure 4** UML class diagram.

g. Identify the subclasses (to specialize entity types) and the superclass (to generalize entity types)
h. Draw entity-relationship/enhanced entity-relationship/UML diagram
i. Review the conceptual design with perspective users

## C. Logical Design (Data Model Mapping)

The second phase of database design is called logical database design. The conceptual data model created from each end-user group's view in the previous phase is refined and mapped onto a logical data model, which can be a relational, network, hierarchical, or object-oriented data model. A global logical design of the database comes from the combination of local logical designs for each end-user group. The technique of normalization is used to test the correctness of a logical data model.

### 1. Mapping to a Relational Model

Mapping is the process of transforming conceptual design to logical design. Given an ER diagram describing a database, there is a standard approach to generating a relational database. An entity type is mapped to a relation in a straightforward way: each attribute of an entity type becomes an attribute of the table. Table I shows the correspondence between ER and relational models.

Consider the Department entity type in Fig. 2. A possible Department relation is mapped as:

```
Department (dCode, dName, dPhone)
```

Additional rules for mapping are as follows:

#### a. Rules for Binary Relationship Types

i. One to one—Merge the two entity types into a single relation.

**Table I** Correspondence between ER and Relational Models

| From ER model | To relational model |
|---|---|
| Entity type | Relation/table |
| Key attribute | Primary/secondary key |
| Simple attribute | Attribute |
| Composite attribute | Set of attributes |
| Relationship | Set of attributes |

ii. One to many—Map the two entity types into the two corresponding relations and add a foreign key to the Many side of the relationship corresponding to the primary key on the one side. Figure 2 has a one-to-many relationship, which can be mapped into the following two relations:

```
Employee (empID, empLast, empFirst,
   dCode)
Department (dCode, dName, dPhone)
```

iii. Many to many—Map the two entity types into two corresponding relations and create a "Relationship" relation with two foreign keys from each of the two relations as a primary key.

#### b. Rule for Ternary Relationship Types

Map the entity types into corresponding relations and create the "Relationship" relation with three foreign keys from each of the three relations as a primary key.

#### c. Rule for Weak Entity Types

Create a relation and include all the attributes of a weak entity. The relation does not have its own primary key. The primary key of the relation is the combination of the primary key of the owner(s) and the partial key of the weak entity type. The weak entity type is an entity type whose existence depends on another entity type (Owner).

#### d. Rule for Each Multivalued Attribute in a Relation

Create a new relation and use the same name as the multivalued attribute. The primary key in the new relation is the combination of the multivalued attribute and the primary key in the parent entity type. For example, department location is a multivalued attribute associated with the Department entity type since one department has more than one location. Since multivalued attributes are not allowed in a relation, we have to split the department location into another table. The primary key is the combination of deptCode and deptLocation. The new relation deptLocation is

```
DeptLocation (deptCode, deptLocation,
   deptPhone)
```

#### e. Rules for Mapping the EER Superclass/Subclass Hierarchy in Fig. 3

i. Create a relation for the superclass and for each subclass. Map each entity type Employee, HourlyEmployee, and SalaryEmployee to a distinct relation

```
Employee (empID, empLast, empFirst,
  empTitle, empPhone)
HourlyEmployee (empID, hourlyRate)
SalaryEmployee (empID, MonthlyPay,
  sickLeaveHours, vacationLeaveHours)
```

ii. Alternatively, we can create just two relations HourlyEmployee and Salaryemployee. Each includes all the attributes of its own entity types as well as all the attributes of Employee (i.e., empID, empLast, empFirst, empTitle, and empPhone).

```
HourlyEmployee (empID, empLast,
  empFirst, empTitle, empPhone,
  hourlyRate)
SalaryEmployee (empID, empLast,
  empFirst, empTitle, empPhone,
  MonthlyPay, sickLeaveHours,
  vacationLeaveHours)
```

This approach is not applicable in some situations. If we have employees who are neither hourly employees nor salaried employees, but contract employees, there is no way to store such employees in the database.

## 2. Normalization

Normalization is a process for assigning attributes to entities. Normalization involves the identification of the required attributes and their subsequent decomposition into normalized tables based on the functional dependency analysis between the attributes. Normalization ensures that the relations derived from the data model do not have data redundancy, which can cause update anomalies when implemented. Table II contains data used by a software consulting firm for its operations. The business rules are assumed as follows:

- The company manages many contracts.
- A contract has one or more employees.
- An employee may be assigned to one or more contracts.
- An employee must have one and only one job title.
- One job title may be used for many employees.
- The company charges its clients by billing the hours spent on each contract.
- The hourly billing rate is dependent on the employee's job title.

### a. Data Redundancies and Anomalies

Obviously, Table II has data redundancies. For example, each time another employee is assigned to a contract, information about the contract such as the contract number and name must be retyped. Each time Mary Fox is assigned to another contract, information about Mary Fox is retyped. The data redundancies invite data inconsistencies and yield three kinds of anomalies: update anomalies, addition anomalies, and deletion anomalies. Modifying the job class for employee Mary Fox requires at least two row alterations which could cause update anomalies if the alterations are done incorrectly. If employee Mary Fox quits, deletions must be made for every entry in which EmpID = 111762334. As such deletions are made, other vital data are lost. These are called deletion anomalies. Insertion anomalies occur when a new employee must be assigned to a contract in order to complete a row definition even if this employee may not have been assigned a contract yet. In order to reduce data redundancies and eliminate the data anomalies, we must normalize Table II.

### b. Functional Dependency Analysis

Normalization is based on functional dependency analysis. Functional dependency analysis on the table is performed as follows:

i. Mark each key attribute component on a separate line, and then write the original key on the last line:

```
ContractNo
EmpID
ContractNo, EmpID
```

**Table II** Data Table Used by a Consulting Firm

| Contract No | Contract Name | Emp ID | Emp Name | Job Title | Hourly Rate | Hours |
|---|---|---|---|---|---|---|
| 0100 | Data conversion | 234890101 | John Smith | DBA | 60.00 | 35 |
| 0100 | Data conversion | 111762334 | Mary Fox | Programmer | 50.00 | 20 |
| 0200 | System update | 111762334 | Mary Fox | Programmer | 50.00 | 15 |
| 0300 | Database design | 546778789 | Tim Wilson | DBA | 60.00 | 39 |

ii. Write the dependent attributes after each new key. The function dependency analysis for Table II is done as follows:

```
ContractNo → ContractName
EmpID → , EmpName, JobTitle
EmpID, ContractNo → ChgHours
JobCode → JobTitle, HourlyRate
```

Functional dependency analysis can also be done via a function dependency diagram as shown in Fig. 5.

### c. Normalization Forms

Normalization works through a series of stages called normal forms. The first three stages are described as first normal form (1NF), second normal form (2NF), and third normal form (3NF). From a structural point of view, a higher normalization form is better than a lower normalization form. That is to say, 2NF is better than 1NF and 3NF is better than 2NF. For most business-oriented database designs, the third normal form is as high as we need to go in the normalization process. Only very specialized applications may require fourth normal form (4NF) or fifth normal form (5NF).

***i. First normal form (1NF)*** A table is in the first normal form (1NF) if all the key attributes are defined and there are no repeating groups in the table. All attributes are dependent or partially dependent on the primary key. Table II is in 1NF. The composite primary key is ContractNo and EmpID. All attributes are dependent or partially dependent on this primary key.

***ii. Second normal form (2NF)*** A table is in 2NF if it is in 1NF and it includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key. In the conversion from 1NF to 2NF, partial dependency needs to be removed. The primary key for Table II is the combination of ContractNo and EmpID. Only ContractNo is the determinant of ContractName. Thus contract name is dependent on ContractNo. Only EmpID is the determinant of EmpName. Thus employee name is dependent on EmpID. Functional dependency analysis can be written in the following format:

```
ContractNo → ContractName
EmpID → EmpName, JobTitle, hourlyRate
EmpID, ContractNo → ChgHours
```

It is still possible for a table in 2NF to exhibit transitive dependency; that is, one or more attributes may be functionally dependent on nonkey attributes.

***iii. Third normal form (3NF)*** A table is in 3NF if it is in 2NF and it contains no transitive dependencies; that is, a condition in which an attribute (non-primary-key) is dependent on another attribute (non-primary-key) that is not part of the primary key.

```
Emp (EmpID, EmpName, JobTitle,
  HourlyRate)
```

Note the above Emp table is in 2NF, but not in 3NF because JobTitle is the determinant of HourlyRate. Functional dependency analysis on the Emp table can be written in the following format:

```
JobCode → JobTitle, HourlyRate
EmpID → EmpName, JobTitle
```

We normalized Table II from 1NF to 2NF, and then to 3NF. According to the function dependency analysis we can finally decompose Table II into four different tables. All the tables are in 3NF:

```
Contract (ContractNo, ContractName)
Emp (EmpID, EmpName, JobTitle)
WorksOn (EmpID, ContractNo, ChgHours)
Job (JobCode, JobTitle, HourlyRate)
```

### d. Denormalization

The highest level of normalization is not always desirable. We usually stop at 3NF. The reverse of normalization is denormalization, which is the process of combining two or more tables into one table. Normalization purity is difficult to reach in the real database environment. Normalized (decomposed) tables require additional processing (join) and reduce system speed. The conflict among design efficiency, information requirements, and processing speed is often resolved through compromises that include denormalization.

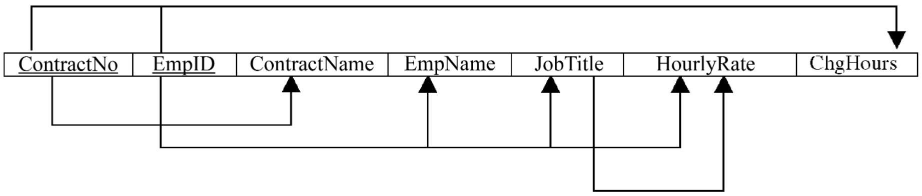


**Figure 5** Function dependency diagram.

#### e. Bottom-up Design and Normalization

Normalization can also be used as the bottom-up approach for the design of relational databases. It is far less popular than the top-down design approach such as ER and EER and is only used for very small and simple databases. An example of this normalization approach would be to create a database using data from a flat file on an existing system. Functional dependency analysis is used to group attributes into relations that represent types of entities. The process of normalizing Table II from 1NF to 3NF is an example. If we take Table II as a flat data file, we have successfully converted it into a small database through the process of normalization.

## D. Choice of DBMS

It becomes important to select an appropriate DBMS package to implement the database after logical design. One approach to choosing a DBMS is to check off DBMS features against requirements. The following features should be checked:

- Data definition—data types, integrity controls, and data dictionary
- Physical definition—file structures and indexing
- Accessibility—query languages, database host language, interface to high-level languages
- Transaction handling—concurrency control and deadlock resolution strategy
- Backup and recovery—checkpoints, logging facility
- Utilities—data upload/download facilities, tuning, and performance measuring tools
- Database security—user access controls and authorization
- Application development environment—Internet connectivity
- Vendor stability and user base
- Maximum number of concurrent users
- Operating system requirements

## E. Physical Design

The aim of physical database design is to decide how the logical database design will be implemented. For the relational database, this involves:

- Defining a set of the table structures, data types for fields, and constraints on these tables such as primary key, foreign key, unique key, not null and domain definitions to check if data are out of the range.
- Identifying the specific storage structures and access methods to retrieve data efficiently. For example, adding a secondary index to a relation.
- Designing security features for the database system including account creation, privilege granting/revocation, access protection, and security level assignment.

Physical design is DBMS-specific whereas logical design by contrast is DBMS-independent. Logical design is concerned with the what; physical database design is concerned with the how. In short, physical design is a process of implementing a database on secondary storage with a specific DBMS.

## F. Implementation

The completion of physical design allows the database administrator to begin the database implementation process:

### 1. Database Creation

All the tables, constraints, indexes, and views need to be created with the selected DBMS. Storage space and the access methods also need to be defined.

### 2. Database Loading

The newly created database contains the table structures. These table structures can be filled by typing the data into the table or by loading the data from existing databases or files. The copying procedure requires the use of special loading utilities in the DBMS. Because of foreign keys and referential integrity, data must be loaded in a specific order.

### 3. Testing

Once the data have been loaded into the database, the database administrator needs to test and fine-tune the database for performance, referential integrity, data consistency, concurrent access, and security constraints. Testing and evaluation must ensure that database security and integrity are maintained, that backup and recovery procedures are in place, and that database access and use are properly secured. If the database implementation fails to meet some criteria, several options may be considered as follows:

- Modify the physical design
- Modify the logical design

- Upgrade or change the DBMS software
- Fine-tune the DBMS configuration parameters for performance related issues

### G. CASE Tools

CASE stands for computer-aided software engineering. CASE tools can be applied to support database development. There are three types of CASE tools: upper-CASE, lower-CASE, and integrated CASE tools:

1. The upper-CASE tool supports database planning and design including data collection and analysis, data model generation and application design.
2. The lower-CASE tool supports database implementation including data conversion, report generation, application code generation, prototyping, and testing.
3. The integrated CASE tool supports all phases of database development and provides the functionality of both upper-CASE and lower-CASE in one tool.

CASE tools were created to improve the productivity and quality of database development. The advantages of using them are:

- Promoting the standards for database development for data, diagrams, documentation, and projects, making them easy to reuse and maintain
- Keeping data linked, consistent and integrated for the organization
- Reducing database development time since some CASE tools can automatically generate diagrams and application of an executable code based on the design

## II. APPLICATION DEVELOPMENT

The role of an application program is to extract and update the information in the database. Application programs are written for the end-users of the database. Most application programs provide user interfaces that allow people to interact with the database.

### A. Functionality Analysis

In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user-defined operations (or transactions) that will be applied to the database, and they include both retrievals and updates. In software design, it is common to use a data flow diagram (DFD), hierarchical input process output (HIPO), or computer-aided software engineering (CASE) tools for specifying functional requirements. Identifying the required functionality for a database application is a critical activity. Inadequate or incomplete functionality will lead to rejection or underutilization of the system.

### B. Transaction Design (DBMS-Independent)

The design of a database transaction is based on the functionality analysis and is DBMS-independent. A transaction is an action carried out by a single user or application program, which accesses or changes the content of the database. There are two kinds of transactions: retrieval transactions and update transactions. Sometimes a transaction can be a combination of the two. A transaction may be composed of several operations, such as transfer of money from one account to another account. However, from the user perspective it is only a single task. A transaction is a logical unit of work that must be either completed or aborted. No intermediate states are acceptable. The purpose of transaction design is to define and document the high-level characteristics of the transactions required on the database system. This activity should be carried out early in the design process to ensure that the logical data model is capable of supporting all the required transactions. It is important to specify the documentation of each transaction as follows:

- Data to be used by the transaction
- Function description
- Output of the transaction
- Importance to users

### C. User Interface Design

In addition to designing functionality, we have to design an appropriate user interface to the database application. The user interface is the menu that users need to follow when they use applications to retrieve or update data in the database. Most developers use the Access, ASP, JSP, or other GUI tools to generate an interface. Shneiderman's 1992 guidelines will help developers to determine if the interface design is user-friendly. Some factors are

- Meaningful title
- Comprehensible instructions
- Logical grouping and sequencing of fields
- Visually appealing layout
- Consistent use of terminology, abbreviations, and color
- Visible space and boundaries for data-entry fields
- Convenient cursor movement
- Error correction for characters and fields
- Meaningful error messages for unacceptable values
- Quit button

## D. Application Implementation

The interface and application design can be implemented in many ways. The implementation is DBMS-dependent. A database developer should be knowledgeable about the development environment of the selected DBMS product. Therefore he or she will be able to pick an appropriate tool for application development. Database applications can be classified into two approaches according to their connectivity to the DBMSs.

### 1. Embedded SQL

The traditional approach to database application programming uses embedded SQL. Embedded SQL means the integration of SQL with a general-purpose programming language that is called a host language. The host language can be any high-level language such as C, COBOL, C++, Java, or BASIC. A DBMS specific preprocessor transforms the embedded SQL statements into function calls in the host language. The details of this translation vary across DBMSs, and therefore even though the source code can be compiled to work with different DBMSs, the final executable works only with one specific DBMS.

### 2. ODBC and JDBC

Open database connectivity (ODBC) and Java database connectivity (JDBC) are new technologies used in database application programming. They integrate SQL with a general-purpose programming language and they provide a standard way for the application programs to interact with the database through an application programming interface (API). In contrast to embedded SQL, ODBC and JDBC allow a single executable to access different DBMSs without recompilation. Applications using ODBC or JDBC are DBMS-independent at both the source code level and the executable level, while embedded SQL is DBMS-independent only at the source code level. In addition, when using ODBC or JDBC, an application can access not only one DBMS, but several different DBMSs simultaneously.

### 3. JDBC and Java Technology

JDBC contains Java classes and interfaces that provide low-level access to databases. It is the most prominent and mature approach for querying and modifying relational databases from Java applications. Coming after ODBC, JDBC defines a database access API that supports SQL functionality and enables access to a wide range of relational DBMS products. With JDBC, Java can be used as the host language for writing database applications. The goal of JDBC is to give the programmer the ability to write applets, servlets, and applications that are independent of any particular DBMS. JDBC contains two parts: an API for application writers and a lower-level driver API for driver writers. The JDBC API is a predefined Java library called java.sql that defines a set of interfaces and classes to be used for communicating with a database. The JDBC driver acts like a translator. It receives the client applications request in Java methods, translates it into a format that the database can understand, then presents the request to the database using the database native protocol. The response is received by the JDBC driver, translated back into Java data format, and presented to the client application. The 7 steps to query databases with JDBC are:

- Import the JDBC package
- Load and register the JDBC driver
- Establish connection to the database
- Create a SQL statement
- Execute a SQL statement
- Retrieve the query results
- Close the statement and connection

Java servlets are a new technology for developing server-based Web database applications. Servlets help provide secure access to a web site, interact with the database, dynamically generate HTML, and maintain unique session information for each client. Servlets communicate with thin clients—applications that require minimum client-side support. The server is responsible for the database access. Clients connect to the server using standard protocols such as JDBC available on all client platforms. Thus, the logical code can be written once and reside on the server for access by clients. Java servlets can be used in place of traditional CGI scripts for the following reasons:

- Easier to write
- Faster to run
- Not tied to any platform
- Multithreading locks prevent collisions
- Faster speed and persistent information
- Can parse XML
- A new tool to support JSP/Java Server Page

## E. Testing

Developed database applications need to be thoroughly tested before they go live. The testing should be based on the design specifications, performance requirements, and realistic data. If testing is conducted successfully, it will uncover errors within the application programs and possibly the database structure. The main testing strategies include:

- Top-down testing. Testing subsystems to find design errors early.
- Bottom-up testing. Testing lower-level functions, then working up to higher level modules.
- Stress testing. Testing the upper and lower limits of the system to find the defects that would not normally be identified.

# III. EVOLUTION OF DATABASE DEVELOPMENT

## A. History of Database Development

Database systems evolved from file processing systems. As database management systems become more and more popular, file processing systems are slowly fading from the scene. This is because database management systems are superior to file processing systems in the following ways

- Data dictionary management
- Data storage management
- Security management
- Multi-user access control
- Data integrity management
- Backup and recovery management

### 1. Hierarchical DBMS

In the late 1960s, IBM developed the first commercial hierarchical DBMS information management system (IMS), and its DL/1-language. The principles behind the hierarchical model are derived from IMS. Hierarchical DBMSs organize their record types in a hierarchical tree. This approach is well suited for large systems containing a lot of data. Hierarchical databases support two types of information—the record type which is a record containing data, and parent–child relations (PCR) which define a 1:N relationship between one parent record and N child-records. Hierarchical DBMS retrieve data fast because all the paths that link record types are predefined. Hierarchical databases are still used in many legacy systems and IMS is still the leading hierarchical DBMS used by a large number of banks, insurance companies, and hospitals as well as several government agencies.

### 2. Network DBMS

The original network model and language were presented in the CODASYL Database Task Group's 1971 report; hence it is also called the DBTO model. Revised reports in 1978 and 1981 incorporated more recent concepts. The network DBMS offers an efficient access path to its data and is capable of representing almost any informational structure containing simple types (e.g., integers, floats, strings, and characters). The network DBMS is more flexible than the hierarchical DBMS because it represents and navigates among 1:1, 1:N, and M:N relationships. But the programmer still has to know the physical representation of data to be able to access it. Accordingly, applications using a network database have to be changed every time the structure of the database changes, making database maintenance tedious. Both the hierarchical and network DBMSs were accessible from the programming language (usually COBOL) using a low-level interface.

### 3. Relational Databases

In 1970 Edgar F. Codd, at the IBM San Jose Research Laboratory, proposed a new data representation framework called the relational data model which established the fundamentals of the relational database model. Codd suggested that all data in a database could be represented as a tabular structure (tables with columns and rows, which he called relations) and that these relations could be accessed using a high-level nonprocedural (or declarative) language. Instead of writing programs to access data, this approach only needed a predicate that identified the desired records or combination of records. This would lead to higher programmer productivity and in the beginning of the 1980s several relational DBMS

(RDBMS) products emerged (e.g., Oracle, Informix, Ingres, and DB2).

The relational approach separates the program from the physical implementation of the database, making the program less sensitive to changes of the physical representation of the data by unifying data and metadata in the database. SQL and RDBMSs have become widely used due to the separation of the physical and logical representation (and of course to marketing). It is much easier to understand tables and records than pointers to records. Most significantly, research showed that a high-level relational query language could give performance comparable to the best record-oriented databases

## B. Current Trends in Database Development

Database technology is in a period of intensive change and innovation that is both revolutionary and evolutionary. The revolutionary aspect refers to the innovation of an object-oriented database management system (OODBMS) based on object-oriented programming technology. The evolutionary aspect refers to the promotion of a new extended version of relational database technology under the name object-relational database management system (ORDBMS). The success of relational DBMSs in the past decades is evident. However, the basic relational model and earlier versions of SQL proved inadequate to support object presentation. It has been said that traditional SQL DBMSs experience difficulty when confronted with the kinds of “complex data” found in application areas such as hardware and software design, science and medicine, document processing, mechanical and electrical engineering, etc. To meet the above challenges, object-oriented database management systems are proposed as an alternative to relational database management systems and are aimed at application domains where complex objects play a central role. The approach is heavily influenced by the paradigm of object-oriented programming languages and can be understood as an attempt to add functionality to support an object-oriented programming environment. The relational DBMSs are still dominant and evolving continuously, and in particular, have been incorporating many concepts that were developed in object databases. The object-relational DBMS emerged as a way of enhancing the capabilities of relational DBMSs with some of the features that appeared in object-oriented DBMSs. Figure 6 illustrates the current trends in the database development.

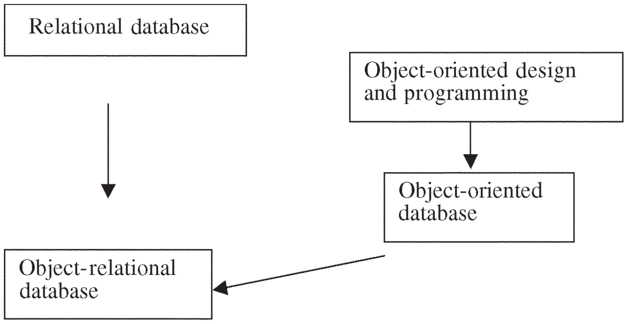

Relational database
Object-oriented design and programming
Object-oriented database
Object-relational database

**Figure 6** Current trends in database development.

### 1. Object-Oriented DBMS

Object-oriented database management systems are proposed as an alternative to relational database management systems and are aimed at application domains where complex objects play a central role. The approach is heavily influenced by the paradigm of object-oriented programming languages and can be understood as an attempt to add functionality to support an object-oriented programming environment. Zdonik and Maier specify that an OODBMS must, at a minimum, provide database functionality and data encapsulation and support object identity and objects with complex states. The current commercial OODBMSs are listed in Table III.

Recent database market surveys indicate that OODBMS will probably continue to occupy a niche within the database market. This niche will be characterized by applications that require very large amounts of data with several complex relations and with specialized data types. For example, the OODBMS seems likely to maintain its standing in computer-aided design (CAD), computer-aided manufacturing (CAM), geographical information system (GIS), specialized multimedia applications, mapping applications, simulation modeling, and scientific applications.

**Table III** **Object-Oriented DBMS Products by Vendors**

| Vendor | OODBMS |
|---|---|
| Computer Associates | Jasmine |
| Gemstone Systems, Inc. | GemStone |
| O2 Technology | O2 |
| Object Design | Object Store |
| Objectivity | Objectivity/DB |
| Versant Object Technology | Versant ODBMS |

### 2. Object-Relational DBMS

In 1990, Stonebraker *et al.* suggested extending the capabilities of a RDBMS to include support for richer object structures and rules. The resulting ORDBMS is a hybrid of the RDBMS and OODBMS which allows users to take advantage of OODBMS and to maintain a consistent data structure in an existing relational database. To overcome the identified weakness of relational DBMS, three of the leading RDBMS vendors, IBM, Informix, and Oracle have all extended their systems to ORDBMS (see Table IV).

ORDBMS will likely become dominant in most complex business applications because of the need to maintain compatibility with existing systems, the universal acceptance of relational databases, and the support of SQL-99, which added object-relational features.

**Table IV** **Object-Relational DBMS Products by Vendors**

| Vendor | RDBMS | ORDBMS |
|---|---|---|
| IBM | DB/2 | Universal database |
| Informix | Dynamic server | Universal server |
| Oracle | Oracle 7.x | Oracle 8i |

## SEE ALSO THE FOLLOWING ARTICLES

Database Administration • Database Systems • Data Modeling: Entity-Relationship Data Model • Data Modeling: Object-Oriented Data Model • Documentation for Software and IS Development • Network Database Systems • Object-Oriented Databases • Relational Database Systems • Structured Query Language (SQL)

## BIBLIOGRAPHY

Connolly, T. M.. and Begg, C. E. (2002). *Database systems: A practical approach to design, implementation, and management,* 3rd ed. Reading, MA: Addison–Wesley.

Date, C. J. (2000). *An introduction to database systems,* 7th ed. Reading, MA: Addison–Wesley.

Elmasri, R., and Navathe, S. B. (2000). *Fundamentals of database systems,* 3rd ed. Reading, MA: Addison–Wesley.

Ramakrishman, R., and Gehrke, J. (2000). *Database management systems,* 2nd ed. Boston: McGraw–Hill.

Riccardi, G. (2001). *Principles of database systems with internet and Java applications.* Reading, MA: Addison–Wesley.

Rob, P., and Coronel, C. (2001). *Database systems: Design, implementation, and management,* 4th ed. Cambridge, MA: Course Technology.

Ullman, J. D., and Widom, J. (1997). *First course in database systems.* Upper Saddle River, NJ: Prentice Hall.

# Database Machines

**Catherine M. Ricardo**
*Iona College*

## GLOSSARY

**bottleneck** A resource or part of a system that has smaller capacity than other components, resulting in delays while other parts wait for the resource to finish its task. For example, an I/O bottleneck occurs when the system must wait for input or output to finish before resuming a task.

**cache memory** A small, expensive storage area used to temporarily hold data that will be accessed frequently.

**charged couple device (CCD)** A type of solid-state integrated circuit that represents data as an electrical charge.

**concurrency control** The management of the interaction between two or more simultaneous processes so that they do not interfere with one another thereby causing errors.

**cylinder** The vertical set of tracks in the same relative position on the different surfaces of a disk pack.

**database management system (DBMS)** Software that supports the definition, creation, and maintenance of a database, and allows users and applications to access the data.

**data integrity** The correctness and internal consistency of data stored in a database. It is expressed as integrity constraints, which are rules that the database is not permitted to violate.

**data mining** The process of examining large databases with the purpose of finding new information by discovering patterns or rules of interest, usually as part of decision support systems for business.

**data warehouse** An integrated collection of data about a particular enterprise that represents facts about the subjects of that enterprise at a given period of time and is used for decision making.

**encryption** The process of encoding data so that it will not be readable by users or applications without a decryption key.

**hashing** A method of calculating the address for a record by using the value of a key field as input to an algorithm called the hashing algorithm. The output of the algorithm is the target address.

**join operation** An operation involving records in two tables that have a common attribute. Those records that have the same value for the common attribute are put together to form the join.

**partitioned data** Data that are split into disjoint fragments or subsets such that the union of the subsets gives back the original set of data.

**query** An operation on a database by which the user specifies which records, parts of records, or combinations of records are to be retrieved.

**redundant arrays of independent disks (RAID)** An architecture in which an array of small inexpensive disks is treated as a single logical disk. Data are striped or distributed over the multiple disks which can then be read in parallel to speed retrieval.

**replicated data** Data that are repeated and stored at more than one location.

## I. INTRODUCTION

A database machine or database computer is a device whose basic function is managing access to a database. The purpose of such a machine is to provide faster application processing through the offloading of the database access to this dedicated device. The concept of database machines became popular during the 1980s, when it was widely believed that these special-purpose machines would be in widespread use within a decade. Several vendors offered such systems. They used specialized architecture and storage technologies to optimize performance of database functions. Due to a variety of factors, including advances in technology and most notably the move to client-server architecture, these expectations for specialized database machines failed to materialize. However, with the continuing development of many very large databases, the growing popularity of data warehouses, and the development of data mining applications, all of which require storage of vast quantities of data and fast processing, dedicated database servers have become widely used. A database server is a machine that manages the database in a networked environment. Unlike a true database machine, a server is a general-purpose computer without specialized architecture or storage devices.

## II. FUNCTIONS OF A DATABASE MACHINE

In a traditional database environment, a general-purpose computer is used to run the database management system (DBMS), as well as a variety of other software and applications under its operating system. The database files reside on a disk that is under the computer's control. When a user or application program requests data, the computer processes the request and manages the disk controllers to access the data files. In a database machine environment, the general-purpose computer, called the host, does not run the DBMS software. Instead the DBMS runs on the database machine, a separate computer that controls the devices on which the database files reside. When a user or program requests data access, the request is submitted to the host, which passes it to the database machine for processing. The dedicated machine then performs the following functions:

- Accepts the data request and identifies which stored records will be needed to satisfy the request
- Checks that the user is authorized to access those items and to perform the requested operations on them
- Chooses the best path for data access
- Performs concurrency control so that other data requests submitted at the same time do not cause errors; this is necessary if at least one of the requests is an update
- Handles the recovery subsystem, to ensure that the database can be restored to a correct state in the event of a transaction or system failure
- Maintains data integrity, checking that no integrity constraints are violated
- Directs the actual data access using its device controllers
- Handles data encryption, if used
- Formats the retrieved data, if any
- Returns the data or results to the host machine

## III. DATABASE MACHINE ARCHITECTURE

The database machine concept can be implemented in several ways, including the use of backend machines, parallel processors, and specialized storage devices.

### A. Backend Machine

In the backend machine architecture, illustrated in Fig. 1, the database computer may be a general-

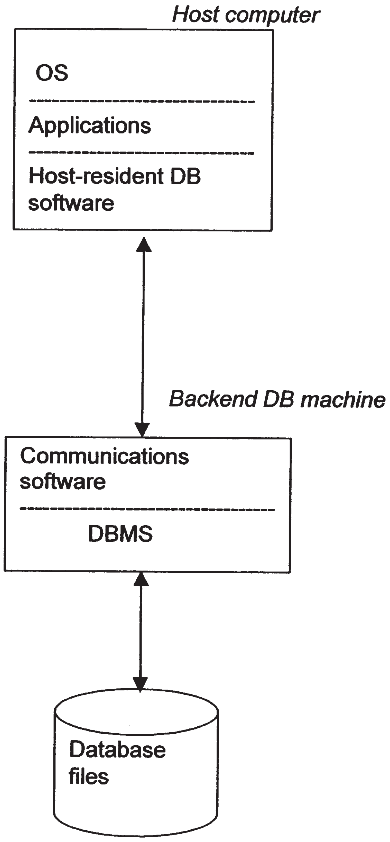


**Figure 1** Backend machine architecture.

purpose machine that is used as a server for database processing, or it may be a special-purpose computer. In either case, the host maintains the user interface, and the backend performs all the database access, as described in the previous section. If the backend is a special purpose machine, it may use a modified operating system, and some database functions such as join operations may be microcoded to optimize data access operations. The development of these special purpose computers is generally not cost-effective, so in practice servers are most often used today.

## B. Parallel Architecture

In a parallel database machine environment, instead of a single backend, there are multiple backend machines connected either to a single host or to multiple hosts. The backends control multiple disk units containing the database. The entire database may be replicated on several disks or the files may be partitioned over the disks. Data partitioning can be done by placing new records on successive disks in round-robin fashion, by hashing on some attribute, or by range partitioning, which means placing records by designating disks according to a range of values for a certain attribute. When a query is processed, since the required data may reside on different disks, the query is decomposed into subqueries that are then processed using the appropriate partition of the database. If there are multiple hosts, any of the hosts can communicate with any of the backends by using a programmable switch to direct data request traffic, and any of the backends can access data on any of the disks by means of a switch for data traffic. Figure 2 illustrates

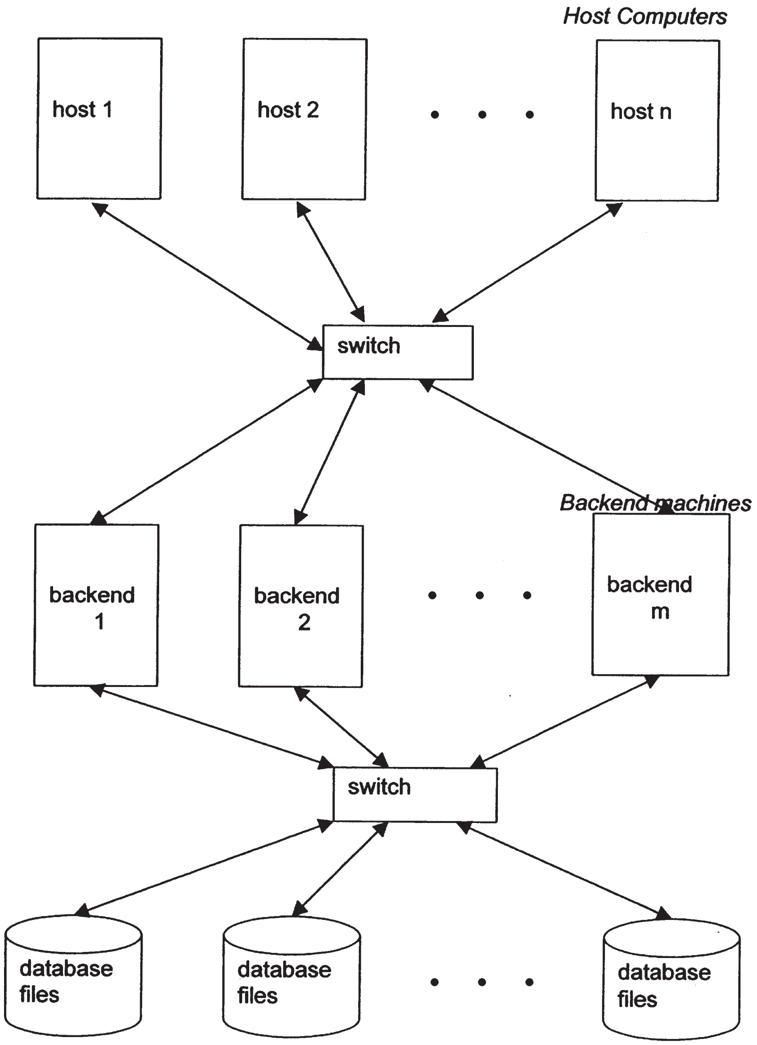


**Figure 2** Parallel architecture.

this architecture. The result is a very fault-tolerant system that can keep functioning despite failure of a host, a backend, or a disk unit. Each component can serve as a backup for the other components of its type, taking over its functions in the event of failure. This architecture allows data redundancy but the communications costs may be considerable. Parallel system architectures can be shared-memory, shared-nothing, shared-disk, or hierarchical (cluster). In a shared-memory design, all processors have access to any memory module or disk unit. In the shared-disk approach, all processors have access to any disk unit, but each has its exclusive access to its own memory. In shared-nothing architecture, each processor has exclusive control of its own memory and its own disk unit or units. A hierarchical or cluster architecture involves a shared-nothing machine made up of nodes that are shared-memory.

### C. Specialized Storage Devices

A variety of secondary storage technologies have been used for database machines, including charged couple device (CCD) memory, magnetic bubble memory, and head-per-track disks. In associative disk storage, an intelligent controller is used to manage data searching. Although a variety of techniques can be used to implement associative memory, all of these devices function as if there were a fixed read/write head containing a processor for each track. With the ability to perform logical comparisons on the track, data items can be retrieved by value, rather than by location. By moving the search logic to the controller hardware as the data is being read from disk, it is possible to retrieve only records that qualify for the query being processed, reducing the amount of data that is transmitted to the buffer. The effect is the offloading of the basic retrieval functions from the host computer to these units. Eventually these specialized storage units were found to be prohibitively expensive, and parallel disks were used instead.

## IV. ADVANTAGES OF DATABASE MACHINES

Using database machines or database servers can provide the following benefits, depending on the architecture chosen:

- Increased performance. In a backend architecture, the offloading of the database management and operations to another device frees the host computer, allowing it to perform other processing in parallel with the database processing. The use of a specialized operating system and microcoding of database operations can make the backend itself extremely efficient. If parallel database machine architecture is used, processing power is automatically increased. The use of associative storage likewise offloads some of the processing load from the host, allowing it to perform other functions.
- Database sharing. A single database machine can function as a backend for multiple host computers, allowing many users to share data.
- Increased reliability. If multiple servers are used, the system can continue to function despite the failure of one of the servers.
- Increased data availability. If the database is replicated and one of the copies is not available, a backup copy can be used, allowing processing to continue. If the database is partitioned, processing involving other sections of the database can continue.
- Improved security. If a single backend machine controls all access to the database files, all data requests must be directed to that machine, allowing centralization of the security subsystem. User authorization can be checked before any database access. Since there is no user interface on the database machine, all requests must also go through the host computer, which can apply initial access control checks.
- Greater flexibility. Because a variety of architectures can be used, the system designer can choose the configuration that best suits the environment. Additional backends or disk units may be added in a modular fashion as needed.
- Load balancing. If there are multiple servers, the system can direct requests to the machine that is least busy at a given time to balance processing load.

## V. CHALLENGES FOR DATABASE MACHINES

There are also major challenges facing designers and developers of database machines, including:

- *Cost.* The major disadvantage to database machines has been the cost of developing the systems. With rapid advances in technology, it is not cost-effective to develop specialized machines that depend heavily on technologies that soon become obsolete.
- *I/O bottleneck.* The I/O bottleneck refers to the slow speed of input/output relative to processing

speed. Since most databases use secondary storage devices, typically disk, it is necessary to consider the delay involved in locating and transferring records from these devices. Various methods have been used to attempt to address this problem. For example, clustering of records, that involves placing records that are most often accessed together on the same cylinder or adjacent cylinders, can be used to reduce seek time. This technique can be used by many DBMs, but it was an inherent feature of early database machines, including DBC and DIRECT. Another solution is using specialized hardware. For example, disk units with fixed read/write heads per track were used in some early machines, such as Data Base Computer (DBC). These expensive specialized units were replaced by arrays of moving-head disks in later database machines, such as DBC/1012, GAMMA, and Active-Graph Machine (AGM). RAID technology is used by most systems today.

- *Scalability*. Scalability refers to the ability of a database machine to handle larger tasks quickly. The usual solution is to use parallel machines to spread the processing load; however, this presents the problem of control among the units. If the units are controlled centrally, a bottleneck may be introduced, since all units need access to the controller. A better solution is distributed control, as found in DBC/1012 and GAMMA. Parallel machines ideally have both linear speedup and linear scaleup. With linear speedup, if the hardware is doubled, the processing job should be completed in half the time. With linear scaleup, if the hardware is doubled, it is possible to perform a job that is twice as large in the same amount of time.

# VI. EXAMPLES

## A. Overview

Britton-Lee, Tandem, and Teradata all offered early commercial specialized database machines. Research and academic institutions designed or prototyped several others, including DBC at Ohio State, DIRECT and GAMMA at the University of Wisconsin, and AGM at the University of California, Irvine. Other research systems included Bubba, Arbre, Volcano, SDC, XPERS, and Persist. Current commercial database machines or database servers include the NCR World-Mark series and WhiteCross Data Exploration Server. Among current DBMS providers, IBM produces Parallel Sysplex, a shared everything clustering architecture combining DB2 with OS/390 in a tightly coupled environment that provides many of the same benefits as a database machine. Informix and Oracle work with several hardware vendor partners to maximize database performance on those machines.

## B. Britton-Lee IDM

Britton-Lee developed one of the first commercial database machines, the Intelligent Database Machine, IDM, released in 1981. It was an integrated hardware and software system that ran the IDM RDBMS (Relational Database Management System) based on the INGRES DBMS, and functioned as a backend for mainframes from several vendors, including DEC and IBM. The IDM host resident software installed on the host computer allowed the mainframe access to the IDM backend. Host and backend communicated using IDM Network communications software that linked the IDM host resident software and RDBMS, the backend relational DBMS. IDM had a specialized operating system modified for efficient data access, microprocessors designed for specific tasks, a database accelerator that optimized execution of common processes, and disk cache for frequently used records. Subsystems managed security, transaction management, query optimization, concurrency control, backup and recovery, and other tasks. It had its own language, Intelligent Database Language (IDL), for database creation and management.

## C. Tandem Non-Stop System

Also in the early 1980s, Tandem developed its Non-Stop System, an early parallel architecture system that featured multiple processors and multiple disks. The system was designed to provide high performance and reliability by duplication of components. The basic hardware consisted of 2–16 processors linked by an InterProcessor Bus. Each processor module consisted of a central processing unit, a memory unit, an I/O channel, and an InterProcessor Bus interface. Figure 3 shows the basic architecture of the simplest Tandem system. Note that while any processor can control any disk controller, only one processor actually has that control at any given time. To increase reliability, each query, transaction, or application is assigned to one primary processor and a backup processor, that can take over execution if the primary one fails. Reliability is further increased by mirrored disks, writing all data to two disks, so that processing can continue even if a disk failure occurs.

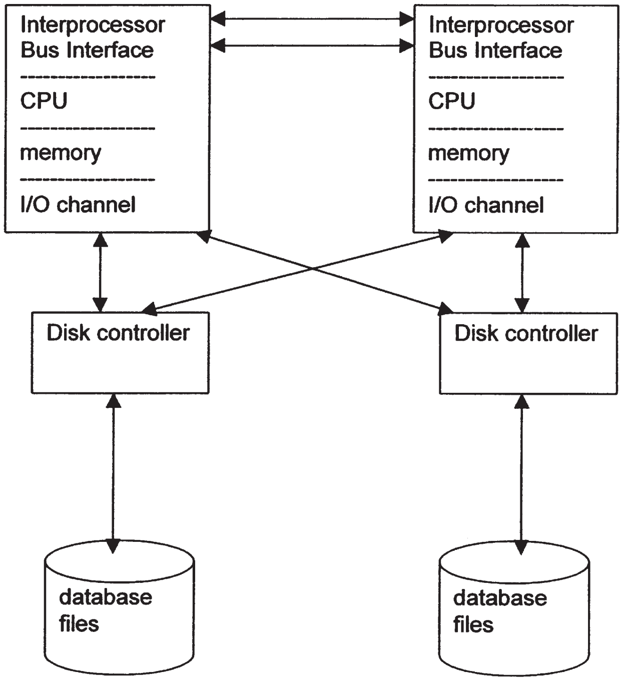


**Figure 3** Tandem Non-Stop architecture.

The original group of Tandem database software products, collectively called ENCOMPASS, included

- ENSCRIBE, an RDBMS
- DDL, a data definition language
- A data dictionary
- ENFORM, a query language
- PATHWAY, a facility that provides an interface for designing and controlling transactions
- TMF, a transaction management facility that also handles recovery

Tandem's Non-Stop architecture is still in use. The company later paired with Compaq to address the needs of smaller customers.

## D. Teradata DBC

Teradata also developed backend machines to interface with large mainframes. The DBC/1012 Model 1 was announced in 1984 as a backend for the IBM 370, and later models served as backends for a variety of mainframes, minicomputers, and workstations. Teradata's pioneering DBC/1012 systems used parallel processing to provide high performance, load balancing, and high reliability. Figure 4 shows a sample configuration for this system. The parallel database processors, called access module processors (AMPs), manage disk storage units that can hold up to 500 megabytes each. Data are partitioned across the disk units in such a way that records from the same table are spread across several disk units. Hashing is used to determine record placement. A set of interface processors (IFPs) provides communications between the mainframe and the AMPs. The system is linked by an intelligent interconnection network, called YNet, a bus that routes messages between the IFPs and the AMPs and handles some other tasks. The system was designed to be highly modular, allowing designers to add additional microprocessors to handle larger databases. It handled up to 1024 processors, each of which could handle up to four disk units. The basic system components are

- Host computers, which could be mainframes or a variety of minicomputers or workstations, connected on a network.
- TDP, Teradata Director Program, software that resides in each host machine to handle activity between the host and the Teradata backend.
- IFPs, at least two per mainframe host, each having a channel interface controller, a processor, and two interfaces to the bus; their job is to receive requests from the hosts, interpret and decompose them, route subrequests, and consolidate responses for the requesting process.
- COPs, communication processors that perform the same services for the minicomputers or workstations. Each has a processor, two interfaces to the bus, and a network adapter.
- YNet, the intelligent bus that carries communications from the IFPs and COPs to the AMPs, manages interprocessor communications between the AMPs, consolidates results of subrequests, and returns results to the IFP or COP that submitted the request. To ensure fault tolerance, the system must have two YNets to serve as backups for each other.
- AMPs, each of which manages the data on up to four disk units.
- Disk units, on which the data is partitioned using a hashing scheme for parallel processing. Because of the partitioning, several disk units are searched in parallel in answering each query. To ensure data availability, primary and secondary copies of data could be made. These were high-volume Winchester disks. A two-megabyte disk cache was available for each AMP.

## E. NCR WorldMark

Teradata later paired with NCR. The NCR 3700 was the successor to the DBC 1012 and used essentially

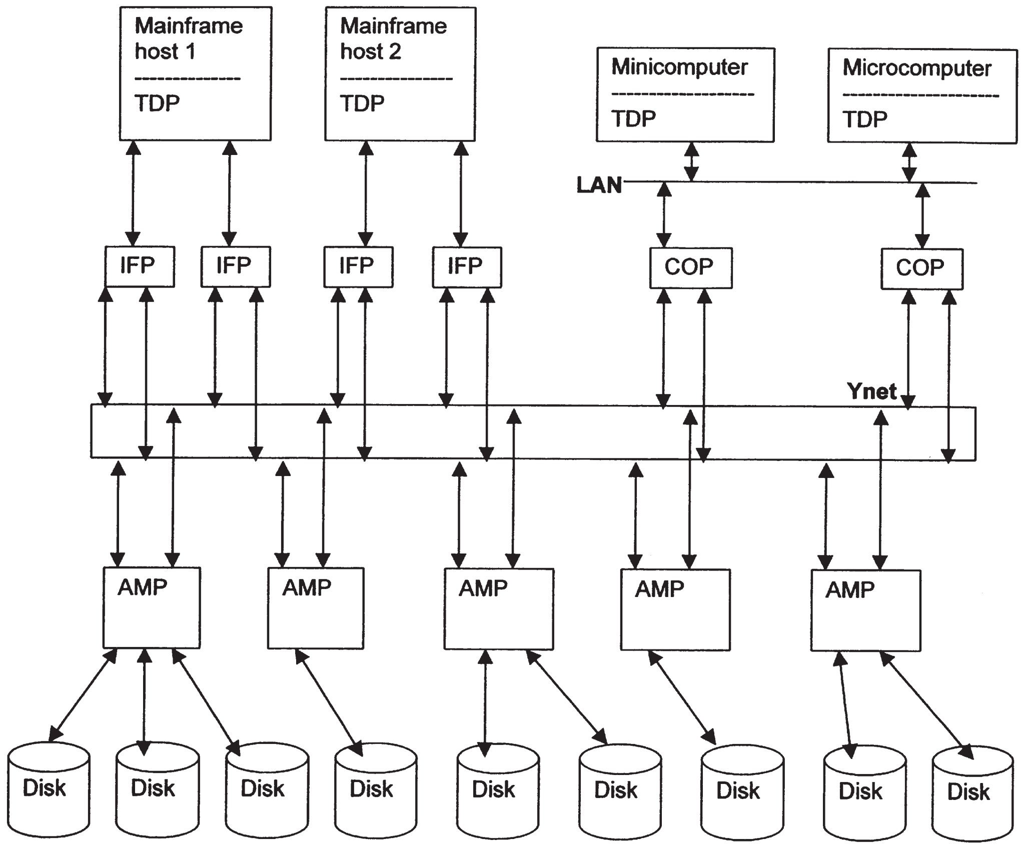


**Figure 4** Teradata DBC/1012 architecture.

the same technology. Recently, NCR has focused on providing scalable environments for data warehousing, including Windows- and UNIX-based systems. Models in NCR's WorldMark series use the Teradata MPP (massively parallel processing) architecture, including the interconnection now called BYNET, and scalability from 2–512 nodes, handling up to over 100 terabytes, as well as RAID technology.

## F. WhiteCross Data Exploration Server

WhiteCross Technologies offers a database hardware and software package that targets the data mining market. Its Data Exploration Server, WX/DES, uses a linearly scalable MPP server architecture. It employs very large memory (VLM) technology for fast in-memory processing of large databases, eliminating the need for indexes. Query speed is reported to be 5 million rows per second per processor, with linear speedup as additional processors are added. It can support an unlimited number of communications processors, which can feed data from a variety of sources in parallel. It uses RAID technology and disk swapping for fault tolerance. The architecture has been optimized for processing of joins and complex queries. It allows concurrent data loading and data analysis. WhiteCross provides a proprietary software suite, Exploration Studio, for data mining. It includes MiningSTUDIO, which provides decision tree and rule-based algorithms, neural network models, automatic cluster detection, data profiling and data visualization, and other tools. QuerySTUDIO provides visual tools for users' queries, and ApplicationSTUDIO is a software development kit using DCOM/ActiveX technology for writing applications in a variety of languages.

## SEE ALSO THE FOLLOWING ARTICLES

Database Development Process • Database Systems • Data Mining • Data Warehousing and Data Marts • Distributed Databases • Encryption • Relational Database Systems • Structured Query Language

## BIBLIOGRAPHY

Boral, H., and DeWitt, D. (1983). Database machines: An idea whose time has passed? A critique of the future of database machines. *Proceedings of the 1983 Workshop on Database Machines* (H. O. Leilich and M. Misikoff, ed.), pp. 166–187. New York: Springer-Verlag.

Boral, H., Alexander, W., Clay, L., Copeland, G., Danforth, S., Franklin, M., Hart, B., Smith, M., and Valduriez, P. (1990). Prototyping bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering,* 2, 1, 4–24.

Chen, P., Lee, E., Gibson, G., and Katz, R. (1994). RAID high performance, reliable secondary storage. *ACM Computing Surveys,* 26, 2, 145–185.

DeWitt, D., Gerber, R., Graek, G., Heytens, M., Kuman, K., and Muralikrishna, M. (1986). GAMMA: A high performance dataflow database machine. *Proceedings of 12th International Conference on Very Large Data Bases,* 107–121.

DeWitt, D., and Gray, J. (1992). Parallel database systems: The future of high performance database systems. *Communications of the ACM,* 35, 6, 85–98.

Hsiao, D., ed. (1983). *Advanced database machine architecture,* Englewood Cliffs, NJ: Prentice-Hall.

Hsiao, D. (1986). Future database machine architectures. in *New Directions for Database Systems* (G. Ariav and J. Clifford, eds.), pp. 12–18, New Jersey: Ablex.

Ozsu, M. T., and Valduriez, P. (1999). *Principles of distributed database systems, 2nd ed,* Upper Saddle River, NJ: Prentice-Hall.

Qadah, G. (1985). Database machines: A survey. *AFIPS Conference Proceedings,* National Computer Conference, 54, 211–223.

Su, S. (1988). *Database computers: Principles, architectures and techniques.* New York: McGraw-Hill.

# Database Systems

**Abraham Silberschatz**
*Bell Laboratories*

**Henry F. Korth**
*Bell Laboratories*

**S. Sudarshan**
*Indian Institute of Technology, Bombay*

## GLOSSARY

**data manipulation language** A language that enables users to access or manipulate data as organized by the appropriate data model.

**data model** A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

**hard disk** A data storage device where data are encoded magnetically on the surfaces of one or more platters spinning on a common axis. The disk has an arm containing a read-write head for each platter surface, and the arm pivots to position the read-write head at a desired distance from the center of the platter.

**security** Protection of data from unauthorized access and update. Includes mechanisms to authenticate users and to control which users are authorized to access specific data.

**transaction** A unit of activity which should be executed atomically, that is, be completed or appear to never have started. Updates made by a completed transaction should be durable, that is, not lost even if there are system failures.

**A DATABASE-MANAGEMENT SYSTEM (DBMS)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results such as lost updates or inconsistent data.

## I. DATABASE SYSTEM APPLICATIONS

Databases are widely used in a variety of application domains. Here are some representative applications:

- Banking—for customer information, accounts, loans, and banking transactions
- Airlines—for reservations and schedule information (Airlines were among the first to use databases in a geographically distributed manner. Terminals situated around the world accessed the central database system through phone lines and other data networks.)
- Universities—for student information, course registrations, and grades
- Electronic commerce—for product catalogs, order and shipping data, billing, and customer data

Portions of this article were adapted by permission from Chapter 1 of *Database System Concepts,* 4th ed., by Silberschatz, Korth, and Sudarshan, McGraw–Hill, 2001.

- Telecommunication—for translation of toll-free phone numbers, portability of local phone numbers among competing service providers, and home-location databases for wireless networks

Other applications include billing for credit-card transactions, network management and billing for telecom, financial applications such as stock markets and investment-banking companies, retail and online sales, manufacturing, and human-resource functions such as salaries, payroll taxes, and so on.

As the list illustrates, databases form an essential part of almost all enterprises today. Most of us interact frequently with databases, sometimes explicitly even without our knowledge. For example, when you access a bank Web site and retrieve your bank balance and transaction information, the information is retrieved from the bank's database system. When you access a Web site, information about you may be retrieved from a database, to select which advertisements should be shown to you. Furthermore, data about your Web accesses may be stored in a database. Although user interfaces hide details of access to a database, and most people are not even aware they are dealing with a database, accessing databases forms an essential part of almost everyone's life today.

The importance of database systems can be judged in another way—today, database system vendors like Oracle are among the largest software companies in the world, and database systems form an important part of the product line of more diversified companies like Microsoft and IBM.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data. This article provides a brief overview of the issues in the design and development of database system applications, as well as an overview of the internal structure of database systems.

## II. HISTORY OF DATABASE SYSTEMS

Data processing drives the growth of the computer industry, as it has from the earliest days of commercial computers. In fact, automation of data processing tasks predates computers. Punched cards, invented by Hollerith, were used at the very beginning of the 20th century to record U.S. census data, and mechanical systems were used to process the cards and tabulate results. Punched cards were later widely used as a means of entering data into computers.

Techniques for data storage and processing have evolved over the years and have been driven by two factors: the technology for storage of data, and application requirements. In the 1950s and 1960s, magnetic tapes were the primary means of data storage. A typical database application of the era managed the payroll of an organization, storing details of salaries, taxes, and so on as records of a file, stored sequentially on a tape.

Processing of data consisted of reading data from one or more tapes and writing data to a new tape. Data could also be input from punched card decks, and output to printers. For example, salary raises were processed by entering the raises on punched cards and reading the punched card deck in synchronization with a tape containing the master salary details. The records had to be in the same sorted order. The salary raises would be added to the salary read from the master tape and written to a new tape; the new tape would become the new master tape.

Tapes (and card decks) could be read only sequentially, and data sizes were much larger than main memory; thus, data processing programs were forced to process data in a particular order, by reading and merging data from tapes and card decks.

The emergence of hard disks in the late 1960s changed the scenario for data processing greatly, since hard disks allowed direct access to data. The position of data on disk was immaterial, since any location on disk could be accessed in just tens of milliseconds. Data were thus freed from the tyranny of sequentiality.

With direct access to data on disks, it was possible to store data structures such as lists and trees on disk. A node of a list or tree could store the disk address of another node, which could then be accessed within tens of a millisecond, instead of minutes. Application programmers could construct and manipulate these data structures.

Direct access to data also made possible online access to data. It was possible to accept a request for even a single piece of information, such as seat availability on a particular flight, and efficiently locate it using "index" structures, and answer the request. Similarly, a reservation request could be processed efficiently by updating data at a few locations on the disk without having to read or write all data on the disk.

The "network" and "hierarchical" data models developed in this era provided an abstract view of these data structures, hiding some of the low level implementation details of the data structures. However, application programmers had to deal with the task of navigating the data structures to find required data.

A landmark paper by Codd in 1970 defined the relational model, where data are represented in tabular form, and data are accessed in a nonprocedural manner without the need for the user or programmer to be concerned about the manner in which the data are stored. The simplicity of the relational model and the possibility of hiding implementation details completely from the programmer were enticing indeed. Codd later won the prestigious Association of Computing Machinery Turing Award for his work.

Although academically interesting, the relational model was not used in practice initially, because of its perceived performance disadvantages; relational databases could not match the performance of existing network and hierarchical databases. That changed with System R, a groundbreaking project at IBM Research that developed techniques for the construction of an efficient relational database system. Initial commercial relational database systems, such as IBM DB2, Oracle, Ingres, and DEC Rdb, played a major role in advancing techniques for efficient processing of declarative queries.

The 1980s saw the emergence of the object-oriented database model, employing concepts from the object-oriented programming paradigm such as inheritance, object identity, encapsulation, and a rich type system. The 1990s saw these concepts merge into the relational model via object-relational database systems.

In recent years, the XML language has evolved from being a document markup language to an emerging standard for data interchange among applications, especially web-based ones. The management of XML data represents one of the major challenges to database researchers and product developers today.

## III. DATABASE SYSTEM FUNCTIONALITY

Database systems, in the most general sense, provide the means for storing data and facilities to query the data. Databases also provide features to ensure integrity and security of data, to deal with software and hardware failures, and to deal with problems due to concurrent access.

It was these challenges, among others, that prompted the development of database systems. In later sections, we shall see the concepts and algorithms that enable database systems to solve these difficulties. We use a bank enterprise as a running example of a typical data-processing application found in a corporation.

Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. A typical *file-processing system* is implemented directly on a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.

Before database management systems came along, organizations usually stored information in such systems. Keeping organizational information in a file-processing system has several disadvantages that are addressed by database systems. Contrasting the two approaches helps highlight the benefits of using database systems.

### A. Data Representation

Consider how a bank may store information about accounts and customers who own accounts. Suppose a bank provides both checking and savings accounts and maintains one file for each account type. The file contains a large number of records, each containing information about the account, such as the account number, balance available, and so on. In a file-processing system, the application code that fetches and updates information must deal with low level details such as the format in which information is stored.

In contrast, a database system provides a higher-level view of the data. For instance, a relational database provides a view of data as consisting of a number of records, each containing several attributes. Lower level details of how each record is stored in a file are hidden from applications. Providing such a layer of abstraction also enables the database system to implement efficient but more complex means of storing data, without burdening application programs with details of data storage.

Deciding how to organize information logically in files is another challenge. For instance, the bank must store information about customers. A straightforward method of doing so would be to store customer information, such as address and telephone number, along with the account records. Now suppose a customer has both savings and checking accounts. That customer's information would be stored *redundantly,* in two places. This redundancy leads to higher storage and access cost. In addition, it may lead to *data inconsistency;* that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

A better design would keep customer information in a separate file, with each customer identified by a customer identifier, and store only the customer identifier with the account information. Several techniques are available for designing how to represent information. Underlying these techniques is a *data model,* which is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. Data models and techniques for database design are described in Section IV.

## B. Data Manipulation

A database system provides convenient mechanisms for users to manipulate stored data. By contrast, in a file-processing system a number of application programs would be required to manipulate the files, including

- A program to debit or credit an account
- A program to add a new account
- A program to find the balance of an account
- A program to generate monthly statements

Application programmers would write these application programs to meet the needs of the bank. New application programs are added to the system as the need arises. Thus, as time goes by, the system acquires more files and more application programs.

Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* customers. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same officer needs to trim that list to include only those customers who have an account balance of $10,000 or more. As expected, a program to generate such a list does not exist. Again, the officer has the preceding two options, neither of which is satisfactory.

The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

Database systems provide *query languages* that simplify the task of retrieving desired information. These languages are “declarative,” that is, they allow the user to specify what is required without being bothered about how exactly to get the required information. The database system takes on the responsibility of understanding and fulfilling the user’s request. Database query languages are described in Section V.

## C. Integrity and Security

The data values stored in the database must satisfy certain types of *integrity constraints* (also known as *consistency constraints*). For example, the balance of a bank account may never fall below a prescribed amount (say, $25).

Database systems allow such constraints to be specified explicitly, and reject any updates to the database that cause the constraints to be violated. If a file-processing system were used instead of a database, such constraints can be ensured by adding an appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts.

Database systems allow explicit specification of which users are allowed what sort of access (such as read or update) to what data. If a file-processing system were used, application programs would require an extra code to verify if an attempted access is authorized. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult.

## D. Failures and Concurrency Problems

A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *B,* resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all.

It is difficult to ensure atomicity when writing applications accessing files in a conventional file-processing system. In contrast, database systems provide built-in support to ensure atomicity.

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A,* containing $500. If two customers withdraw funds (say $50 and $100, respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350.

To guard against such *concurrent-access anomalies,* database systems supervise data access by concurrent accesses and ensure that the database state remains consistent. If a file system were used directly, supervision would be difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

Section VIII describes how database systems provide support for atomicity, control of concurrency, and related properties.

## IV. DATA MODELS

A database system is a collection of interrelated files and a set of programs that allow users to access and modify these files. The need for efficiency has led designers to use complex data structures to represent data in the database. The low-level *physical view* of data is too detailed for most users. A major purpose of a database system is to provide users with an *abstract* or *logical view* of the data. That is, the system hides certain details of how the data are stored and maintained.

Underlying the structure of a database is the *data model:* a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. To illustrate the concept of a data model, we outline two data models in this section: the entity-relationship model and the relational model. Both provide a way to describe the design of a database at the logical level.

### A. The Entity-Relationship Model

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities,* and of *relationships* among these objects. The entity-relationship model is widely used in database design. An entity is a “thing” or “object” in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities.

Entities are described in a database by a set of *attributes.* For example, the attributes *account-number* and *balance* may describe one particular account in a bank, and they form attributes of the *account* entity set. Similarly, attributes *customer-name, customer-street* address, and *customer-city* may describe a *customer* entity.

An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city). A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.

A *relationship* is an association among several entities. For example, a *depositor* relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an entity set and relationship set, respectively.

The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram,* which is built up from the following components:

- Rectangles, which represent entity sets
- Ellipses, which represent attributes
- Diamonds, which represent relationships among entity sets
- Lines, which link attributes to entity sets and entity sets to relationships

Each component is labeled with the entity or relationship that it represents.

As an illustration, consider part of a database banking system consisting of customers and the accounts that these customers have. Figure 1 shows the corresponding E-R diagram. The E-R diagram indicates that there are two entity sets, *customer* and *account,* with attributes as outlined earlier. The diagram also shows a relationship *depositor* between customer and account.

In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is *mapping cardinalities,* which express the number of entities to which another entity can be associated via a relationship set. For example, if each

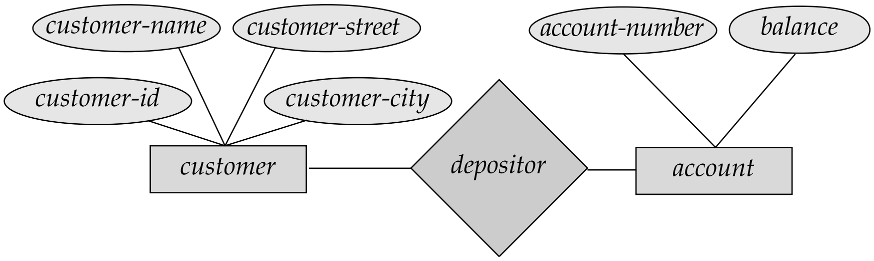


**Figure 1** A sample E-R diagram.

account must belong to only one customer, the E-R model can express that constraint.

## B. Relational Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Figure 2 presents a sample relational database comprising three tables: One shows details of bank customers, the second shows accounts, and the third shows which accounts belong to which customers.

| *customer-id* | *customer-name* | *customer-street* | *customer-city* |
|---|---|---|---|
| 192-83-7465 | Johnson | 12 Alma St. | Palo Alto |
| 019-28-3746 | Smith | 4 North St. | Rye |
| 677-89-9011 | Hayes | 3 Main St. | Harrison |
| 182-73-6091 | Turner | 123 Putnam Ave. | Stamford |
| 321-12-3123 | Jones | 100 Main St. | Harrison |
| 336-66-9999 | Lindsay | 175 Park Ave. | Pittsfield |
| 019-28-3746 | Smith | 72 North St. | Rye |

(a) The *customer* table

| *account-number* | *balance* |
|---|---|
| A-101 | 500 |
| A-215 | 700 |
| A-102 | 400 |
| A-305 | 350 |
| A-201 | 900 |
| A-217 | 750 |
| A-222 | 700 |

(b) The *account* table

| *customer-id* | *account-number* |
|---|---|
| 192-83-7465 | A-101 |
| 192-83-7465 | A-201 |
| 019-28-3746 | A-215 |
| 677-89-9011 | A-102 |
| 182-73-6091 | A-305 |
| 321-12-3123 | A-217 |
| 336-66-9999 | A-222 |
| 019-28-3746 | A-201 |

(c) The *depositor* table

**Figure 2** A sample relational database.

The first table, the *customer* table, shows, for example, that the customer identified by customer-id 192-83-7465 is named Johnson and lives at 12 Alma St. in Palo Alto. The second table, *account,* shows, for example, that account A-101 has a balance of $500, and A-201 has a balance of $900.

The third table shows which accounts belong to which customers. For example, account number A-101 belongs to the customer whose customer-id is 192-83-7465, namely Johnson, and customers 192-83-7465 ( Johnson) and 019-28-3746 (Smith) share account number A-201 (they may share a business venture).

The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type.

It is not hard to see how tables may be stored in files. For instance, a special character (such as a comma) may be used to delimit the different attributes of a record, and another special character (such as a newline character) may be used to delimit records. The relational model hides such low-level implementation details from database developers and users.

The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model. One of the reasons for the success of the relational model is its strong mathematical foundation, which has helped in the design of declarative query languages. The mathematical foundation has also helped in the construction of efficient query evaluation systems.

The relational model is at a lower level of abstraction than the E-R model. Database designs are often carried out in the E-R model, and then translated to the relational model. For example, it is easy to see that the tables *customer* and *account* correspond to the entity sets of the same name, while the table *depositor* corresponds to the relationship set *depositor.*

We also note that it is possible to create schemas in the relational model that have problems such as unnecessarily duplicated information. For example, suppose we store *account-number* as an attribute of the *customer* record. Then, to represent the fact that accounts A-101 and A-201 both belong to customer Johnson (with customer-id 192-83-7465), we would need to store two rows in the customer table. The values for customer-name, customer-street, and customer-city for Johnson would get unnecessarily duplicated in the two rows.

Relational database design theory offers a formal way of distinguishing good schema designs from bad schema designs. The process of *normalization* deals with the conversion of any given schema into one that satisfies the requirements of a good schema design.

### C. Other Data Models

The object-oriented data model is another data model that has seen increasing attention. The object-oriented model can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.

The object-relational data model combines features of the object-oriented data model and relational data model.

Semistructured data models permit the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The extensible markup language (XML) is widely used to represent semistructured data.

Historically, two other data models, the network data model and the hierarchical data model, preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are little used now, except in old database code that is still in service in some places.

### D. Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an *instance* of the database. The overall design of the database is called the database *schema*. Schemas are changed infrequently, if at all.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an instance of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction. The physical schema describes the database design at the physical level, while the logical schema describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema and can usually be changed easily without affecting application programs. Application programs are said to exhibit *physical data independence* if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

At the lowest level, the database stores information as an instance of the physical schema. However, the database allows the same information to be viewed as an instance of the logical schema.

## V. DATABASE LANGUAGES

A database system provides a data definition language to specify the database schema and a data manipulation language to express database queries and updates. In practice, the data definition and data manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

### A. Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a *data-definition language* (DDL).

For instance, the following statement in the SQL language defines the *account* table:

**create table** *account*

(*account-number* **char**(10),

*balance* **integer**)

Execution of the above DDL statement creates the account table. In addition, it updates a special set of tables called the data dictionary or data directory.

A data dictionary contains *metadata*—that is, data about data. The schema of a table is an example of metadata. A database system consults the data dictionary before reading or modifying actual data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a *data storage and definition* language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

The data values stored in the database must satisfy certain consistency constraints. For example, suppose the balance on an account should not fall below $100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated.

## B. Data-Manipulation Language

Data manipulation is

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database

A *data-manipulation language* (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. There are basically two types:

- *Procedural* DMLs require a user to specify *what* data are needed and *how* to get those data.
- *Declarative* DMLs (also referred to as *nonprocedural* DMLs) require a user to specify *what* data are needed *without* specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural.

A *query* is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a *query language.* Although technically incorrect, it is common practice to use the terms *query language* and *data-manipulation language* synonymously.

This query in the SQL language finds the name of the customer whose customer-id is 192-83-7465:

**select** *customer.customer-name*

**from** *customer*

**where** *customer.customer-id* = '192-83-7465'

The query specifies that those rows *from* the table *customer where* the *customer-id* is 192-83-7465 must be retrieved, and the *customer-name* attribute of these rows must be displayed. If the query were run on the table in Fig. 2, the name Johnson would be displayed.

Queries may involve information from more than one table. For instance, the following query finds the balance of all accounts owned by the customer with customer-id 192-83-7465.

**select** *account.balance*

**from** *depositor, account*

**where** *depositor.customer-id* = '192-83-7465' **and**

*depositor.account-number* = *account.account-number*

If the above query were run on the tables in Fig. 2, the system would find that the two accounts numbered A-101 and A-201 are owned by customer 192-83-7465 and would print out the balances of the two accounts, namely 500 and 900.

There are a number of database query languages in use, either commercially or experimentally, of which the SQL language is by far the most widely used.

# VI. DATABASE APPLICATION ARCHITECTURE

Application programs are programs that are used to interact with the database. Application programs are usually written in a *host* language, such as Cobol, C, C++, or Java. Examples in a banking system are programs that generate payroll checks, debit accounts, credit accounts, or transfer funds between accounts.

To access the database from an application program, DML statements need to be executed from the host language. There are two ways to do this:

- By providing an application program interface (set of procedures) that can be used to send DML and DDL statements to the database, and retrieve the results. The Open Database Connectivity (ODBC) standard defined by Microsoft for use with the C language is a commonly used application program interface standard. The Java

Database Connectivity ( JDBC) standard provides corresponding features to the Java language.
- By extending the host language syntax to embed DML calls within the host language program. Usually, a special character prefaces DML calls, and a preprocessor, called the *DML precompiler,* converts the DML statements to normal procedure calls in the host language.

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between *client* machines, on which remote database users work, and *server* machines, on which the database system runs.

Database applications are usually partitioned into two or three parts, as in Fig. 3. In a *two-tier architecture,* the application is partitioned into a component that resides at the client machine and that invokes the database system component at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

In contrast, in a *three-tier architecture,* the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an *application server.* The application server in turn communicates with a database system to access data. The *business logic* of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications and for applications that run on the World Wide Web.

Although the term application server can be applied to servers for any applications, the term is increasingly being used to refer to systems that manage an entire range of enterprise activities, such as financial applications, personnel management, sales, inventory/stores management, as well as business planning activities.

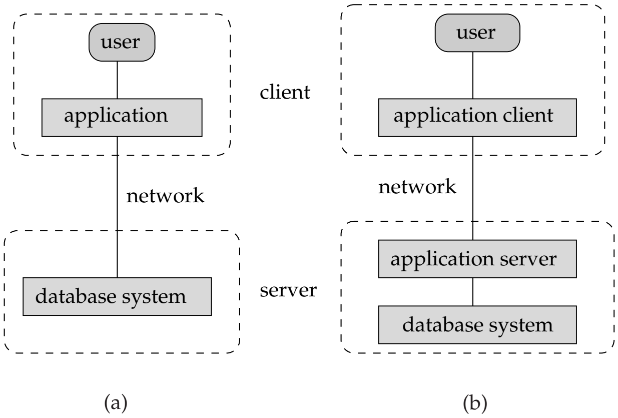


**Figure 3** (a) Two-tier and (b) three-tier architectures.

## VII. DATABASE USERS AND ADMINISTRATORS

People who work with a database can be categorized as database users or database administrators. Users can be differentiated by the way they expect to interact with the system.

- *Naive users* are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. Examples include bank tellers and Web users who interact with databases through Web forms.
- *Application programmers* are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. There are also special types of programming languages that combine imperative control structures (for example, for loops, while loops, and if-then-else statements) with statements of the data manipulation language.
- *Sophisticated users* interact with the system without writing programs. Instead, they form their requests either by using a database query language or by using specialized tools. Analysts who submit queries to explore data in the database fall in this category.

*Online analytical processing* (OLAP) tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region). The tools also permit the analyst to select specific regions, look at data in more detail (for example, sales by city within a region), or look at the data in less detail (for example, aggregate products together by category).

Another class of tools for analysts is *data mining* tools, which help them find certain kinds of patterns in data. For example, the analyst may wish to find groups of products that are often bought by a customer at the same time.

A person who has central control over the system is called a *database administrator* (DBA). The functions of a DBA include schema definition, storage structure and access-method definition, schema and physical-organization modification in response to changes in database requirements, and routine maintenance such as backing up data to guard against data loss, and ensuring availability of enough free disk space for new data.

## VIII. TRANSACTION MANAGEMENT

Often, several operations on the database form a single logical unit of work. An example is a funds transfer, as in Section III.A, in which one account (say *A*) is debited and another account (say *B*) is credited. Clearly, it is essential that either both the credit and debit occur, or that neither occur. That is, the funds transfer must happen in its entirety or not at all. This all-or-none requirement is called *atomicity*. In addition, it is essential that the execution of the funds transfer preserve the consistency of the database. That is, the value of the sum $A + B$ must be preserved. This correctness requirement is called *consistency*. Finally, after the successful execution of a funds transfer, the new values of accounts *A* and *B* must persist, despite the possibility of system failure. This persistence requirement is called *durability*.

A *transaction* is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. However, during the execution of a transaction, it may be necessary temporarily to allow inconsistency, since either the debit of *A* or the credit of *B* must be done before the other. This temporary inconsistency, although necessary, may lead to difficulty if a failure occurs.

It is the programmer's responsibility to define properly the various transactions, so that each preserves the consistency of the database. For example, the transaction to transfer funds from account *A* to account *B* could be defined to be composed of two separate programs: one that debits account *A*, and another that credits account *B*. The execution of these two programs one after the other will indeed preserve consistency. However, each program by itself does not transform the database from a consistent state to a new consistent state. Thus, those programs are not transactions.

Ensuring the atomicity and durability properties is the responsibility of the database system itself—specifically, of the *transaction-management component*. In the absence of failures, all transactions complete successfully, and atomicity is achieved easily. However, because of various types of failure, a transaction may not always complete its execution successfully. If we are to ensure the atomicity property, a failed transaction must have no effect on the state of the database. Thus, the database must be restored to the state in which it was before the transaction in question started executing. The database system must therefore perform *failure recovery*, that is, detect system failures and restore the database to the state that existed prior to the occurrence of the failure.

Finally, when several transactions update the database concurrently, the consistency of data may no longer be preserved, even though each individual transaction is correct. It is the responsibility of the *concurrency-control manager* to control the interaction among the concurrent transactions, to ensure the consistency of the database.

Database systems designed for use on small personal computers may not have all these features. For example, many small systems allow only one user to access the database at a time. Others do not offer backup and recovery, leaving that to the user. These restrictions allow for a smaller data manager, with fewer requirements for physical resources—especially main memory. Although such a low-cost, low-feature approach is adequate for small personal databases, it is inadequate for a medium- to large-scale enterprise.

## IX. DATABASE SYSTEM STRUCTURE

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

The *storage manager* is a program module responsible for storing, retrieving, and updating data in the database. It provides a physical schema level abstraction of the data. The raw data are stored on a magnetic disk or other storage device, using the file system that is usually provided by a conventional operating system. The storage manager translates data between the file-system level representation and the physical schema abstraction.

The *query processor* component uses the physical schema abstraction provided by the storage manager to process queries, updates, and DML statements submitted to the database system.

The storage manager and the query manager each have several subcomponents. Figure 4 shows these components and the connections among them.

For example, the *transaction manager* component of the storage manager ensures the transactional properties of atomicity in the event of failure, durability of

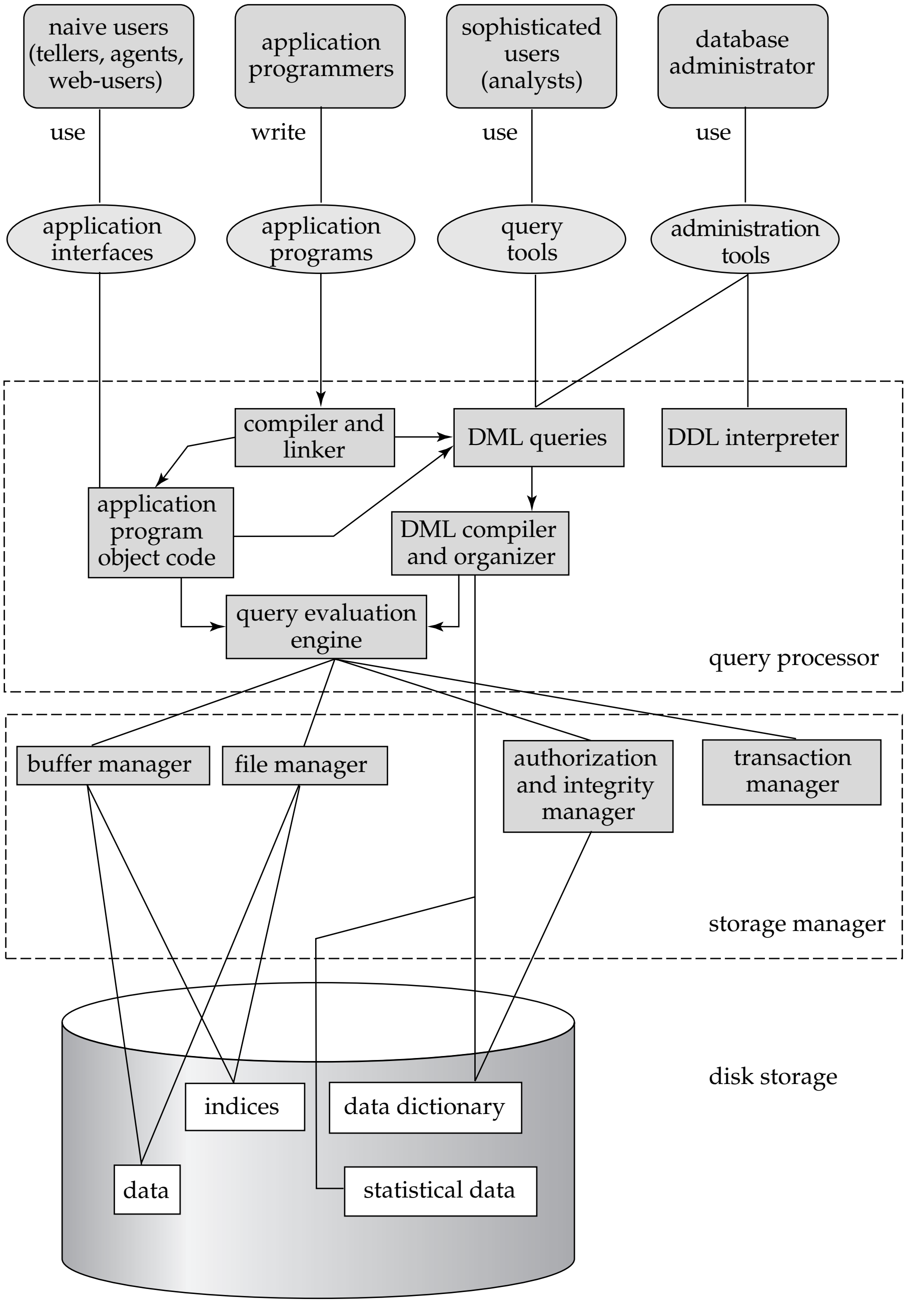


**Figure 4** System structure.

updates in the event of failure, and controls concurrency to ensure that there are no problems.

Data structures stored on disk by the storage manager include:

- Data files, which store the database itself
- Data dictionary, which stores metadata about the structure of the database, in particular the schema of the database
- Indices, which provide fast access to data items that hold particular values.

The query processor is important because it helps the database system simplify and facilitate access to data. High-level views help to achieve this goal; with them, users of the system are not burdened unnecessarily with the physical details of the implementation of the system. However, quick processing of updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The query compiler also performs *query optimization,* that is, it picks the lowest cost evaluation plan from among the alternatives. The query evaluation engine executes low-level instructions generated by the query compiler.

## SEE ALSO THE FOLLOWING ARTICLES

Database Administration • Database Development Process • Database Machines • Data Modeling: Entity-Relationship Data Model • Data Modeling: Object-Oriented Data Model • Network Database Systems • Object-Oriented Databases • Relational Database Systems • Structured Query Language (SQL) • Systems Science

## BIBLIOGRAPHY

Bernstein, P., Brodie, M., Ceri, S., DeWitt, D., Franklin, M., Garcia-Molina, H., Gary, J., Held, J., Hellerstein, J., Jagadish, H. V., Lesk, M., Maier, D., Naughton, J., Pirahesh, H., Stonebraker, M., and Ullman, J. (December 1998). The Asilomar report on database research. *ACM SIGMOD Record* **27**(4).

Bernstein, P., and Newcomer, E. (1997). *Principles of Transaction Processing*. Morgan Kaufmann.

Codd, E. F. (June 1970). A relational model for large shared data banks. *Communications of the ACM* **13**(6), 377–387.

Elmasri, R., and Navathe, S. B. (2000). *Fundamentals of Database Systems.* 3rd edition. Benjamin Cummings.

Gray, J., and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann.

O'Neil, P., and O'Neil, E. (2000). *Database: Principles, Programming, Performance.* 2nd edition. Morgan Kaufmann.

Ramakrishnan, R., and Gehrke, J. (2000). *Database Management Systems.* 2nd edition. McGraw Hill.

Stonebraker, M., and Hellerstein, J. (1998). *Readings in Database Systems.* 3rd edition. Morgan Kaufmann.

Silberschatz, A., Korth, H. F., and Sudarshan, S. (2001). *Database System Concepts.* 4th edition. McGraw Hill.

Silberschatz, A., Stonebraker, M., and Ullman, J. (1966). Database research: Achievements and opportunities into the 21st century. Technical Report CS-TR-96-1563, Department of Computer Science, Stanford University, Stanford.

Ullman, J. D., and Widom, J. (1997). *A First Course in Database Systems.* Prentice Hall.

## Web Resources

IBM DB2, commercial database system. Available at www.ibm.com/software/data.

Informix, commercial database system. Available at www.informix.com.

Microsoft SQL Server, commercial database system. Available at www.microsoft.com/sql.

MySQL, free/public domain database system. Available at www.mysql.com.

Oracle, commercial database system. Available at www.oracle.com.

Postgre SQL, free/public domain database system. Available at www.postgresql.org.

Sybase, commercial database system. Available at www.sybase.com.

# Data Compression

**Khalid Sayood**
*University of Nebraska, Lincoln*

## I. INTRODUCTION

Data compression involves the development of a compact representation of information. Most representations of information contain large amounts of redundancy. Redundancy can exist in various forms. It may exist in the form of correlation: spatially close pixels in an image are generally also close in value. The redundancy might be due to context: the number of possibilities for a particular letter in a piece of English text is drastically reduced if the previous letter is a *q*. It can be probabilistic in nature: the letter *e* is much more likely to occur in a piece of English text than the letter *q*. It can be a result of how the information-bearing sequence was generated: voiced speech has a periodic structure. Or, the redundancy can be a function of the user of the information: when looking at an image we cannot see above a certain spatial frequency; therefore, the high-frequency information is redundant for this application. Redundancy is defined by the *Merriam-Webster Dictionary* as “the part of the message that can be eliminated without the loss of essential information.” Therefore, one aspect of data compression is redundancy removal. Characterization of redundancy involves some form of modeling. Hence, this step in the compression process is also known as modeling. For historical reasons another name applied to this process is decorrelation.

After the redundancy removal process the information needs to be encoded into a binary representation. At this stage we make use of the fact that if the information is represented using a particular alphabet some letters may occur with higher probability than others. In the coding step we use shorter code words to represent letters that occur more frequently, thus lowering the average number of bits required to represent each letter.

Compression in all its forms exploits structure, or redundancy, in the data to achieve a compact representation. The design of a compression algorithm involves understanding the types of redundancy present in the data and then developing strategies for exploiting these redundancies to obtain a compact representation of the data. People have come up with many ingenious ways of characterizing and using the different types of redundancies present in different kinds of technologies from the telegraph to the cellular phone and digital movies.

One way of classifying compression schemes is by the model used to characterize the redundancy. However, more popularly, compression schemes are divided into two main groups: lossless compression and lossy compression. Lossless compression preserves all the information in the data being compressed, and the reconstruction is identical to the original data. In lossy compression some of the information contained in the original data is irretrievably lost. The loss in information is, in some sense, a payment for achieving higher levels of compression. We begin our examination of data compression schemes by first looking at lossless compression techniques.

## II. LOSSLESS COMPRESSION

Lossless compression involves finding a representation which will exactly represent the source. There should be no loss of information, and the decompressed, or reconstructed, sequence should be identical to the original sequence. The requirement that there be no loss of information puts a limit on how much compression we can get. We can get some idea about this limit by looking at some concepts from information theory.

## A. Information and Entropy

In one sense it is impossible to denote anyone as the parent of data compression: people have been finding compact ways of representing information since the dawn of recorded history. One could argue that the drawings on the cave walls are representations of a significant amount of information and therefore qualify as a form of data compression. Significantly less controversial would be the characterizing of the Morse code as a form of data compression. Samuel Morse took advantage of the fact that certain letters such as *e* and *a* occur more frequently in the English language than *q* or *z* to assign shorter code words to the more frequently occurring letters. This results in lower average transmission time per letter. The first person to put data compression on a sound theoretical footing was Claude E. Shannon (1948). He developed a quantitative notion of information that formed the basis of a mathematical representation of the communication process.

Suppose we conduct a series of independent experiments where each experiment has outcomes $A_1$, $A_2$, . . ., $A_M$. Shannon associated with each outcome a quantity called *self information,* defined as

$$i(A_k) = \log \frac{1}{P(A_k)}$$

The units of self-information are bits, nats, or Hartleys, depending on whether the base of the logarithm is 2 *e,* or 10. The average amount of information associated with the experiment is called the *entropy H:*

$$H = E[i(A)] = \sum_{k=1}^{M} i(A_k)P(A_k) = \sum_{k=1}^{M} P(A_k) \log \frac{1}{P(A_k)}$$

$$= -\sum_{k=1}^{M} P(A_k) \log P(A_k) \quad (1)$$

Suppose the "experiment" is the generation of a letter by a source, and further suppose the source generates each letter independently according to some probability model. Then $H$ as defined in Eq. (1) is called the entropy of the source. Shannon showed that if we compute the entropy in bits (use logarithm base 2) the entropy is the smallest average number of bits needed to represent the source output.

We can get an intuitive feel for this connection between the entropy of a source and the average number of bits needed to represent its output (denoted by *rate*) by performing a couple of experiments.

Suppose we have a source which puts out one of four letters $\{A_1, A_2, A_3, A_4\}$. In order to ascertain the outcome of the experiment we are allowed to ask a predetermined sequence of questions which can be answered with a *yes* or a *no.* Consider first the case of a source which puts out each letter with equal probability, that is,

$$P(A_k) = \frac{1}{4} \qquad k = 1,2,3,4$$

From Eq. (1) the entropy of this source is two bits. The sequence of questions can be represented as a flowchart as shown in Fig. 1. Notice that we need to ask two questions in order to determine the output of the source. Each answer can be represented by a single bit (1 for *yes* and 0 for *no*); therefore, we need two bits to represent each output of the source. Now consider a

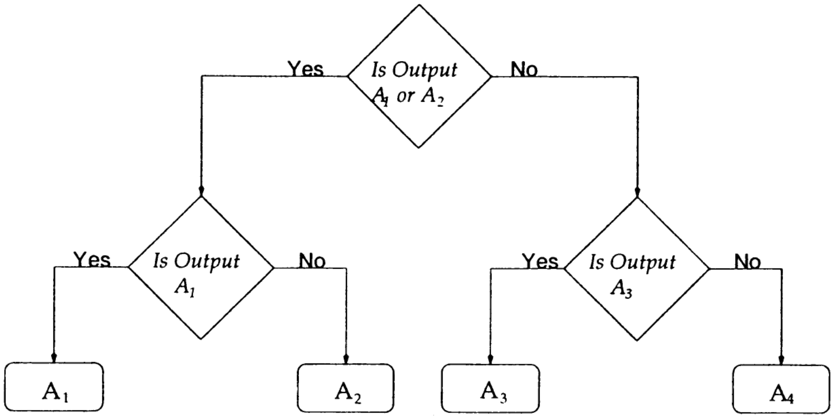


**Figure 1** Sequence of questions for equiprobable source.

slightly different situation in which the source puts out the four different letters with different probabilities:

$$P(A_1) = \frac{1}{2}, \quad P(A_2) = \frac{1}{4}, \quad P(A_3) = \frac{1}{8}, \quad P(A_4) = \frac{1}{8}$$

The entropy of this source is 1.75 bits.

Armed with the information about the source, we construct a different sequence of questions shown in Fig. 2. Notice that when the output of the source is $A_1$ we need ask only one question. Because $P(A_1) = \frac{1}{2}$ on the average this will happen half the time. If the source output is $A_2$ we need to ask two questions. This will happen a quarter of the time. If the source output is $A_3$ or $A_4$ we will need to ask three questions. This too will happen a quarter of the time. Thus, half the time we can represent the output of the source with one bit, a quarter of the time with two bits, and another quarter of the time with three bits. Therefore, on the average we will need 1.75 bits to represent the output of the source. We should note that if our information about the frequency with which the letters occur is wrong we will end up using more bits than if we had used the question sequence of Fig. 1. We can easily see this effect by switching the probabilities of $A_1$ and $A_2$ and keeping the questioning strategy of Fig. 2.

Both these examples demonstrate the link between average information, or entropy of the source, and the average number of bits, or the rate, required to represent the output of the source. They also demonstrate the need for taking into account the probabilistic structure of the source when creating a representation. Note that incorrect estimates of the probability will substantially decrease the compression efficiency of the procedure.

The creation of a binary representation for the source output, or the generation of a *code* for a source, is the topic of the next section.

## B. Coding

In the second example of the previous section the way we obtained an efficient representation was to use fewer bits to represent letters that occurred more frequently—the same idea that Samuel Morse had. It is a simple idea, but in order to use it we need an algorithm for systematically generating variable length code words for a given source. David Huffman (1951) created such an algorithm for a class project. We describe this algorithm below. Another coding algorithm which is fast gaining popularity is the arithmetic coding algorithm. We will also describe the arithmetic coding algorithm in this section.

### 1. Huffman Coding

Huffman began with two rather obvious conditions on the code and then added a third that allowed for the construction of the code. The conditions were:

1. The codes corresponding to the higher probability letters could not be longer than the code words associated with the lower probability letters.

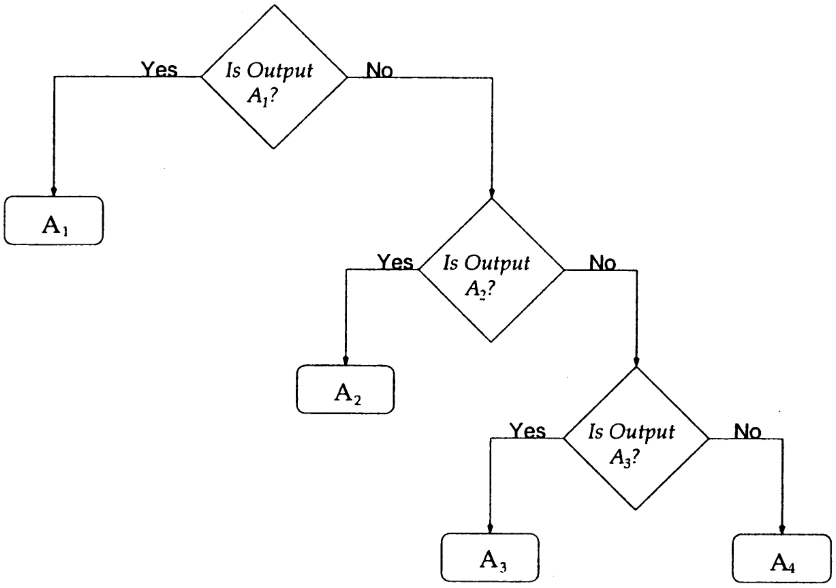


**Figure 2** Sequence of questions for second source.

2. The two lowest probability letters had to have code words of the same length.

He added a third condition to generate a practical compression scheme.

3. The two lowest probability letters have codes that are identical except for the last bit.

It is easy to visualize these conditions if we think of the code words as the path through a binary tree: a zero bit denoting a left branch and a one bit denoting a right branch. The two lowest probability letters would then be the two leaves of a node at the lowest level of the tree. We can consider the parent node of these leaves as a letter in a reduced alphabet which is obtained as a combination of the two lowest probability symbols. The probability of the letter corresponding to this node would be the sum of the probabilities of the two individual letters. Now, we can find the two lowest probability symbols of the reduced alphabet and treat them as the two leaves of a common node. We can continue in this fashion until we have completed the tree.

We can see this process best through an example.

**Example 1: Huffman Coding**

Suppose we have a source alphabet with five letters $\{a_1,a_2,a_3,a_4,a_5\}$ with probabilities of occurrence $P(a_1) = 0.15$, $P(a_2) = 0.04$, $P(a_3) = 0.26$, $P(a_4) = 0.05$, and $P(a_5) = 0.50$. We can calculate the entropy to be

$$H = -\sum_{i=1}^{5} P(a_i) \log P(a_i) = 1.817684 \text{ bits}$$

If we sort the probabilities in descending order we can see that the two letters with the lowest probabilities are $a_2$ and $a_4$. These will become the leaves on the lowest level of the binary tree. The parent node of these leaves will have a probability of 0.09. If we consider the parent node as a letter in a reduced alphabet it will be one of the two letters with the lowest probability: the other one being $a_1$. Continuing in this manner we get the binary tree shown in Fig. 3. The code is

| | |
|---|---|
| $a_1$ | 110 |
| $a_2$ | 1111 |
| $a_3$ | 10 |
| $a_4$ | 1110 |
| $a_5$ | 0 |

It can be shown that, for a sequence of independent letters, or a memoryless source, the rate of the Huffman code will always be within one bit of the entropy.

$$H \leq R \leq 1$$

In fact we can show that (Gallagher, 1978) if $p_{max}$ is the largest probability in the probability model, then

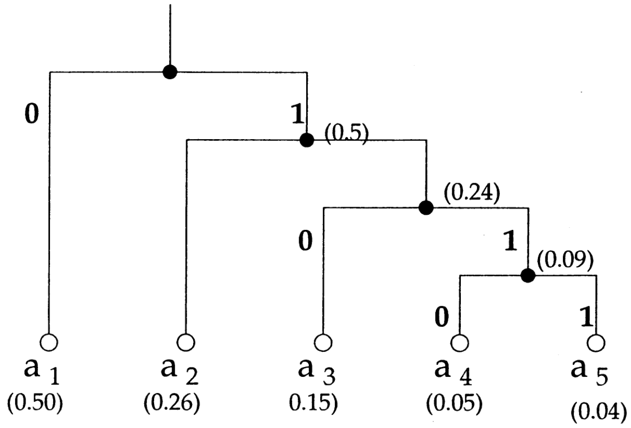


**Figure 3** Huffman code for the five letter alphabet.

for $p_{max} < 0.5$ the upper bound for the Huffman code is $H(\mathcal{S}) + p_{max}$, while for $p_{max} \geq 0.5$, the upper bound is $H(\mathcal{S}) + p_{max} + 0.086$. If instead of coding each letter separately we group the letters into blocks containing $n$ letters, then the rate is guaranteed to be even closer to the entropy of the source. Using our looser bound we can show that the bounds on the average length of the code will be

$$H \leq R \leq \frac{1}{n}$$

However, blocking letters together means an exponential increase in the size of the alphabet. Therefore, in many situations this particular approach is not very practical.

## 2. Arithmetic Coding

Practical arithmetic coding came into existence in 1976 through the work of Risannen (1976) and Pasco (1976). However, the basic ideas of arithmetic coding have their origins in the original work of Shannon (1948). (For a brief history see Sayood, 2000). Arithmetic coding relies on the fact that there are an uncountably infinite number of numbers between 0 and 1 (or any other nonzero interval on the real number line). Therefore, we can assign a unique number from the unit interval to any sequence of symbols from a finite alphabet. We can then encode the entire sequence with this single number which acts as a label or tag for this sequence. In other words, this number is a code for this sequence: a binary representation of this number is a binary code for this sequence. Because this tag is unique, in theory, given the tag the decoder can reconstruct the entire sequence. In order to implement this idea we need a mapping from the sequence to the tag and an inverse mapping from the tag to the sequence.

One particular mapping is the cumulative density function of the sequence. Let us view the sequence to be encoded as the realization of a sequence of random variables $\{X_1, X_2, \cdots\}$ and represent the set of all possible realizations which have nonzero probability of occurrence in lexicographic order by $\{\mathbf{X}_i\}$. Given a particular realization

$$\mathbf{X}_k = x_{k,1}, x_{k,2}, \cdots$$

the cumulative distribution function $F_{\mathbf{X}}(x_{k,1}, x_{k,2}, \cdots)$ is a number between 0 and 1. Furthermore, as we are only dealing with sequences with nonzero probability, this number is unique to the sequence $\mathbf{X}_k$. In fact, we can uniquely assign the half-open interval $[F_{\mathbf{X}}(\mathbf{X}_{k-1}), F_{\mathbf{X}}(\mathbf{X}_{k-1}))$ to the sequence $\mathbf{X}_k$. Any number in this interval, which we will refer to as the tag interval for $\mathbf{X}_k$, can be used as a label or tag for the sequence $\mathbf{X}_k$. The arithmetic coding algorithm is essentially the calculation of the end points of this tag interval. Before we describe the algorithm for computing these end points let us look at an example for a sequence of manageable length.

Suppose we have an *iid* sequence with letters from an alphabet $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$. The probability of occurrence of the letters are given by $p_0 = 0.3$, $p_1 = 0.1$, $p_2 = 0.2$, and $p_4 = 0.4$. Ordering the letters from the smallest to the largest index, we have $F_X(a_1) = 0.3$, $F_X(a_2) = 0.4$, $F_X(a_3) = 0.6$, and $F_X(a_4) = 1.0$. We will find the tag interval for the sequence $a_4, a_1, a_2, a_4$. We begin with a sequence that consists of a single letter $a_4$. Given that $F_X(a_4) = 1.0$ and $F_X(a_3) = 0.6$, the tag interval is $[0.6, 1.0)$. Now consider the two-letter sequence $a_4$, $a_1$. If we impose a lexicographic ordering,

$$\begin{aligned} &F_{X_1,X_2}(X_1 = a_4, X_2 = a_1) \\ &= \sum_{i=1}^{3} \sum_{j=1}^{4} Pr[X_1 = a_i, X_2 = a_j] + Pr[X_1 = a_4, X_2 = a_1] \\ &= F_X(a_3) + Pr[X_1 = a_4, X_2 = a_1] \\ &= 0.6 + 0.4 \times 0.3 = 0.72 \end{aligned}$$

The two-letter sequence prior to $a_4$, $a_1$ in the lexicographic ordering is $a_3$, $a_4$. We can compute $F_{X1,X2}(X_1 = a_3, X_2 = a_4) = 0.6$; therefore, the tag interval is $(0.6, 0.72]$. Another way of obtaining the tag interval would be to partition the single letter tag interval $[0.6, 1)$ in the same proportions as the partitioning of the unit interval. As in the case of the unit interval, the first subpartition would correspond to the letter $a_1$, the second subpartition would correspond to the letter $a_2$, and so on. As the second letter of the sequence under consideration is $a_1$, the tag interval would be the first subinterval, which is $[0.6, 0.72)$. Note that the tag interval for the two-letter sequence is wholly contained in the tag interval corresponding to the first letter. Furthermore, note that the size of the interval is $p_4 p_1 = 0.12$. Continuing in this manner for the three-letter sequence, we can compute $F_{X1,X2,X3}(a_4, a_1, a_2) = 0.672$ and $F_{X1,X2,X3}(a_4, a_1, a_1) = 0.648$. Thus, the tag interval is $(0.648, 0.672]$. Again notice that the tag interval for the three letter sequence is entirely contained within the tag interval for the two-letter sequence, and the size of the interval has been reduced to $p_4 p_1 p_2 = 0.024$. This progression is shown in Fig. 4. If we represent the upper limit of the tag interval at time $n$ by $u^{(n)}$ and the lower limit by $l^{(n)}$, we can show that

$$l^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})\, F_X(x_n - 1) \quad (2)$$

$$u^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})\, F_X(x_n). \quad (3)$$

Notice that as the sequence length increases, the tag interval is always a subset of the previous tag interval. However, the number of digits required to represent the tag intervals increases as well. This was one of the problems with the initial formulation of arithmetic coding. Another was the fact that you could not send the code until the last element of the sequence was encoded. Both these problems can be partially resolved by noting that if we use a binary alphabet, once the interval is complete in either the upper or lower half of the unit interval the most significant digits of the upper and lower limits of the tag interval are identical. Furthermore, there is no possibility of the tag interval migrating from the half of the unit interval in which it is contained to the other half of the unit interval. Therefore, we can send the most significant bit of the upper and lower intervals as the tag for the sequence. At the same time we can shift the most significant bit out of the upper and lower limits, effectively doubling the size of the tag interval. Thus, each time the tag interval is trapped in either the upper or lower half of the unit interval, we obtain one more bit of the code and we expand the interval. This way we prevent the necessity for increasing precision *as long as the tag interval resides entirely within the top or bottom half of the unit interval.* We can also start transmitting the code before the entire sequence has been encoded. We have to use a slightly more complicated strategy when the tag interval straddles the midpoint of the unit interval. We leave the details to Sayood (2000).

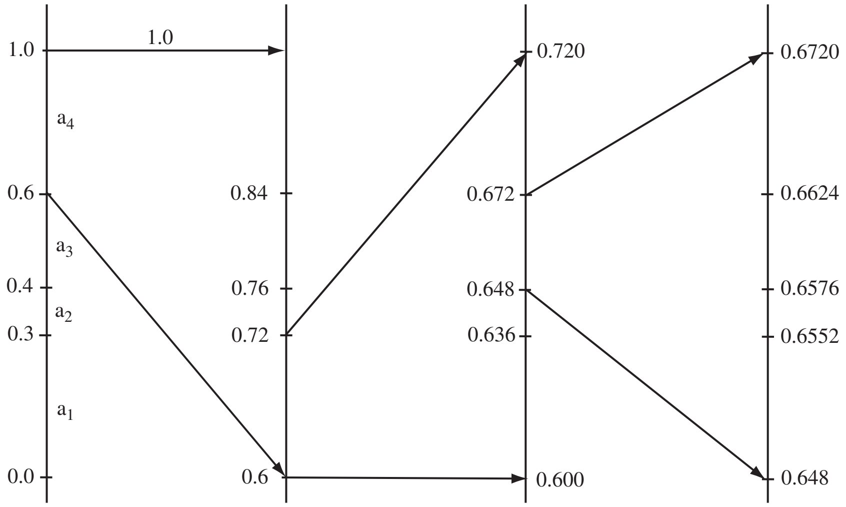


**Figure 4** Narrowing of the tag interval for the sequence $a_4a_0a_2a_4$.

## C. Sources with Memory

Throughout our discussion we have been assuming that each symbol in a sequence occurs independently of the previous sequence. In other words, there is no *memory* in the sequence. Most sequences of interest to us are not made up of independently occurring symbols. In such cases an assumption of independence would give us an incorrect estimate of the probability of occurrence of that symbol, thus leading to an inefficient code. By taking account of the dependencies in the sequence we can significantly improve the amount of compression available. Consider the letter *u* in a piece of English text. The frequency of occurrence of the letter *u* occurring in a piece of English text is approximately 0.018. If we had to encode the letter *u* and we used this value as our estimate of the probability of the letter, we would need approximately $\lfloor \log_2 \frac{1}{0.018} \rfloor = 6$ bits. However, if we knew the previous letter was *q* our estimate of the probability of the letter *u* would be substantially more than 0.018 and thus encoding *u* would require fewer bits. Another way of looking at this is by noting that if we have an accurate estimate of the probability of occurrence of particular symbols we can obtain a much more efficient code. If we know the preceding letter(s) we can obtain a more accurate estimate of the probabilities of occurrence of the letters that follow. Similarly, if we look at pixel values in a natural image and assume that the pixels occur independently, then our estimate of the probability of the values that each pixel could take on would be approximately the same (and small). If we took into account the values of the neighboring pixels, the probability that the pixel under consideration would have a similar value would be quite high. If, in fact, the pixel had a value close to its neighbors, encoding it in this context would require many fewer bits than encoding it independent of its neighbors (Memon and Sayood, 1995).

Shannon (1948) incorporated this dependence in the definition of entropy in the following manner. Define

$$G_n = -\sum_{i_1=m}^{i_1=m} \sum_{i_2=m}^{i_2=m} \cdots \sum_{i_n=1}^{i_n=m} P(X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n) \log P(X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n)$$

where $\{X_1, X_2, \ldots, X_n\}$ is a sequence of length $n$ generated by a source $\mathcal{S}$. Then the entropy of the source is defined as

$$H(\mathcal{S}) = \lim_{n\to\infty} \frac{1}{n} G_n$$

There are a large number of compression schemes that make use of this dependence.

## D. Context-Based Coding

The most direct approach to using this dependence is to code each symbol based on the probabilities provided by the context. If we are to sequentially encode the sequence $x_1, x_2, \ldots, x_n$, we need $-\log \Pi_{i=0}^{n-1} p(x_{i+1} | x_1, x_2, \ldots, x_n)$ bits (Weinberger *et al.*, 1995). The history of the sequence makes up its context. In practice, if we had these probabilities available they could be used by an arithmetic coder to code the particular symbol. To use the entire history of the sequence as the context is generally not feasible. Therefore, the context is made up of some subset of the history. It generally consists of those symbols that our knowledge of the source tells us will have an affect on the probability of the symbol being encoded and which are available to both the encoder and the decoder. For example, in the binary image coding standard JBIG, the context may consist of the pixels in the immediate neighborhood of the pixel being encoded along with a pixel some distance away which reflects the periodicity in half-tone images. Some of the most popular context-based schemes today are *ppm* (prediction with partial match) schemes.

### 1. Prediction with Partial Match

In general, the larger the size of a context, the higher the probability that we can accurately predict the symbol to be encoded. Suppose we are encoding the fourth letter of the sequence *their*. The probability of the letter *i* is approximately 0.053. It is not substantially changed when we use the single letter context *e,* as *e* is often followed by most letters of the alphabet. In just this paragraph it has been followed by *d, i, l, n, q, r, t,* and *x*. If we increase the context to two letters *he,* the probability of *i* following *he* increases. It increases even more when we go to the three-letter context *the*. Thus, the larger the context, the better off we usually are. However, all the contexts and the probabilities associated with the contexts have to be available to both the encoder and the decoder. The number of contexts increases exponentially with the size of the context. This puts a limit on the size of the context we can use. Furthermore, there is the matter of obtaining the probabilities relative to the contexts. The most efficient approach is to obtain the frequency of occurrence in each context from the past history of the sequence. If the history is not very large, or the symbol is an infrequent one, it is quite possible that the symbol to be encoded has not occurred in this particular context.

The *ppm* algorithm initially proposed by Cleary and Witten (1984) starts out by using a large context of predetermined size. If the symbol to be encoded has not occurred in this context, an escape symbol is sent and the size of the context is reduced. For example, when encoding *i* in the example above we could start out with a context size of three and use the context *the*. If *i* has not appeared in this context, we would send an escape symbol and use the second order context *he*. If *i* has not appeared in this context either, we reduce the context size to one and look to see how often *i* has occurred in the context of *e*. If this is zero, we again send an escape and use the zero order context. The zero order context is simply the frequency of occurrence of *i* in the history. If *i* has never occurred before, we send another escape symbol and encode *i,* assuming an equally likely occurrence of each letter of the alphabet.

We can see that if the symbol has occurred in the longest context, it will likely have a high probability and will require few bits. However, if it has not, we pay a price in the shape of the escape symbol. Thus, there is a tradeoff here. Longer contexts may mean higher probabilities or a long sequence of escape symbols. One way out of this has been proposed by Cleary and Teahan (1997) under the name *ppm**. In *ppm** we look at the longest context that has appeared in the history. It is very likely that this context will be followed by the symbol we are attempting to encode. If so, we encode the symbol with very few bits. If not, we drop to a relatively short context length and continue with the standard *ppm* algorithm. There are a number of variants of the *ppm** algorithm, with the most popular being *ppmz* developed by Bloom. It can be found at *http://www.cbloom.com/src/ppmz.html*.

## E. Predictive Coding

If the data we are attempting to compress consists of numerical values, such as images, using context-based approaches directly can be problematic. There are several reasons for this. Most context-based schemes exploit exact reoccurrence of patterns. Images are usually acquired using sensors that have a small amount of noise. While this noise may not be perceptible, it is sufficient to reduce the occurrence of exact repetitions of patterns. A simple alternative to using the context approach is to generate a prediction for the value to be encoded and encode the prediction error. If there is considerable dependence among the values with high probability, the prediction will be close to the actual value and the prediction error will be a small number. We can encode this small number, which as it occurs with high probability will require fewer bits to encode. An example of this is shown in Fig. 5. The plot on the left of Fig. 5 is the histogram of the pixel values of the *sensin* image, while the plot on the right is the histogram of the differences between neighboring pixels. We can see that the small differences occur much more frequently and therefore can be encoded using fewer bits. As

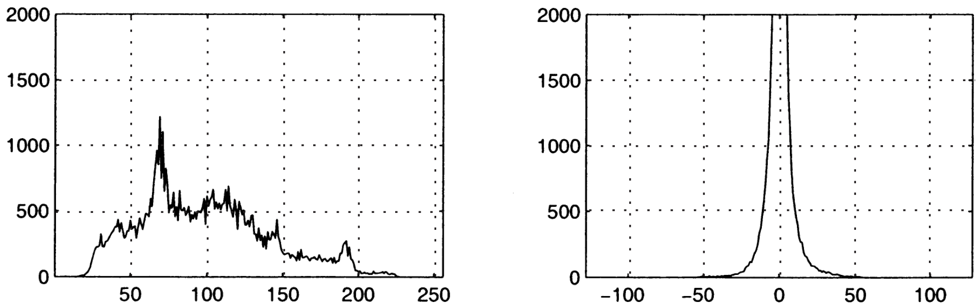


**Figure 5** Histograms of pixel values in the *sensin* image and of the differences between neighboring pixels.

opposed to this, the actual pixel values have a much more uniform distribution.

Because of the strong correlation between pixels in a neighborhood, predictive coding has been highly effective for image compression. It is used in the current state-of-the-art algorithm CALIC (Wu and Memon, 1996) and forms the basis of JPEG-LS, which is the standard for lossless image compression.

## 1. JPEG-LS

The JPEG-LS standard uses a two-stage prediction scheme followed by a context-based coder to encode the difference between the pixel value and the prediction. For a given pixel the prediction is generated in the following manner. Suppose we have a pixel with four neighboring pixels as shown in Fig. 6. The initial prediction *X* is obtained as

```
if $NW \geq max(W,N)$
$X = max(W,N)$
else
{
    if $NW \leq min(W,N)$
    $X = min(W,N)$
    else
    $X = W + N - NW$
}
```

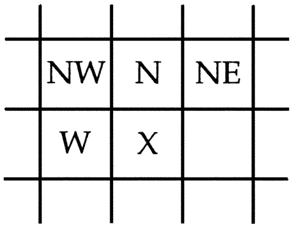


**Figure 6** Labeling of pixels in the neighborhood of a pixel to be encoded.

This prediction is then refined by using an estimate of how much the prediction differed from the actual value in similar situations in the past. The "situation" is characterized by an activity measure which is obtained using the differences of the pixels in the neighborhood. The differences *NE* − *N*, *N* − *NW*, and *NW* − *W* are compared against thresholds which can be defined by the user and a number between 0 and 364 and a *SIGN* parameter which takes on the values +1 and −1. The sign parameter is used to decide whether the correction should be added or subtracted from the original prediction. The difference between the pixel value and its prediction is mapped into the range of values occupied by the pixel and encoded using Golomb codes. For details, see Sayood (2000) and Weinberger *et al.* (1998).

## 2. Burrows-Wheeler Transform

The Burrows-Wheeler transform (BWT) uses the memory in a sequence in a somewhat different manner than either of the two techniques described previously. In this technique the entire sequence to be encoded is read in and all possible cyclic shifts of the sequence are generated. These are then sorted in lexicographic order. The last letter of each sorted cyclically shifted sequence is then transmitted along with the location of the original sequence in the sorted list. The sorting results in long sequences of identical letters in the transmitted sequence. This structure can be used to provide a very efficient representation of the transmitted sequence. The easiest way to understand the BWT algorithm is through an example.

### Example: BWT

Suppose we wish to encode the sequence *RINTINTIN.* We first obtain all cyclical shifts of this sequence and then sort them in lexicographic order as shown in Fig. 7.

We transmit the string consisting of the last letters in the lexicographically ordered set and the position

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | I | N | T | I | N | T | I | N | I | N | R | I | N | T | I | N | T |
| N | R | I | N | T | I | N | T | I | I | N | T | I | N | R | I | N | T |
| I | N | R | I | N | T | I | N | T | I | N | T | I | N | T | I | N | R |
| T | I | N | R | I | N | T | I | N | N | R | I | N | T | I | N | T | I |
| N | T | I | N | R | I | N | T | I | N | T | I | N | R | I | N | T | I |
| I | N | T | I | N | R | I | N | T | N | T | I | N | T | I | N | R | I |
| T | I | N | T | I | N | R | I | N | R | I | N | T | I | N | R | I | N |
| N | T | I | N | T | I | N | R | I | T | I | N | R | I | N | T | I | N |
| I | N | T | I | N | T | I | N | R | T | I | N | T | I | N | R | I | N |

**Figure 7** Cyclically shifted versions of the original sequence and lexicographically order set of the cyclical shifts.

of the original string in the lexicographically ordered set. For this example the transmitted sequence would be the string *TTRIIINNN* and the index 7. Notice that the string to be transmitted contains relatively long strings of the same letter. If our example string had been realistically long, the runs of identical letters would have been correspondingly long. Such a string is easy to compress. The method of choice for BWT is move-to-front coding (Burrows and Wheeler, 1994; Nelson, 1996; Sayood, 2000).

Once the string and the index have been received, we decode them by working backwards. We know the last letter in the string is its seventh element which is *N*. To find the letter before *N* we need to generate the string containing the first letters of the lexicographically ordered set. This can be easily obtained as *IIINNNRTT* by lexicographically ordering the received sequence. We will refer to this sequence of first letters as the $\mathcal{F}$ sequence and the sequence of last letters as the $\mathcal{L}$ sequence. Note that given a particular string and its cyclic shift, the last letter of the string becomes the first letter of the cyclically shifted version, and the last letter of the cyclically shifted version is the letter prior to the last letter in the original sequence. Armed with this fact and the knowledge of where the original sequence was in the lexicographically ordered set, we can decode the received sequence. We know the original sequence was seventh in the lexicographically ordered set; therefore, the last letter of the sequence has to be *N*. This is the first *N* in $\mathcal{L}$. Looking in $\mathcal{F}$ we see that the first *N* appears in the fourth location. The fourth element in $\mathcal{L}$ is *I*. Therefore, the letter preceding *N* in our decoded sequence is *I*. This *I* is the first *I* in $\mathcal{L}$; therefore, we look for the first *I* in $\mathcal{F}$. The first *I* occurs in the first location. The letter in the first location in $\mathcal{L}$ is *T*. Therefore, the decoded sequence becomes *TIN*. Continuing in this fashion we can decode the entire sequence.

The BWT is the basis for the compression utilities *bzip* and *bzip2*. For more details on BWT compression, see Burrows and Wheeler (1994), Nelson and Gailly (1996), and Sayood (2000).

## F. Dictionary Coding

One of the earliest forms of compression is to create a dictionary of commonly occurring patterns which is available to both the encoder and the decoder. When this pattern occurs in a sequence to be encoded it can be replaced by the index of its entry in the dictionary. A problem with this approach is that a dictionary that works well for one sequence may not work well for another sequence. In 1977 and 1978, Jacob Ziv and Abraham Lempel (1977, 1978) described two different ways of making the dictionary adapt to the sequence being encoded. These two approaches form the basis for many of the popular compression programs of today.

### 1. LZ77

The 1977 algorithm commonly referred to as LZ77 uses the past of the sequence as the dictionary. Whenever a pattern recurs within a predetermined window, it is replaced by a pointer to the beginning of its previous occurrence and the length of the pattern. Consider the following (admittedly weird) sequence:

*a mild fork for miles vorkosigan*

We repeat this sentence in Fig. 8 with recurrences replaced by pointers. If there were sufficient recurrences of long patterns, we can see how this might result in a compressed representation. In Fig. 8 we can see that the "compressed" representation consists of both pointers and fragments of text that have not been encountered previously. We need to inform the decoder when a group of bits is to be interpreted as a pointer and when it is to be interpreted as text. There are a number of different ways in which this can be done. The original LZ77 algorithm used a sequence of triples <*o,l,c*> to encode the source output, where *o* is the offset or distance to the previous recurrence, *l* is the length of the pattern, and *c* corresponds to the character following the recurrence. In this approach when a symbol was encountered for the first time (in the encoding window) the values for *o* and *l* were set to 0. Storer and Syzmanski (1982) suggested using a one bit flag to differentiate between pointers and symbols which had not occurred previously in the coding

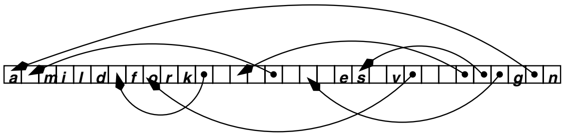


**Figure 8** Illustration for LZ77.

window. In their variant, commonly known as LZSS, the one bit flag is followed by either a pair (<*o,l*>) or the code word of the new symbol. There are a number of variants of both the LZ77 and the LZSS algorithms which are the basis for several popularly used compression utilities. Included among them are *gzip* and *PNG* (portable network graphics).

### 2. LZ78

The 1978 algorithm requires actually building a dynamic dictionary based on the past symbols in the sequence. The sequence is encoded using pairs of code words <*i,c*>, where *i* is the index to a pattern in the dictionary and *c* is the code word of the character following the current symbol. The most popular variant of the LZ78 algorithm is the LZW algorithm developed by Terry Welch (1984). In this variation the dictionary initially contains all the individual letters of the source alphabet. The encoder operates on the sequence by collecting the symbols in the sequence into a pattern $p$ until such time as the addition of another symbol $\alpha$ will result in a pattern which is not contained in the dictionary. The index to $p$ is then transmitted, the pattern $p$ concatenated with $\alpha$ is added as the next entry in the dictionary, and a new pattern $p$ is begun with $\alpha$ as the first symbol.

Variants of the LZW algorithm are used in the UNIX *compress* command as part of the graphic interchange format (GIF) and for the modem protocol v. 42bis.

## III. LOSSY COMPRESSION

The requirement that no information be lost in the compression process puts a limit on the amount of compression we can obtain. The lowest number of bits per sample is the entropy of the source. This is a quantity over which we generally have no control. In many applications this requirement of no loss is excessive. For example, there is high-frequency information in an image which cannot be perceived by the human visual system. It makes no sense to preserve this information for images that are destined for human consumption. Similarly, when we listen to sampled speech we cannot perceive the exact numerical value of each sample. Therefore, it makes no sense to expend coding resources to preserve the exact value of each speech sample. In short, there are numerous applications in which the preservation of all information present in the source output is not necessary. For these applications we relax the requirement that the reconstructed signal be identical to the original. This allows us to create compression schemes that can provide a much higher level of compression. However, it should be kept in mind that we generally pay for higher compression by increased information loss. Therefore, we measure the performance of the compression system using two metrics. We measure the amount of compression as before; however, we also measure the amount of distortion introduced by the loss of information. The measure of distortion is generally some variant of the mean squared error. If at all possible, it is more useful to let the application define the distortion.

In this section we describe a number of compression techniques that allow loss of information, hence the name lossy compression. We begin with a look at quantization which, in one way or another, is at the heart of all lossy compression schemes.

## A. Quantization

Quantization is the process of representing the output of a source with a large (possibly infinite) alphabet with a small alphabet. It is a many-to-one mapping and therefore irreversible. Quantization can be performed on a sample-by-sample basis or it can be performed on a group of samples. The former is called scalar quantization and the latter is called vector quantization. We look at each in turn.

### 1. Scalar Quantization

Let us, for the moment, assume that the output alphabet of the source is all or some portion of the real number line. Thus, the size of the source alphabet is infinite. We would like to represent the source output using a finite number of code words $M$. The quantizer consists of two processes: an encoding process that maps the output of the source into one of the $M$ code words, and a decoding process that maps each code word into a reconstruction value. The encoding process can be viewed as a partition of the source alphabet, while the decoding process consists of obtaining representation values for each partition. An example for a quantizer with an alphabet size of four is shown in Fig. 9. If the quantizer output alphabet includes 0, the quantizer is called a *midtread* quantizer. Otherwise, it is called a *midrise* quantizer.

The simplest case is when the partitions are of the same size. Quantizers with same size partitions are called *uniform quantizers.* If the source is uniformly distributed between $-A$ and $A$, the size of the partition or quantization interval $\Delta$ is $2A/M$. The reconstruction values are located in the middle of each quanti-

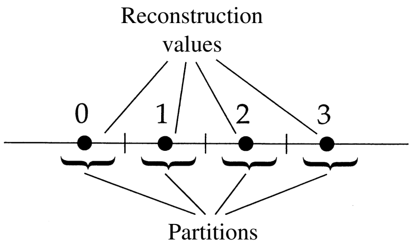


**Figure 9** Quantizer with alphabet size of four.

zation interval. If we define the difference between the input $x$ and the output $Q(x)$ to be the quantization noise, we can show that the variance of the quantization noise in this situation is $\Delta^2/12$. If the distribution of the source output is other than uniform, we can optimize the value of $\Delta$ for the particular distribution (Gersho and Gray, 1991; Max, 1960).

Often, the distribution of the quantizer input is a peaked distribution modeled as a Gaussian or Laplacian distribution. If the encoder is going to use a fixed length code, that is, each quantizer output is encoded using the same number of bits, we can get a lower average distortion if we use smaller partitions corresponding to the lower value inputs. Such a quantizer is called a *nonuniform quantizer* and is specified by the boundary values of the partition $b_i$ and the reconstruction levels $y_i$. If we know the probability density function $f_X(x)$, the boundary and reconstruction values for an $M$-level quantizer which minimizes the mean squared error can be obtained by iteratively solving the following equations.

$$y_j = \frac{\int_{bj-1}^{bj} x f_X(x)\, dx\}}{\int_{b_{j-1}}^{b_j} fX(x)\, dx} \tag{4}$$

$$b_j = \frac{\mathrm{y}_{j+1} + \mathrm{y}_j}{2} \tag{5}$$

Rather than changing the size of the quantization intervals, we can also implement a nonuniform quantizer as shown in Fig. 10. The high-probability input region is "spread out" so as to make use of multiple quantization intervals. The mapping is reversed after the quantizer. For a companding function $c(x)$ and a source which lies between $\pm x_{max}$, the variance of the quantization noise is

$$\sigma_q^2 = \frac{x_{max}^2}{3M^2} \int_{-x_{max}}^{x_{max}} \frac{f_X(x)}{(c'(x))^2}\, dx. \tag{6}$$

If a variable length code such as a Huffman code or arithmetic coding is used to encode the output of the quantizer, Gish and Pierce (1968) showed that the optimum quantizer is a uniform quantizer which covers the entire range of the source output. If the range is large and our desired distortion is small, the number of quantizer levels can become quite large. In these situations we can use a quantizer with a limited output alphabet called a *recursively indexed quantizer* (Sayood and Na, 1992).

The JPEG algorithm uses a set of uniform scalar quantizers for quantizing the coefficients used to represent the image. The quantizer levels are then encoded using a variable length code.

## 2. Vector Quantization

The idea of representing groups of samples rather than individual samples has been present since Shannon's original papers (1948). There are several advantages to representing sequences. Consider the samples of the signal shown in Fig. 11. The values vary approximately between −4 and 4. We could quantize these samples with an eight-level scalar quantizer with $\Delta = 1$. So the reconstruction values would be $\{\pm\frac{1}{2}, \pm\frac{3}{2}, \pm\frac{5}{2}, \pm\frac{7}{2}\}$. If we were to use a fixed length code we would need three bits to represent the eight quantizer outputs. If we look at the output of the quantizer in pairs, we get 64 possible reconstruction values. These are represented by the larger filled circles in Fig. 12. However, if we plot the samples of the signal as pairs (as in Fig. 13) we see that the samples are clustered along a line in the first and third quadrants. This is due to the fact that there is a high degree of correlation between neighboring samples, which means that in two dimensions the samples will cluster

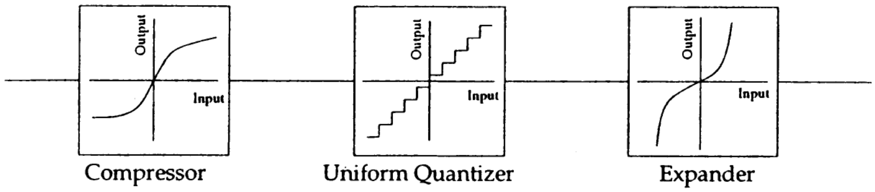


**Figure 10** Companded quantization.

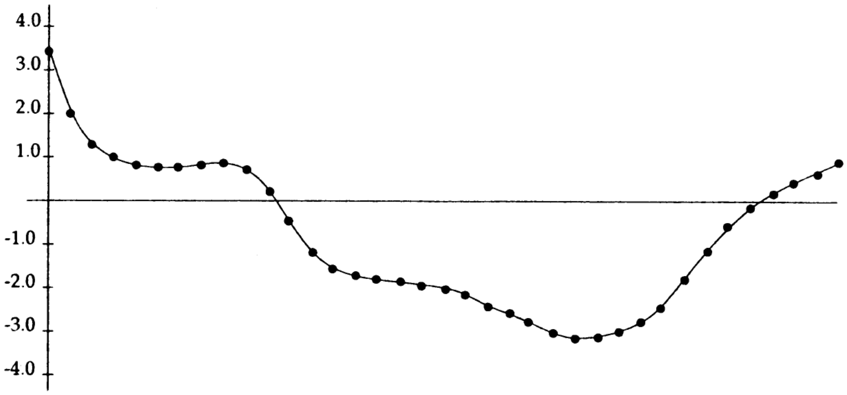


**Figure 11** Samples of a signal.

around the $y = x$ line. Looking from a two-dimensional point of view, it makes much more sense to place all the 64 output points of the quantizer close to the $y = x$ line. For a fixed length encoding, we would need six bits to represent the 64 different quantizer outputs. As each quantizer output is a representation of two samples, we would end up with three bits per sample. Therefore, for the same number of bits, we would get a more accurate representation of the input and, therefore, incur less distortion. We pay for this decrease in distortion in several different ways. The first is through an increase in the complexity of the encoder. The scalar quantizer has a very simple encoder. In the case of the two-dimensional quantizer, we need to block the input samples into "vectors" and then compare them against all the possible quantizer output values. For three bits per sample and two dimensions this translates to 64 possible compares. However, for the same number of bits and a block size, or vector dimension, of 10, the number of quantizer outputs would be $2^{3\times 10}$ which is 1,073,741,824! As it generally requires a large block size to get the full advantage of a vector quantizer, this means that the rate at which a vector quantizer (VQ) operates (i.e., bits per sample) is usually quite low.

The second way we pay is because of the fact that the quantizer becomes tuned to our assumptions about the source. For example, if we put all our quan-

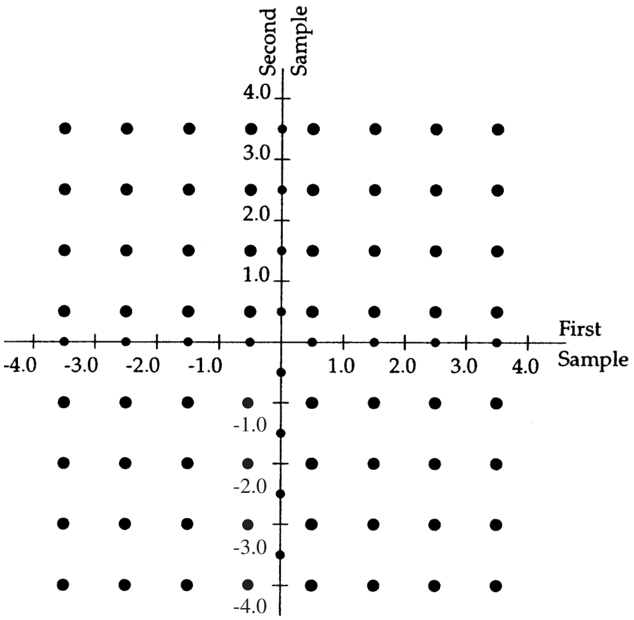


**Figure 12** Two-dimensional view of an eight-level scalar quantizer.

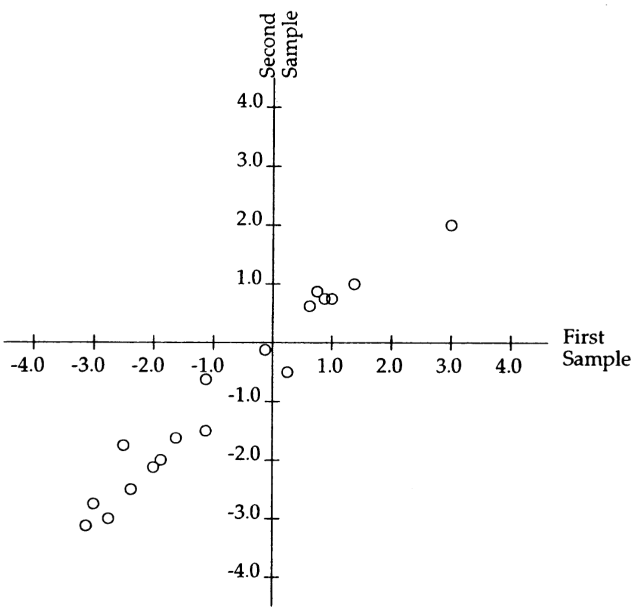


**Figure 13** Two-dimensional view of an eight-level scalar quantizer.

tizer output values along the $y = x$ line and then we started getting input vectors which lay in the second or fourth quadrants, we would end up with substantial distortion in the reconstruction.

The operation of a VQ can be summarized as follows (see Fig. 14). Both the encoder and the decoder have copies of the VQ codebook. The codebook consists of the quantizer reconstruction vectors. The encoder blocks the input into an $N$ sample vector and finds the vector in the codebook which is closest (usually in the Euclidean sense) to the input. The encoder then sends the index of the closest match. The decoder, upon receiving the index, performs a table lookup and obtains the reconstruction vector.

The VQ codebook is usually obtained using a clustering algorithm popularized by Linde, Buzo, and Gray (1980). It is generally referred to by the initials of the three authors (LBG). The LBG algorithm obtains a nearest neighbor partition of the source output space by making use of a training set. Selection of the training set can be an important aspect of the design of the VQ as it embodies our assumptions about the source output. Details of the LBG algorithm can be found in a number of places (see Gersho and Gray, 1991; Linde, Buzo, and Gray, 1980; Sayood, 2000).

There are a number of variations on the basic VQ described above. The most well known is the tree structured vector quantizer (TSVQ). Details on these can be found in Gersho and Gray (1991).

## B. Predictive Coding

If we have a sequence with sample values that vary slowly as in the signal shown in Fig. 11, knowledge of the previous samples gives us a lot of information about the current sample. This knowledge can be used in a number of different ways. One of the earliest attempts at exploiting this redundancy was in the development of differential pulse code modulation (DPCM) (Cutler, 1952). A version of DPCM is the algorithm used in the International Telecommunication Union (ITU) standard G.726 for speech coding.

The DPCM system consists of two blocks as shown in Fig. 15. The function of the predictor is to obtain an estimate of the current sample based on the *reconstructed* values of the past sample. The difference between this estimate, or prediction, and the actual value is quantized, encoded, and transmitted to the receiver. The decoder generates an estimate identical to the encoder, which is then added on to generate the reconstructed value. The requirement that the prediction algorithm use only the reconstructed values is to ensure that the prediction at both the encoder and the decoder are identical. The reconstructed values used by the predictor, and the prediction algorithm, are dependent on the nature of the data being encoded. For example, for speech coding the predictor often uses the immediate past several values of the sequence, along with a sample that is a pitch period away, to form the prediction. In image compression the predictor may use the same set of pixels used by the JPEG-LS algorithm to form the prediction.

The predictor generally used in a DPCM scheme is a linear predictor. That is, the predicted value is obtained as a weighted sum of past reconstructed values.

$$p_n = \sum_{i \in \mathcal{I}} a_i x_i$$

where $\mathcal{I}$ is an index set corresponding to the samples to be used for prediction. The coefficients $a_i$ are generally referred to as the predictor coefficients. If we assume the source output to be wide sense stationary, the predictor coefficients can be obtained as a solution of the discrete form of the Weiner-Hopf equations

$$A = R^{-1}P$$

where $A$ is an $M \times 1$ vector of predictor coefficients, $R$ is the $M \times M$ autocorrelation matrix of the set of

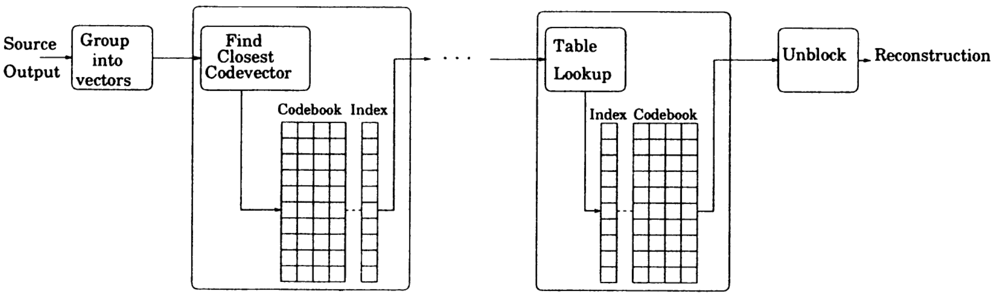


**Figure 14** Operation of a vector quantizer.

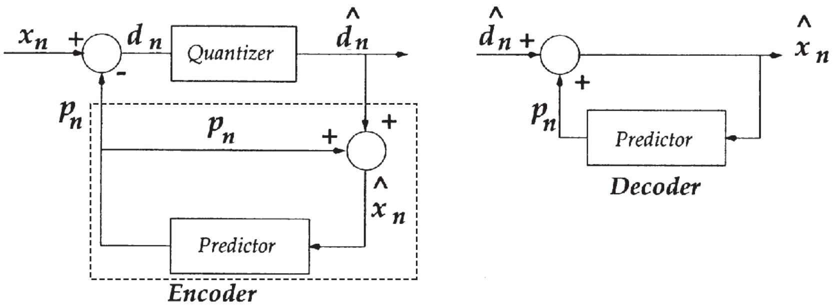


**Figure 15** Block diagram of a DPCM system.

samples used to form the prediction, and $P$ is the $M \times 1$ vector of autocorrelation coefficients between the elements of the index set and the value to be estimated.

Both the quantizer and the predictor in the DPCM system can be adaptive. The most common form of adaptivity for DPCM in speech coding is based on the reconstruction values. This allows both the encoder and the decoder, in the absence of channel errors, to adapt using the same information.

In Fig. 16 we show the results of encoding the *sensin* image using a simple fixed predictor and a recursively indexed quantizer with entropy coding.

## C. Transform Coding

Transform coding first became popular in the early 1970s as a way of performing vector quantization (Huang and Schultheiss, 1963). Transform coding consists of three steps. The data to be compressed is divided into blocks, and the data in each block is transformed to a set of coefficients. The transform is selected to compact most of the energy into as few coefficients as possible. The coefficients are then quantized with different quantizers for each coefficient. Finally, the quantizer labels are encoded.

Transform coding generally involves linear transforms. Consider the situation where the data is blocked into vectors of length $M$. The transform coefficients can be obtained by multiplying the data with a transform matrix of dimension $M \times M$.

$$\theta = AX$$

where $X$ is a vector of size $M \times 1$ containing the data, and $\theta$ is the $M \times 1$ vector of transform coefficients. The data can be recovered by taking the inverse transform:

$$X = A^{-1}\theta$$

In most transforms of interest the transform matrix is unitary, that is,

$$A^{-1} = A^T$$

If the data is stationary we can show that the optimum transform in terms of providing the most energy com-

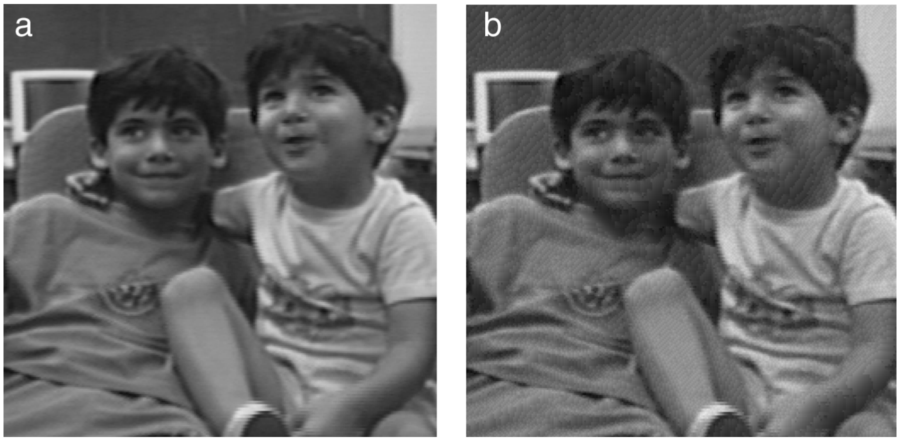


**Figure 16** (a) The original *sensin* image and (b) the *sensin* image coded at one bit per pixel using DPCM and the recursively indexed quantizer.

paction is the Karhunen-Loeve transform (KLT). The KLT transform matrix is constructed from the eigenvectors of the autocorrelation matrix. Therefore, the transform is data dependent. Different sets of data might require different transform matrices. In practice, unless the size of the transform is small relative to the data, it generally costs too many bits to send the transform matrix to the decoder for the KLT to be feasible. The practical alternative to the KLT has been the discrete cosine transform (DCT). The DCT provides comparable compression to the KLT for most sources and has a fixed transform matrix whose elements are given by:

$$[\mathbf{A}]_{i,j} = \begin{cases} \sqrt{\frac{1}{M}} \cos \frac{(2j+1)i\pi}{2M} & i = 0,\ j = 0,1,\cdots, M-1 \\ \sqrt{\frac{2}{M}} \cos \frac{(2j+1)i\pi\}}{2M} & i = 1,2,\cdots, M-1, \\ & \qquad j = 0,1,\cdots, M-1 \end{cases} \tag{7}$$

The rows of the transform matrix are shown in graphical form in Fig. 17. We can see that the rows represent signals of increasing frequency. Thus, the DCT breaks the signal up into its frequency components. As most natural sources, such as speech, have higher low-frequency content the lower order coefficients usually contain most of the information about a particular block. We obtain compression by discarding those coefficients which contain little or no energy.

When coding two-dimensional sources such as images, most transform coding schemes use separable transforms. These are transforms which can be implemented by first taking the transform of the rows and then taking the transform of the columns (or vice versa). The popular JPEG image compression standard uses a separable 8 × 8 DCT as its transform. The image is divided into 8 × 8 blocks. These blocks are then transformed to obtain 8 × 8 blocks of coefficients.

The coefficients have different statistical characteristics and may be of differing importance to the end user. Therefore, they are quantized using different quantizers. In image compression the lower order coefficients are more important in terms of human perception and are therefore quantized using quantizers with smaller step sizes (and hence less quantization noise) than the higher order, higher frequency coefficients. A sample set of step sizes recommended by the JPEG committee (Pennebaker and Mitchell, 1993) is shown in Fig. 18.

Each quantized value $Q(\theta_{i,j})$ is represented by a label

$$l_{i,j} = \left\lfloor \frac{\theta_{ij}}{Q_{ij}} + .5 \right\rfloor$$

where $Q_{ij}$ is the step size in the $i$th row and $j$th column and $\lfloor\ \rfloor$ indicates truncation to an integer value. The reconstruction is obtained from the label by multiplying it with the step size.

The final step involves coding the quantization levels. The JPEG image compression standard uses a

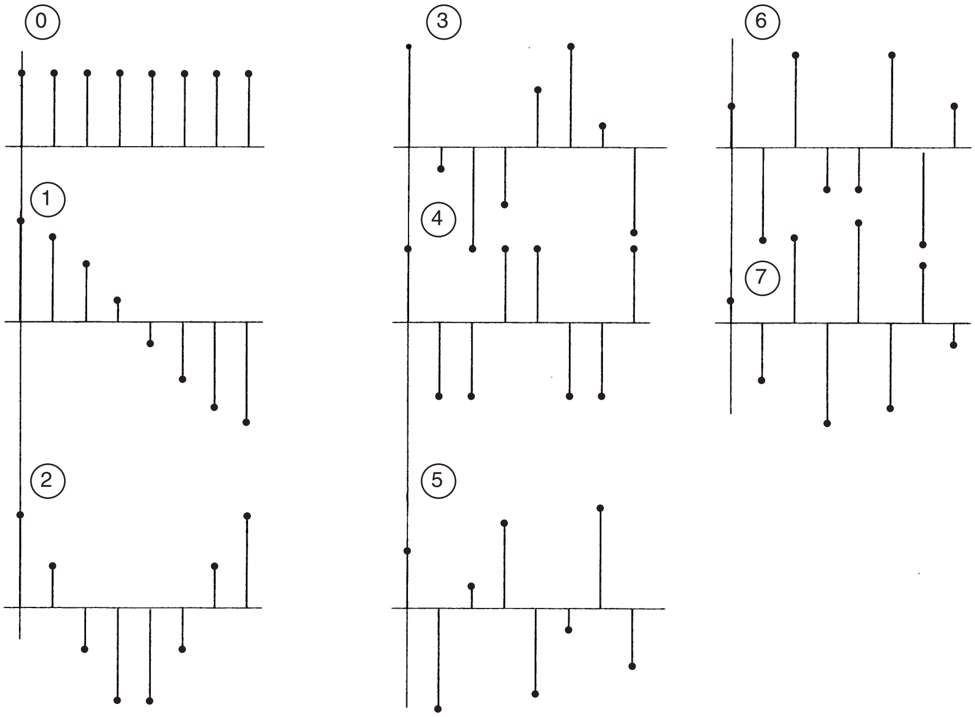


**Figure 17** Basis set for the DCT. The numbers in the circles correspond to the row of the transform matrix.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 11 | 10 | 16 | 24 | 40 | 51 | 61 |
| 12 | 12 | 14 | 19 | 26 | 58 | 60 | 55 |
| 14 | 13 | 16 | 24 | 40 | 57 | 69 | 56 |
| 14 | 17 | 22 | 29 | 51 | 87 | 80 | 62 |
| 18 | 22 | 37 | 56 | 68 | 109 | 103 | 77 |
| 24 | 35 | 55 | 64 | 81 | 104 | 113 | 92 |
| 49 | 64 | 78 | 87 | 103 | 121 | 120 | 101 |
| 72 | 92 | 95 | 98 | 112 | 100 | 103 | 99 |

**Figure 18** Sample quantization table.

minor variation of a scheme by Chen and Pratt (1984). The lowest order coefficient, commonly referred to as the DC coefficient, has been found to be correlated from block to block. It is therefore coded differentially. The remaining coefficients are scanned in zigzag order as shown in Fig. 19. As most of the higher frequency coefficients are generally small and the step sizes used to quantize them are relatively large, most of the labels at the tail end of the scan are zero. We take advantage of this fact by sending an end of block symbol after the last nonzero label on the scan. Thus, a large number of coefficients are represented using a single code word. We represent the remaining coefficients using a code which is indexed by the magnitude of the coefficient and the number of zero valued labels preceding it. Details of the coding can be found in a number of places including Pennebaker and Mitchell (1993) and Sayood (2000).

In Fig. 20 we show the *sensin* image coded at 0.5 bits per pixel using JPEG.

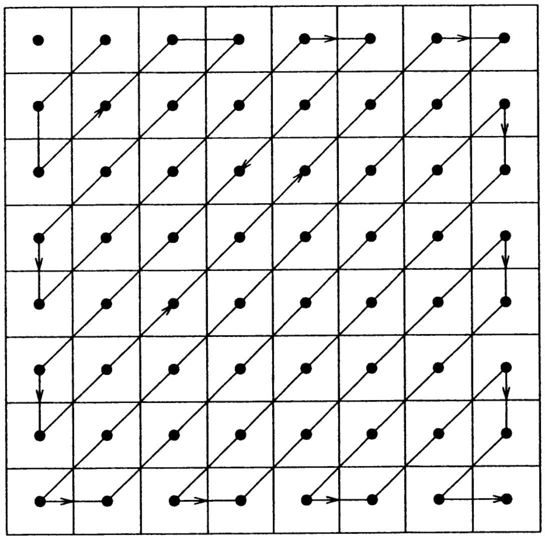


**Figure 19** The zigzag scanning pattern for an 8 × 8 transform.

## D. Subband/Wavelet Coding

Transform coding at low rates tends to give the reconstructed image a blocky appearance. The image in Fig. 21 has been coded at 0.25 bits per pixel using the JPEG algorithm. The blocky appearance is clearly apparent. This has led to the increasing popularity of subband and wavelet-based schemes. The implementation for both subband and wavelet-based schemes is similar. The input is filtered through a bank of filters, called the *analysis filterbank*. The filters cover the entire frequency range of the signal. As the bandwidth of each filter is only a fraction of the bandwidth of the original signal, the Nyquist criterion dictates that the number of samples required at the output of the filter be less than the number of samples per second required at the input of the filter. The output of the filters is subsampled or *decimated* and encoded. The decimated output values are quantized, and the quantization labels are encoded. At the decoder, after the received samples are decoded they are *upsampled* by the insertion of zeros between the received samples and filtered using a bank of reconstruction filters. A two-band subband coding scheme is shown in Fig. 22. The major components of the design of subband coding schemes are the selection of the filters and the encoding method used for the subbands. In order to determine the latter, it may be necessary to allocate a predetermined bit budget between the various bands.

Notice that in the system shown in Fig. 22 if the filters are not ideal filters then at least one of the two analysis filters will have a bandwidth greater than half the bandwidth of the source output. If the source is initially sampled at the Nyquist rate, then when we subsample by two that particular filter output will effectively be sampled at less the Nyquist rate, thus introducing aliasing. One of the objectives of the design of the analysis and synthesis filterbanks is to remove the effect of aliasing.

If we ignore the quantization and coding for the moment, we can determine conditions on the analysis and synthesis filterbanks such that the reconstruction is a delayed version of the source output. If we represent the samples of the source sequence by $x(n)$ and the reconstruction sequence by $x(n)$, then this requirement known as the *perfect reconstruction* (*PR*) requirement can be written as

$$x(n) = cx(n - n_0) \tag{8}$$

where $c$ is a constant. In terms of the $Z$ transforms of the sequences, we can write the PR requirement as

$$X(z) = cz^{-n_0}X(z) \tag{9}$$

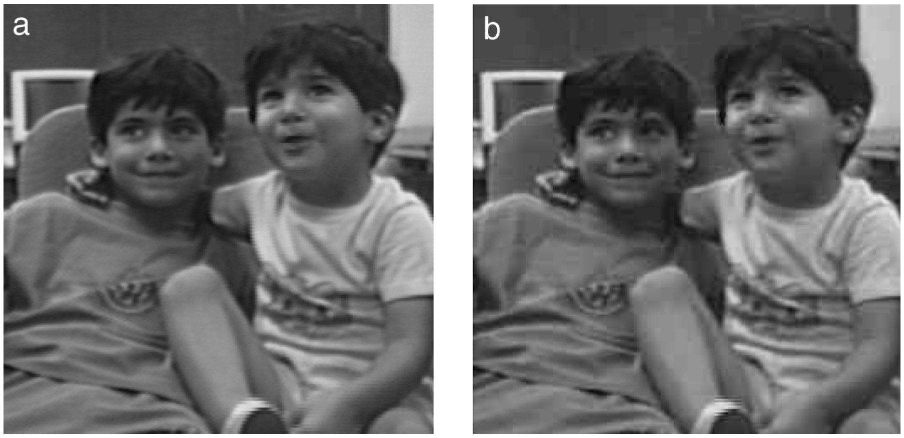


**Figure 20** (a) The original *sensin* image and (b) the *sensin* image coded at 0.5 bits per pixel using JPEG.

Let $H_1(z)$ and $H_2(z)$ be the transfer functions for the analysis filters and $K_1(z)$ and $K_2(z)$ be the transfer functions of the synthesis filters. Then we can show (Sayood, 2000) that

$$X(z) = \frac{1}{2}\,[H_1(z)K_1(z) + H_2(z)K_2(z)]\,X(z) + \frac{1}{2}\,[H_1(-z)K_1(z) + H_2(-z)K_2(z)]\,X(-z) \qquad (10)$$

Examining this equation we can see that in order for the perfect reconstruction condition to be satisfied, we need

$$H_1(-z)K_1(z) + H_2(-z)K_2(z) = 0 \qquad (11)$$

$$H_1(z)K_1(z) + H_2(z)K_2(z) = cz^{-n_0} \qquad (12)$$

The first equation is satisfied if we pick the synthesis filters as

$$K_1(z) = -H_2(z) \qquad (13)$$

$$K_2(z) = H_1(-z) \qquad (14)$$

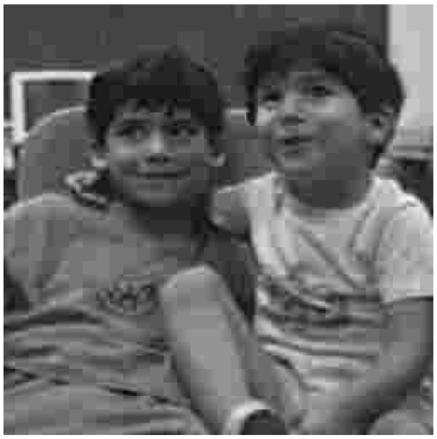


**Figure 21** The *sensin* image coded at 0.25 bits per pixel using JPEG.

To satisfy the second equation we can select $H_2(z)$ as (Mintzer, 1985; Smith and Barnwell, 1984)

$$H_2(z) = z^{-N}\,H_1(-z^{-1}) \qquad (15)$$

Thus, all four filters can be expressed in terms of one prototype filter. We can show that this prototype filter should have an impulse response satisfying

$$\sum_{k=0}^{N} h_k h_{k+2n} = \delta_n \qquad (16)$$

We can arrive at the same requirement on the filter coefficients using a wavelet formulation. Once we have obtained the coefficients for the filters, the compression approach using wavelets is similar. After the source output has been decomposed the next step is the quantization and coding of the coefficients. The two most popular approaches to quantization and coding for image compression are the embedded zerotree (EZW) (Shapiro, 1993) and the set partitioning in hierarchical trees (SPIHT) (Said and Pearlman, 1996) approaches.

Both these approaches make use of the fact that there is a relationship between the various subbands. A natural image is essentially a low-pass signal. Therefore, most of the energy in a wavelet or subband decomposition is concentrated in the LL band. One effect of this is that if the coefficient representing a particular pixel in the LL band is less than a specified threshold, the coefficients corresponding to the same pixel in the other bands will also have a magnitude smaller than that threshold. Thus, during coding we can scan the coefficients in the LL band first and compare them against a sequence of decreasing

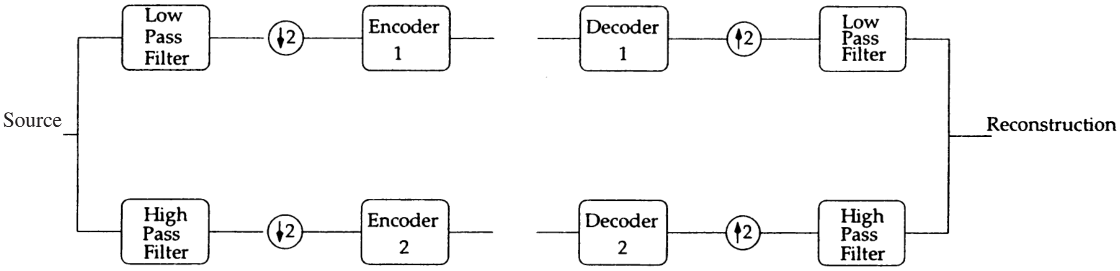


**Figure 22** A two-band subband coding scheme.

thresholds. If the coefficient is less than the threshold we can check to see if the corresponding coefficients in the other bands are also less than this threshold. This information is then transmitted to the decoder. If the coefficients in the other band are also less than the threshold this is a highly efficient code. Note that the efficiency is dependent on the image being low pass. For more high-pass images, such as remotely sensed images, this strategy is not very effective.

In Fig. 23 we have the *sensin* image coded at rates of 0.5 bits per pixel and 0.25 bits per pixel using the SPIHT algorithm. Comparing the 0.25 bits per pixel reconstruction to Fig. 21 we can see the absence of blockiness. However, there are different artifacts that have taken the place of the blockiness. Neither reconstruction is very good at this rate.

A descendant of these techniques, known as EBCOT (Taubman, 2000), is the basis for the new JPEG 2000 image compression standard. Detailed information about these techniques can be found in Said and Pearlman (1996), Sayood (2000), and Shapiro (1993).

## E. Analysis-Synthesis Schemes

When possible, one of the most effective means of compression is to transmit instructions on how to reconstruct the source rather than transmitting the source samples. In order to do this we should have a fairly good idea about how the source samples were generated. One particular source for which this is true is human speech.

Human speech can be modeled as the output of a linear filter which is excited by either white noise or a periodic input or a combination of the two. One of the earliest modern compression algorithms made use of this fact to provide a very high compression of speech. The technique, known as linear predictive coding, has its best known embodiment in the (now outdated) U.S. Government standard LPC-10. Some of the basic aspects of this standard are still alive, albeit in modified form in today's standards.

The LPC-10 standard assumes a model of speech pictured in Fig. 24. The speech is divided into frames. Each frame is classified as voiced or unvoiced. For the

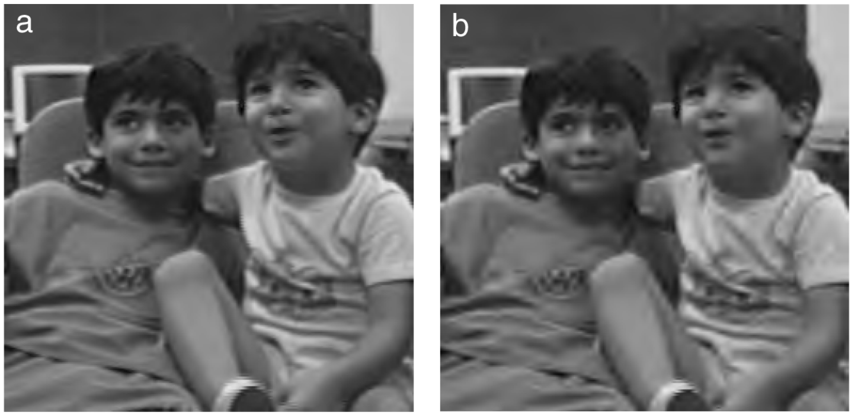


**Figure 23** The *sensin* image coded at (a) 0.5 bits per pixel and (b) 0.25 bits per pixel using SPIHT.

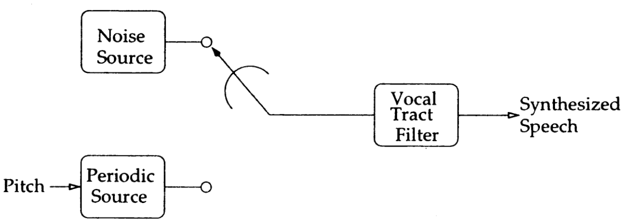


**Figure 24** Speech synthesis model used by LPC-10.

voiced speech the pitch period for the speech sample is extracted. The parameters of the vocal tract filter are also extracted and quantized. All this information is sent to the decoder. The decoder synthesizes the speech samples $y_n$ as

$$y_n = \sum_{i=1}^{M} b_i y_{n-i} + G\epsilon_n \tag{17}$$

where $\{b_i\}$ is the coefficient of the vocal tract filter. The input to the filter, the sequence $\{\epsilon_n\}$, is either the output of a noise generator or a periodic pulse train, where the period of the pulse train is the pitch period.

Since the introduction of the LPC-10 standard there has been a considerable increase in the sophistication of speech coders. In code excited linear prediction (CELP) the vocal tract filter is excited by elements of an excitation codebook. The entries of the codebook are used as input to a vocal filter of the form

$$y_n = \sum_{i=1}^{10} b_i y_{n-i} + \beta y_{n-P} + G\epsilon_n \tag{18}$$

where $P$ is the pitch period. The synthesized speech is compared with the actual speech, and the codebook entry that provides the closest perceptual match is selected. The index for this entry is sent to the decoder along with the vocal tract filter parameters.

Mixed excitation linear prediction (MELP) uses a somewhat more complex approach to generating the excitation signal. The input is subjected to a multiband voicing analysis using five filters. The results of the analysis are used with a complex pitch detection strategy to obtain a rich excitation signal.

## F. Video Compression

Currently, the source that requires the most resources in terms of bits and, therefore, has benefitted the most from compression is video. We can think of video as a sequence of images. With this view video compression becomes repetitive image compression and we can compress each frame separately. This is the point of view adopted by M-JPEG, or motion JPEG, in which each frame is compressed using the JPEG algorithm.

However, we know that in most video sequences there is a substantial amount of correlation between frames. It is much more efficient to send differences between the frames rather than the frames themselves. This idea is the basis for several international standards in video compression. In the following we briefly look at some of the compression algorithms used in these standards. Note that the standards contain much more than just the compression algorithms.

The ITU H.261 and its descendant ITU H.263[1] are international standards developed by the ITU, which is a part of the United Nations organization. A block diagram of the H.261 video coding algorithm is shown in Fig. 25. The image is divided into blocks of size 8 × 8. The previous frame is used to predict the values of the pixels in the block being encoded. As the objects in each frame may have been offset from the previous frame, the block in the identical location is not always used. Instead the block of size 8 × 8 in the previous frame which is closest to the block being encoded in the current frame is used as the predicted value. In order to reduce computations the search area for the closest match is restricted to lie within a prescribed region around the location of the block being encoded. This form of prediction is known as *motion compensated prediction.* The offset of the block used for prediction from the block being encoded is referred to as the *motion vector* and is transmitted to the decoder. The loop filter is used to prevent sharp transitions in the previous frame from generating high frequency components in the difference.

The difference is encoded using transform coding. The DCT is used followed by uniform quantization. The DC coefficient is quantized using a scalar quantizer with a step size of 8. The other coefficients are quantized with 1 of 31 other quantizers, all of which are midtread quantizers, with step sizes between 2 and 62. The selection of the quantizer depends in part on the availability of transmission resources. If higher compression is needed (fewer bits available), a larger step size is selected. If less compression is acceptable, a smaller step size is selected. The quantization labels are scanned in a zigzag fashion and encoded in a manner similar to (though not the same as) JPEG.

[1] Originally published in 1996; an update published in 1998 is commonly referred to as H.263+.

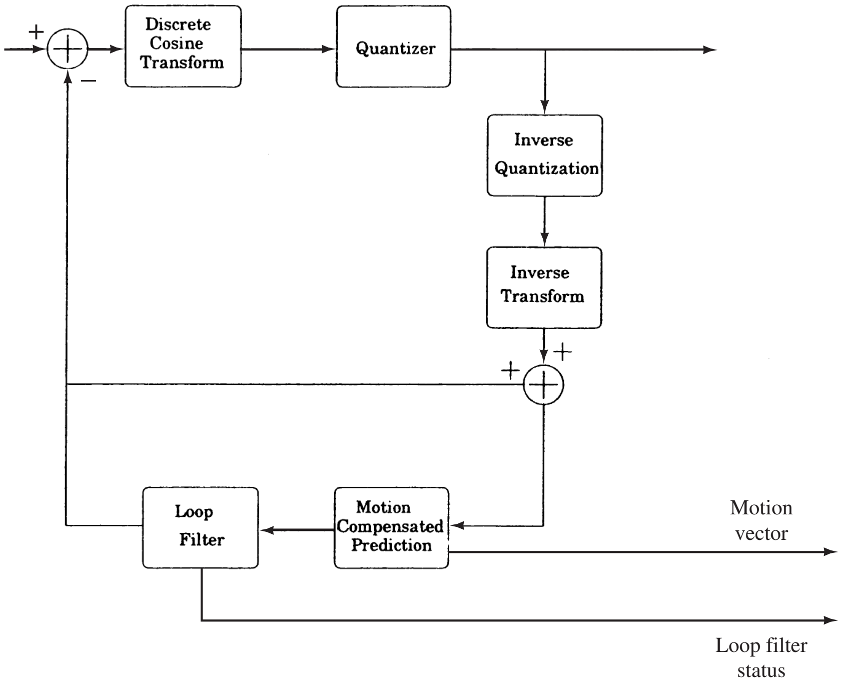


**Figure 25** Block diagram of the ITU-T H.261 video compression algorithm.

The coding algorithm for ITU-T H.263 is similar to that used for H.261 with some improvements. The improvements include:

- Better motion compensated prediction
- Better coding
- Increased number of formats
- Increased error resilience

There are a number of other improvements that are not essential for the compression algorithm. As mentioned before, there are two versions of H.263. As the earlier version is a subset of the latter one; we only describe the later version.

The prediction is enhanced in H.263 in a number of ways. By interpolating between neighboring pixels in the frame being searched, a “larger image” is created. This essentially allows motion compensation displacement of half pixel steps rather than integer number of pixels. There is an unrestricted motion vector mode that allows references to areas outside the picture, where the outside areas are generated by duplicating the pixels at the image boundaries. Finally, in H.263+ the prediction can be generated by a frame that is not the previous frame. An independent segment decoding mode allows the frame to be broken into segments where each segment can be decoded independently. This prevents error propagation and also allows for greater control over quality of regions in the reconstruction. The H.263 standard also allows for bidirectional prediction.

As the H.261 algorithm was designed for videoconferencing, there was no consideration given to the need for random access. The MPEG standards incorporate this need by requiring that at fixed intervals a frame of an image be encoded without reference to past frames. The MPEG-1 standard defines three different kinds of frames: *I* frames, *P* frames, and *B* frames. An *I* frame is coded without reference to previous frames, that is, no use is made of prediction from previous frames. The use of the *I* frames allows random access. If such frames did not exist in the video sequence, then to view any given frame we would have to decompress all previous frames, as the reconstruction of each frame would be dependent on the prediction from previous frames. The use of periodic *I* frames is also necessary if the standard is to be used for compressing television programming. A viewer should have the ability to turn on the television (and the MPEG decoder) at more or less any time during the broadcast. If there are periodic *I* frames available, then the decoder can start decoding from the first *I* frame. Without the redundancy removal provided via prediction the compression obtained with *I* frames is less than would have been possible if prediction had been used.

The *P* frame is similar to frames in the H.261 standard in that it is generated based on prediction from previous reconstructed frames. The similarity is closer to H.263, as the MPEG-1 standard allows for half pixel shifts during motion compensated prediction.

The *B* frame was introduced in the MPEG-1 standard to offset the loss of compression efficiency occasioned by the *I* frames. The *B* frame is generated using prediction from the previous *P* or *I* frame and the nearest future *P* or *I* frame. This results in extremely good prediction and a high level of compression. The *B* frames are not used to predict other frames, therefore, the *B* frames can tolerate more error. This also permits higher levels of compression.

The various frames are organized together in a *group of pictures* (GOP). A GOP is the smallest random access unit in the video sequence. Therefore, it has to contain at least one *I* frame. Furthermore, the first *I* frame in a GOP is either the first frame of the GOP or is preceded by *B* frames which use motion compensated prediction only from this *I* frame. A possible GOP is shown in Fig. 26. Notice that in order to reconstruct frames 2, 3, and 4, which are *B* frames, we need to have the *I* and *P* frames. Therefore, the order in which these frames are transmitted is different from the order in which they are displayed.

The MPEG-2 standard extends the MPEG-1 standard to higher bit rates, bigger picture sizes, and interlaced frames. Where MPEG-1 allows half pixel displacements, the MPEG-2 standard requires half pixel displacements for motion compensation. Furthermore, the MPEG-2 standard contains several additional modes of prediction. A full description of any of these standards is well beyond the scope of this article. For details the readers are referred to the standards ISO/IEC IS 11172, 13818, and 14496 and books Gibson *et al.* (1998), Mitchell *et al.* (1997), and Sayood (2000).

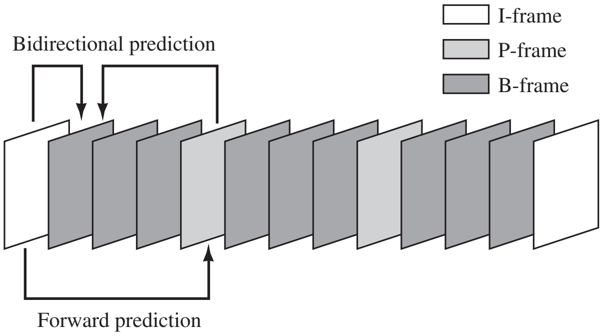


**Figure 26** A possible arrangement for a GOP.

## IV. FURTHER INFORMATION

We have described a number of compression techniques. How they compare relative to each other depends on the performance criteria, which in turn depend on the application. An excellent resource for people interested in comparisons between the multitude of compression programs available for different applications is the *Archive Compression Test* maintained by Jeff Gilchrist at http://act.by.net. Another excellent resource on the Internet is the data compression page maintained by Mark Nelson at http://dogma.net/DataCompression/. This page contains links to many other data compression resources, including programs and a set of informative articles by Mark Nelson. Programs implementing some of the techniques described here can also be obtained at ftp://ftp.mkp.com/pub/Sayood/.

## SEE ALSO THE FOLLOWING ARTICLES

Desktop Publishing • Electronic Data Interchange • Error Detecting and Correcting Codes • Multimedia

## BIBLIOGRAPHY

Burrows, M., and Wheeler, D. J. (1994). A Block Sorting Data Compression Algorithm. Technical Report SRC 124, Digital Systems Research Center.

Chen, W.-H., and Pratt, W. K. (March 1984). Scene Adaptive Coder. IEEE Trans. Communications. COM-32:225–232.

Cleary, J. G., and Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. IEEE Trans. Communications. 32(4):396–402.

Cleary, J. G., and Teahan, W. J. (February 1997). Unbounded length contexts for PPM. Computer Journal, 40:30–74.

Cutler, C. C. (July 29, 1952). Differential Quantization for Television Signals. U.S. Patent 2 605 361.

Gallagher, R. G. (November 1978). Variations on a theme by Huffman. IEEE Trans. Information Theory. IT-24(6):668–674.

Gersho, A., and Gray, R. M. (1991). *Vector Quantization and Signal Compression.* Dordrecht/Norwell, MA: Kluwer Academic.

Gibson, J. D., Berger, T., Lookabaugh, T., Lindbergh, D., and Baker, R. (1998). *Digital Compression for Multimedia: Principles and Standards.* San Mateo, CA: Morgan Kaufmann.

Gish, H., and Pierce, J. N. (September 1968). Asymptotically efficient quantization. IEEE Trans. Information Theory. IT-14:676–683.

Huang, J.-Y., and Schultheiss, P. M. (September 1963). Block quantization of correlated gaussian random variables. IEEE Trans. Communication Systems. CS-11:289–296.

Huffman, D. A. (1951). A method for the construction of minimum redundancy codes. Proc. IRE. 40:1098–1101.

ISO/IECIS 11172. Information Technology—Coding of Moving

Pictures and Associated Audio for Digital Storage Media Up To About 1.5 Mbits/s.

ISO/IECIS 13818. Information Technology—Generic Coding of Moving Pictures and Associated Audio Information.

ISO/IECIS 14496. Coding of Moving Pictures and Audio.

Linde, Y., Buzo, A., and Gray, R. M. (January 1980). An algorithm for vector quantization design. IEEE Trans. Communications. COM-28:84–95.

Max, J. (January 1960). Quantizing for minimum distortion. IRE Trans. Information Theory. IT-6:7–12.

Memon, N. D., and Sayood, K. (1995). Lossless Image Compression: A Comparitive Study. In *Proceedings SPIE Conference on Electronic Imaging*. SPIE.

Mintzer, F. (June 1985). Filters for distortion-free two-band multirate filter banks. *IEEE Trans. Acoustics, Speech, and Signal Processing*. ASSP-33:626–630.

Mitchell, J. L., Pennebaker, W. B., Fogg, C. E., and LeGall, D. J. (1997). *MPEG Video Compression Standard*. London/New York: Chapman & Hall.

Nelson, M. (September 1996). Data compression with the Burrows-Wheeler transform. Dr. Dobbs Journal.

Nelson, M., and Gailly, J.-L. (1996). *The Data Compression Book*. California: M&T Books.

Pasco, R. (1976). Source Coding Algorithms for Fast Data Compression. Ph.D. thesis, Stanford University.

Pennebaker, W. B., and Mitchell, J. L. (1993). *JPEG Still Image Data Compression Standard*. New York: Van Nostrand-Reinhold.

Rissanen, J. J. (May 1976). Generalized Kraft inequality and arithmetic coding. IBM J. Research and Development. 20:198–203.

Said, A., and Pearlman, W. A. (June 1996). A new fast and efficient coder based on set partitioning in hierarchical trees. *IEEE Trans. Circuits and Systems for Video Technologies*. 243–250.

Sayood, K. (2000). *Introduction to Data Compression, Second Edition*. San Mateo, CA: Morgan Kauffman/Academic Press.

Sayood, K., and Na, S. (November 1992). Recursively indexed quantization of memoryless sources. IEEE Trans. Information Theory. IT-38:1602–1609.

Shannon, C. E. (1948). A mathematical theory of communication. Bell System Technical J. 27:379–423,623–656.

Shapiro, J. M. (December 1993). Embedded image coding using zerotrees of wavelet coefficients. IEEE Trans. Signal Processing. SP-41:3445–3462.

Smith, M. J. T., and Barnwell, T. P., III. (1984). A Procedure for Designing Exact Reconstruction Filter Banks for Tree Structured Subband Coders. In *Proceedings IEEE International Conference on Acoustics Speech and Signal Processing*. IEEE.

Storer, J. A., and Syzmanski, T. G. (1982). Data compression via textual substitution. J. ACM. 29:928–951.

Taubman, D. (July 2000). High performance scalable image motion compression with EBCOT. IEEE Trans. Image Processing. IP-9:1158–1170.

Weinberger, M., Seroussi, G., and Sapiro, G. (November 1998). The LOCO-I Lossless Compression Algorithm: Principles and Standardization into JPEG-LS. Technical Report HPL-98-193, Hewlett-Packard Laboratory.

Weinberger, M. J., Rissanen, J. J., and Arps, R. (1995). Applications of Universal Context Modeling to Lossless Compression of Gray-Scale Images. In *Proc. Asilomar Conference on Signals, Systems and Computers,* pp. 229–233. IEEE.

Welch, T. A. (June 1984). A technique for high-performance data compression. IEEE Computer. 8–19.

Wu, X., and Memon, N. D. (May 1996). CALIC—A context based adaptive lossless image coding scheme. IEEE Trans. Communications.

Ziv, J., and Lempel, A. (May 1977). A universal algorithm for data compression. IEEE Trans. Information Theory. IT-23(3):337–343.

Ziv, J., and Lempel, A. (September 1978). Compression of individual sequences via variable-rate coding. IEEE Trans. Information Theory. IT-24(5):530–536.

# Data Envelopment Analysis

**Timothy Anderson**
*Portland State University*

## GLOSSARY

**BCC** The BCC model is one of the most commonly used DEA models. It is credited to Banker, Charnes, and Cooper. This model differs from the CCR model in that it exhibits variable returns to scale rather than constant returns to scale.

**CCR** Perhaps the most commonly used DEA model originating with Charnes, Cooper, and Rhodes. This model exhibits constant returns to scale.

**DMU** A decision making unit. For example, in a DEA application to an information systems context, this may refer to a software development team.

**envelopment and multiplier formulations** DEA is based on linear programming. As with all linear programs, there is a primal and a dual formulation. The selection of which one to call primal and which to call dual is arbitrary and, therefore, the terms envelopment and multiplier are used to eliminate ambiguity. The envelopment model focuses on allowing the "best" DMUs to envelope the rest of the DMUs. The multiplier formulation uses nonnegative weights to cast each DMU in the best possible light relative to the other DMUs.

**inputs and outputs** Inputs are the resources used by a DMU in achieving its goals. Inputs are "bads" in that increasing levels of an input while holding everything else constant should generally result in a lower efficiency score. Outputs have the opposite property. Examples of DEA inputs might include the number of staff assigned to a team or capital expenditures in networking. Outputs might be lines of code or reduced computing time.

**orientation** DEA models often have two important but underappreciated variations based on the orientation of the model. An input-oriented model primarily focuses on input reduction while an output-oriented primarily model focuses on output augmentation.

**returns to scale** Two of the most common returns to scale assumptions are constant and variable. Constant returns to scale (or CRS) implies that doubling each of the inputs used by a DMU should double each of the outputs. Variable returns to scale (or VRS) implies that doubling each of the inputs used by a DMU does not necessarily double each of the outputs.

**weight restrictions** DEA normally does not place any restrictions on the relative trade-offs between the inputs or the trade-offs between the outputs. This can lead to unrealistic or extreme trade-offs. Various weight restriction techniques can be applied to overcome this.

As we struggle with the problem of measuring and evaluating the benefits of information technology projects or better ways to conduct software development projects, it has become apparent that we need better tools for analysis. Data envelopment analysis (DEA) provides a powerful technique for evaluating complex processes and systems such as those encountered in information systems. It provides a quantitative and robust methodology for evaluating software development projects, teams, or IT departments relying on comparison to their peers and can therefore be added to the toolkit of techniques.

This article is meant to be an introduction to Data Envelopment Analysis (DEA), which is becoming an increasingly popular tool for measuring and assessing the performance of information systems and other complex systems. This article is not meant to be an exhaustive tutorial on DEA. For that purpose, there are several books that have recently been published. Because of the varying backgrounds of readers, this introduction focuses more on gaining an intuitive understanding of DEA than mathematical rigor.

## I. INTRODUCTION TO DATA ENVELOPMENT ANALYSIS

Data Envelopment Analysis (DEA) is commonly used to evaluate the efficiency of a number of producers. A typical statistical approach is characterized as a central tendency approach and it evaluates producers relative to an average or representative producer. In contrast, DEA is an extreme point method and compares each producer with only the "best" producers. In the DEA literature, a producer is usually referred to as a decision-making unit or DMU. Extreme point methods are not always the right tool for a problem but are appropriate in certain cases. (See strengths and limitations of DEA in Sections III.C and III.D.)

A fundamental assumption behind DEA is convexity. First, we will assume that if a given producer, A, is capable of producing Y(A) units of output with X(A) inputs, then other producers should also be able to do the same if they were to operate efficiently. Similarly, if producer B is capable of producing Y(B) units of output with X(B) inputs, then other producers should also be capable of the same production schedule. By convexity, we will assume that we can combine a percentage mix of A and B (as well as others) to form a composite producer with composite inputs and composite outputs. Since this composite producer does not necessarily exist, it is sometimes called a virtual producer.

The heart of the analysis lies in finding the "best" virtual producer for each real producer. If the virtual producer is better than the original producer by either making more outputs with the same inputs or making the same outputs with less inputs then the original producer is *inefficient.* Some of the subtleties of DEA are introduced in the various ways that producers A and B can be scaled up or down and combined.

The procedure of finding the best virtual producer can be formulated as a linear program. Analyzing the efficiency of $n$ producers is then a set of $n$ linear programming (LP) problems. The following formulation is one of the standard forms for DEA. The vector $\lambda$ describes the components of other producers used to construct the virtual producer. The matrices $X$ and $Y$ describe the virtual inputs and outputs, respectively. The vectors $X_0$ and $Y_0$ are the input and output vectors for the DMU currently being examined. The products $X\lambda$ and $Y\lambda$ are the targets of performance for the analyzed producer. The value of $\theta$ can be used as a measure of the producer's efficiency. This results in the envelopment formulation of DEA.

$$\begin{aligned} &\min_{\theta,\lambda} \theta, \\ &\text{s.t.} \quad Y\lambda \geq Y_0, \\ &\qquad X\lambda \leq \theta X_0, \\ &\qquad \lambda \geq 0. \end{aligned} \tag{1}$$

It should be emphasized that an LP of this form must be solved for each of the DMUs. There are other ways to formulate this problem such as the ratio model which leads to the multiplier formulation which we will cover shortly. The first constraint of Eq. (1) forces the virtual DMU to produce at least as many outputs as the DMU, $\text{DMU}_0$, with inputs $X_0$ and outputs $Y_0$, to be evaluated. The second constraint finds out how much less input the virtual DMU would need. Hence, it is called input-oriented. The factor used to scale back the inputs is $\theta$ and this value is called a radial measure of the efficiency of $\text{DMU}_0$.

The target of performance as described by $X\lambda$ and $Y\lambda$ may not necessarily be optimal in that there may be nonradial slacks. The following formulation provides a remedy to this problem by introducing slack variables and an infinitesimal, $\varepsilon$, as a multiplier in the objective function that is greater than zero but smaller than any real number. The resulting formulation is given in Eq. (2).

$$\begin{aligned} &\min_{\theta,\lambda,s+,s-} \theta - \varepsilon s^{+} - \varepsilon s^{-}, \\ &\text{s.t.} \quad Y\lambda - Y_0 - s^{+} = 0, \\ &\qquad \theta X_0 - X\lambda - s^{-} = 0, \\ &\qquad \lambda, s^{+}, s^{-} \geq 0. \end{aligned} \tag{2}$$

In the computer codes used, this infinitesimal multiplier is treated preemptively. In the first stage, the goal is to find the vector, $\lambda$, that gives the lowest possible efficiency score, $\theta$. The second stage goal is given the value of $\theta$ from the previous stage, try to maximize the sum of the slacks. The result is that two linear programs are needed for each DMU. Algorithmic extensions have been developed to reduce the computational burden, but since these linear programs are relatively small it is not critical and we defer the reader to more exhaustive volumes by Charnes and colleagues. It is important not to use a finite approximation to $\varepsilon$ such as $10^{-6}$, since this can (1) cause a significant computational problem and (2) yield erroneous results. For more information on computational problems, see Ali.

To see how this operational procedure is used, we first solve Eq. (1) to obtain a value of $\theta = \theta^*$. We then replace Eq. (2) with Eq. (3).

$$\begin{aligned} &\max_{\lambda,s+,s-} \; s^+ + s^-, \\ &\text{s.t.} \quad Y\lambda - Y_0 - s^+ = 0, \\ &\qquad\; \theta X_0 - X\lambda - s^- = 0, \\ &\qquad\; \theta = \theta^* \\ &\qquad\; \lambda, s^+, s^- \geq 0. \end{aligned} \tag{3}$$

Notice that the objective function has changed and the new constraint, $\theta = \theta^*$, sets the radial reduction equal to the value from the first phase. The goal of this second phase is to maximize the slacks to identify all mix inefficiencies.

Putting this all together we obtain new values $\hat{X}_0 = \theta^* X_0 - s^{-*} \leq X_0$ and $\hat{Y}_0 = Y_0 + s^{+*} \geq Y_0$. The values of $\hat{X}_0$ and $\hat{Y}_0$ are referred to as projections for DMU$_0$ and describe the amount of inputs and outputs that the DMU should have needed based upon the model used. This combination of $\hat{X}_0$ and $\hat{Y}_0$ is efficient and the differences between these targets of performance and the actual performance, $\Delta X_0 = X_0 - \hat{X}_0 \geq 0$ and $\Delta Y_0 = Y_0 - \hat{Y}_0 \geq 0$, indicates the inefficiency in the performance of the DMU relative to the other DMUs.

## A. Simple Numerical Example

A simple numerical example will help to show what DEA is doing. Assume that there are three programmers (DMUs), A, B, and C, that have each had programming and/or documentation responsibilities. Our goal is to evaluate each of the programmers to determine their productivity. We may use this information in order to determine project bonus, make employee retention decisions, or various other purposes.

Each of the three programmers spent 100 hours in completing their work. Programmer A only develops code to the exclusion of documentation, programmer C does a lot of documentation, and programmer B does a mix of programming and documentation. This is summarized in the following table.

Now, as a DEA analyst, we play the role of Dr. Frankenstein by combining parts of different programmers. First let us analyze programmer A. Clearly no combination of programmers B and C can produce 40 function points with the constraint of only 100 hours. Therefore, programmer A is efficient at developing function points and receives an efficiency of 1.0.

**Table I** Programmer Data

| | Input | Output #1 | Output #2 |
|---|---|---|---|
| Programmer (DMU) | Hours | Function Points | Pages of documentation |
| A | 100 | 40 | 0 |
| B | 100 | 20 | 5 |
| C | 100 | 10 | 20 |

Now we move on to analyze programmer B. Suppose we try a 50–50 mixture of programmers A and C. This means that $\lambda = [0.5, 0.5]$. The virtual output vector is now,

$$\begin{aligned} Y\lambda = [0.5 * 40 + 0.5 * 10,\ 0.5 * 0 + 0.5 * 20] \\ = [25, 10] \end{aligned}$$

Note that $X = 100 = \mathrm{X}(0)$ where X(0) is the input(s) for the DMU being analyzed. Since $Y\lambda > \mathrm{Y}(0) = [20, 5]$, there is room to scale down the inputs, $X$, and produce a virtual output vector at least equal to or greater than the original output. This scaling down factor would allow us to put an upper bound on the efficiency of that programmer's efficiency. The 50–50 ratio of A and C may not necessarily be the optimal virtual producer. The efficiency, $\theta$, can then be found by solving the corresponding linear program.

It can be seen by inspection that programmer C is efficient because no combination of programmers A and B can produce his total of 20 pages of documentation in only 100 hours. Programmer C is fulfilling the role of producing pages of documentation more efficiently than any other programmer just as programmer A is coding function points more efficiently than anyone else. Programmer B was outproduced by a combination of programmers A and C.

This example can be made more complicated by looking at unequal values of inputs instead of the constant 100 hours, by making it a multiple input problem, or by adding more data points, but the basic principles still hold.

## B. Graphical Example

The single input two-output or two input-one output problems are easy to analyze graphically. The previous numerical example is now solved graphically in Fig. 1. (An assumption of constant returns to scale is made and explained in detail later.) The evaluation of programmer B is depicted in Table II.

If it is assumed that convex combinations of programmers are allowed, then the line segment connecting programmers A and C shows the possibilities of virtual outputs that can be formed from these two programmers. Similar segments can be drawn between A and B along with B and C. The segment AC lies beyond the segments AB and BC, so this means that a convex combination of A and C will create the most outputs for a given set of inputs.

This line is called the efficiency frontier. The efficiency frontier defines the maximum combinations of outputs that can be produced for a given set of inputs. The segment connecting point C to the Doc axis is drawn because of disposability of output. It is assumed that if programmer C can produce 20 pages of documentation and 10 function points, he could also produce 20 pages of documentation without any function points. We have no knowledge though of whether avoiding function points altogether would allow him to increase his production of documentation, so we must assume that it remains constant.

Since programmer B lies below the efficiency frontier, he is inefficient. His efficiency can be determined by comparing him to a virtual programmer formed from programmer A and programmer C. The virtual programmer, called V, is approximately 64% of programmer C and 36% of programmer A. (This can be determined by an application of the lever law. Pull out a ruler and measure the lengths of AV, CV, and AC. The percentage of programmer C is then AV/AC and the percentage of programmer A is CV/AC.)

The efficiency of programmer B is then calculated by finding the fraction of inputs that programmer V would need to produce as many outputs as programmer B. This is easily calculated by looking at the line from the origin, O, to V. The efficiency of programmer B is OB/OV, which is approximately 68%. This figure also shows that programmers A and C are efficient since they lie on the efficiency frontier. In other words, any virtual programmer formed for analyzing programmers A and C will lie on programmers A and C, respectively. Therefore, since the efficiency is calculated as the ratio of OA/OA or OC/OC, programmers A and C will have efficiency scores equal to 1.0.

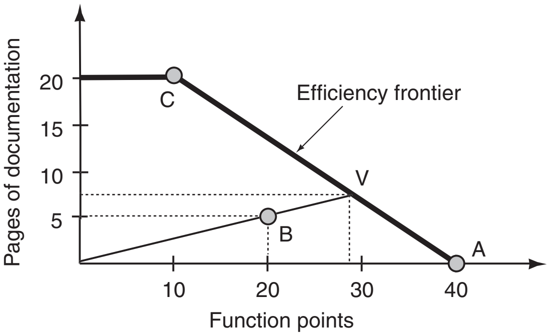


**Figure 1** Graphical example of DEA for Programmer B.

The graphical method is useful in this simple two-dimensional example but gets much harder in higher dimensions. The normal method of evaluating the efficiency of programmer B is by using a linear programming formulation of DEA such as Eqs. (1) or (2).

The following linear program is a numerical illustration of Eq. (1) for the evaluation of DMU (or programmer) B.

$$\begin{aligned} &\min_{\theta,\lambda,} \theta, \\ &\text{s.t.} \quad \begin{bmatrix} 40 & 20 & 10 \\ 0 & 5 & 20 \end{bmatrix} \lambda \geq \begin{bmatrix} 20 \\ 5 \end{bmatrix}, \\ &\qquad [100 \quad 100 \quad 100]\lambda \leq \theta \cdot 100, \\ &\qquad \lambda \geq 0. \end{aligned} \tag{4}$$

Bear in mind that as described earlier, this will give the correct score, θ, but may not necessarily yield the optimal target of performance. To do that, it would be necessary to follow the earlier two-phase approach. In this case, the target of performance is described in Table II. This figure is an illustration of the type of information obtained from a DEA software package, in this case DEA-Solver, 1.0 from Saitech Inc.

The formulation that we have used is an input-oriented formulation in that the focus is on finding the level of input (or the number of hours in this example) that should have been needed to produce the same or more output. The results indicate that Programmer B should have been able to accomplish his or her work in 68.75 hours instead of 100 hours.

If the goal had been instead to determine how much work should have been finished in 100 hours, we would instead use an output-oriented model. The output-oriented equivalent to formulation Eq. (1) is given by Eq. (5).

$$\begin{aligned} &\max_{\phi,\lambda} \phi, \\ &\text{s.t.} \quad Y\lambda \geq \phi Y_0, \\ &\qquad X\lambda \leq X_0, \\ &\qquad \lambda \geq 0. \end{aligned} \tag{5}$$

Equation (5) can be readily extended to account for slacks in the same manner as was given in Eq. (2).

**Table II** Input-Oriented Results

| | Score | | | |
|---|---|---|---|---|
| DMU | I/O Data | Projection | Difference | % |
| A | 1 | | | |
| Hours | 100 | 100 | 0 | 0.00 |
| Function points | 40 | 40 | 0 | 0.00 |
| Pages of documentation | 0 | 0 | 0 | 0.00 |
| B | 0.6875 | | | |
| Hours | 100 | 68.75 | −31.25 | −31.25 |
| Function points | 20 | 20 | 0 | 0.00 |
| Pages of documentation | 5 | 5 | 0 | 0.00 |
| C | 1 | | | |
| Hours | 100 | 100 | 0 | 0.00 |
| Function points | 10 | 10 | 0 | 0.00 |
| Pages of documentation | 20 | 20 | 0 | 0.00 |

The results of the output-oriented model are given in Table III.

The numerical efficiency score of 0.6875 remains unchanged in this model. This is actually the inverse of $\phi$ as given in Eq. (5). In the case of constant returns to scale, $\theta = 1/\phi$ or the results of the objective functions from Eqs. (1) and (5) are reciprocals of each other. This relationship does not necessarily hold in the case of other returns to scale assumptions. The issue of returns to scale is explored in Section III.B and for further treatment of this, the reader is referred to more comprehensive references in the Bibliography.

The results in Table II indicate that Programmer B should have developed code with at least 9 more function points and 2 more pages of documentation in the 100 hours. The decision of orientation is important and should be based upon the goals of the organization(s) involved, whether the primary issue is reducing inputs or increasing outputs.

## C. Data Envelopment Analysis Multiplier Model

Another approach to conducting DEA is to examine the multiplier model. This model can be thought of as an opportunity for each DMU to select the set of weights (or prices) on inputs and outputs that make it look as good as possible relative to its peers. Weights (or prices) can't be negative and a DMU can't pick a pricing scheme that makes itself or any of its peers

**Table III** Output-Oriented Results

| | Score | | | |
|---|---|---|---|---|
| DMU | I/O Data | Projection | Difference | % |
| A | 1 | | | |
| Hours | 100 | 100 | 0 | 0.00 |
| Function points | 40 | 40 | 0 | 0.00 |
| Pages of documentation | 0 | 0 | 0 | 0.00 |
| B | 0.6875 | | | |
| Hours | 100 | 100 | 0 | 0.00 |
| Function points | 20 | 29.09 | 9.09 | 45.45 |
| Pages of documentation | 5 | 7.27 | 2.27 | 45.45 |
| C | 1 | | | |
| Hours | 100 | 100 | 0 | 0.00 |
| Function points | 10 | 10 | 0 | 0.00 |
| Pages of documentation | 20 | 20 | 0 | 0.00 |

appear to be better than perfect (1.0). This is implemented by the Eq. (6).

$$\begin{aligned} \max_{u,v}\ z &= \frac{\sum_{r=1}^{s} u_r y_{r\,0}}{\sum_{i=1}^{m} v_i x_{i0},} \\ \text{s.t.}\quad & \frac{\sum_{r=1}^{s} u_r y_{rj}}{\sum_{i=1}^{m} v_i x_{ij}} \leq 1; \qquad j = 1, \mathrm{K},\ n, \\ & u_r, v_i \geq 0; \qquad r = 1, \mathrm{K},\ s; \qquad i = 1, \mathrm{K},\ m. \end{aligned} \tag{6}$$

In this case, $u_r$, represents the weight (or virtual price) of output $r$. Similarly, $v_i$, represents the weight (or virtual price) of output $i$. The number of DMUs is $n$, the number of outputs is $s$, and the number inputs is $m$. Unfortunately, this model is nonlinear. It can be readily linearized by multiplying out the denominator of the ratio in each of the $n$ constraints and by selecting a particular solution among the many linear multiples of solutions by setting the denominator in the objective function equal to one. The result is the following formulation.

$$\begin{aligned} \max_{u,v}\ z &= \sum_{r=1}^{s} u_r y_{r0}, \\ \text{s.t.}\quad & \sum_{i=1}^{m} v_i x_{i0} = 1, \\ & \sum_{r=1}^{s} u_r y_{rj} \leq \sum_{i=1}^{m} v_i x_{ij}; \qquad j = 1, \mathrm{K},\ n, \\ & u_r, v_i \geq 0; \qquad r = 1, \mathrm{K},\ s; \qquad i = 1, \mathrm{K},\ m. \end{aligned} \tag{7}$$

From here, it is relatively straightforward to implement characteristics such as limiting the relative weight (or price) of inputs or the relative weight (or price) of outputs. For example, to limit the price of input 1 to be greater than or equal to the price of input 2, simply add another constraint, $v_1 \geq v_2$. Active research is underway to improve the weight restriction techniques, but this is frequently enough to eliminate unrealistic weighting schemes from your DEA model.

## II. DATA ENVELOPMENT ANALYSIS MODELS OF INFORMATION SYSTEMS

Several researchers have used DEA to examine software development projects. The earliest work on using DEA to evaluate information systems was conducted by Banker and Kemerer with colleagues. For a particularly clear review of this literature and interesting set of three additional DEA models for software development projects, you should see work by Joseph Paradi. Mahmood has examined issues such as trying to evaluate how information technology investments benefit a company by using DEA to evaluate the complex nature of software projects.

## III. SELECTION ISSUES FOR DATA ENVELOPMENT ANALYSIS MODELS

### A. Input versus Output Orientation

One of the basic choices in selecting a DEA model is whether to use an input-orientation or an output-orientation. The difference is subtle but important and can typically be best understood by considering whether a DMU emphasizes reducing input while achieving the same level of output or emphasizes producing more output given the same level of input. Depending upon the linear program formulation and/or software used, an output-oriented efficiency will be 1.0 or higher. Scores greater than 1.0 indicate inefficiency. For example, an output-oriented score of 1.5 (or 150%) indicates that a DMU should be able to increase output by 50% while consuming the same input. (Technically, it means that a DMU should be able to produce at least 50% more of each output while using no more of any input than is currently being used.)

### B. Returns to Scale

Since this problem uses a constant input value of 100 for all of the programmers, it avoids the complications caused by allowing different returns to scale. Returns to scale refers to increasing or decreasing efficiency based on size. For example, a manufacturer can achieve certain economies of scale by producing a thousand circuit boards at a time rather than one at a time—it might be only 100 times as hard as producing one at a time. This is an example of increasing returns to scale (IRS).

On the other hand, the manufacturer might find it more than a trillion times as difficult to produce a trillion circuit boards at a time, though, because of storage problems and limits on the worldwide copper supply. This range of production illustrates decreasing returns to scale (DRS). Combining the two extreme ranges would necessitate variable returns to scale (VRS).

Constant returns to scale (CRS) means that the producers are able to linearly scale the inputs and outputs without increasing or decreasing efficiency. This is a significant assumption. The assumption of CRS may be valid over limited ranges but its use must be justified. As an aside, CRS efficiency scores will never be higher than VRS efficiency scores.

The CRS assumption can be made in our computer programming example since each hour is relatively independent, and the cumulative outputs are then the sum of individual events. For example, it is expected that doubling the number of hours or time spent will double the number of pages of documentation that a programmer produces, which implies that the CRS assumption can be used. Therefore, other situations such as VRS, IRS, and DRS are not covered here. This also explains why most of the examples concentrate on cases with equal inputs. In the one input model, the CRS assumption allows programmers to be scaled up or down, and so the multiplication of programmer inputs to achieve some constant value is implied in some cases. In a CRS model, the input-oriented efficiency score is exactly equal to the inverse of the output-oriented efficiency score. This is not necessarily true for inefficient DMUs in the case of other returns to scale assumptions.

In the DEA literature, the CRS model is typically referred to as the CCR model after the originators of the seminal publication, Charnes, Cooper, and Rhodes. Similarly, the VRS model is referred to as the BCC model after Banker, Charnes, and Cooper. Converting the CRS (or CCR) model of formulation (2) into the VRS (or BCC) formulation requires adding a constraint that the sum of the λ variables be equal to 1. This ensures that each DMU is compared against one DMU or a composite DMU that adds up to a single DMU rather than being compared solely against radically scaled up or down DMUs. The resulting formulation is given in Eq. (8).

$$\begin{aligned} &\min_{\theta,\lambda,s+,s-} \quad \theta - \varepsilon s^{+} - \varepsilon s^{-}, \\ &\text{s.t.} \quad Y\lambda - Y_0 - s^{+} = 0, \\ &\qquad \theta X_0 - X\lambda - s^{-} = 0, \\ &\qquad \overrightarrow{1} \cdot \lambda = 1 \\ &\qquad \lambda, s^{+}, s^{-} \geq 0. \end{aligned} \tag{8}$$

## C. Strengths of Data Envelopment Analysis

As the earlier list of applications suggests, DEA can be a powerful tool when used well. A few of the characteristics that make it powerful are

- It can handle multiple input and multiple output models
- It doesn't require assumption of a functional form relating inputs to outputs
- DMUs are directly compared against a peer or combination of peers which DEA identifies explicitly
- Inputs and outputs can have very different units, for example, X1 could be in units of lives saved and X2 could be in units of dollars without requiring an *a priori* trade-off between the two

## D. Limitations of Data Envelopment Analysis

The same characteristics that make DEA a powerful tool can also create problems. An analyst should keep these limitations in mind when choosing whether or not to use DEA.

- Since DEA is an extreme point technique, noise (even symmetrical noise with zero mean) such as measurement error can cause problems.
- DEA is good at estimating "relative" efficiency of a DMU, but it converges very slowly to "true" efficiency. In other words, it can tell you how well you are doing compared to your peers but not compared to a "theoretical maximum."
- Since DEA is a nonparametric technique, statistical hypothesis tests are difficult and are the focus of ongoing research.

Since a standard formulation of DEA creates a separate linear program for each DMU, large problems can be computationally intensive. Naive implementations of DEA using off-the-shelf linear programming packages can result in computational problems. I have frequently seen this with respect to the Excel Solver and poorly scaled data. This has improved in recent versions of Excel (Excel 2000's Solver seems to be much more robust), but the prevalence of degeneracy and potential for cycling are still cause for concern.

## E. Other Topics

There are a number of other variations and extensions of DEA that have been developed over the years. For example, eliminating the convexity assumption results in the Free-Disposal Hull model (FDH). FDH is useful when combinations of DMUs or rescaled versions of DMUs are inappropriate or unrealistic. The

FDH model can be easily modeled by constraining the $\lambda$ vector to consist of binary variables.

In this introduction to DEA we have focused on radial DEA models focusing on the efficiency score, $\theta$ or $\phi$. While the slacks from Eq. (2) may provide the opportunity for additional improvement, many studies only consider the efficiency scores and ignore the slacks. Another useful model is the additive model which does not include a radial score. The additive model is simply the same as Eq. (3) if you remove the constraint on $\theta = \theta^*$ and remove $\theta$ from the set of input constraints.

DEA assumes that a DMU has the ability to vary its inputs and outputs in order to reach the efficiency frontier and improve performance. Often you may find that a DMU doesn't have the ability to change certain inputs or outputs in order to try and achieve efficiency. In this case, you can consider models with nondiscretionary variables. This is beyond the scope of this introduction to DEA.

DEA assumes deterministic production of outputs using the inputs. An active area of current research is in developing extensions to DEA to better allow for stochastic variation through the use of chance constraints. On the other hand, very effective methods of sensitivity analysis are available in DEA.

## IV. PROCEDURE FOR APPLYING A DATA ENVELOPMENT ANALYSIS STUDY

Model building is as much an art as it is a science. The following is a description of steps that could be used to conduct a DEA study in the information systems industry.

### A. Decide on the Purpose of the Analysis

Is the goal to compare the efficiency of software development teams, information systems implementations, or corporate information technology departments? In these cases, you may be interested in determining best practices, recognizing excellence, or setting performance targets.

### B. Determine What Constitutes a Decision Making Unit

This sounds like a trivial step, but it can often be difficult to decide what the DMUs are to be studied. If you are comparing information technology departments, do you examine those within similar industries or as divisions of a single, large organization? If you are examining software development projects within a single company, do you consider only COBOL projects or all software development projects? In any case, be sure you have enough DMUs to exceed the number of inputs and outputs by a considerable margin. The following rule of thumb is suggested $n \geq \max\{m \times s, 3(m + s)\}$ where $n$ is the number of DMUs, $m$ is the number of inputs, and $s$ is the number of outputs.

### C. Determine an Input-Output Model

Selecting the inputs and outputs can be difficult. You may try to get an experienced team to consider what the important resources are for a DMU (as you defined earlier) to meet its missions. Next, how you define or measure this mission may provide a starting point for a list of your outputs. You may find certain factors that don't fall well within either category. These additional factors could be used later to explain the efficiency scores. For example, it might be interesting to see if there is a relationship between efficiency and the number of years of experience with the company. In this case, you could then use regression or other statistical techniques to examine the relationship between efficiency scores and years of company experience.

Often you will find that something that is an input is a good rather than a bad or an output is a bad rather than a good. For example, Paradi and colleagues conducted three separate DEA studies of software development projects. For outputs, they used: function points, quality, and time-to-market. The time to market output is a "bad" because increases in time-to-market should reduce the efficiency score. They converted it therefore to a "good" by subtracting each DMU's time-to-market from the maximum time-to-market in the dataset. They then used this modified time-to-market score as their output. A similar transformation was needed on the second output since quality was measured by companies as either defects or rework hours.

Also, some inputs or resources might be common across all of the DMUs such that it falls out or they may be so common as to be nonlimiting with respect to achieving the outputs. If so, you may want to consider dropping this input from the model. For example, is administrative support considered an important input? In a model of information technology departments, there may be wide variation between how the departments use administrative support, in which case it might be a useful input. On the other hand, most of the models of software development projects to date have not seen the need to include this as an input.

It is also generally a good idea to experiment with adding or removing inputs and outputs from the input-output model to determine whether and how much they affect the efficiency scores.

## D. Decide on the Specific Data Envelopment Analysis Model

It will be necessary to decide whether or not CRS hold or whether variable returns to scale is a better fit for the application. One way to answer this question is with the following two questions.

1. If a DMU doubles its inputs as defined earlier, should it be expected to double its outputs?
2. If a DMU halves its inputs, would you expect it to produce half as much output?

If the answer to both 1 and 2 is affirmative, than you should use a CRS model such as Eqs. (1) or (2). If you answer no to both questions, a VRS model may be most appropriate. There are alternatives when one of the two questions is in the affirmative and one is in the negative such as the nondecreasing returns to scale and nonincreasing returns to scale models. Dealing with this problem is beyond the scope of this introduction.

## E. Look for Relationships between Inputs and Outputs

You may find situations where there are some natural ways of aggregating inputs together. For example, if you have three budget expense categories as separate inputs, you may want to combine them into a single input. This will help reduce the dimensionality of the problem. In other cases, you may want to find a better way to model the value of inputs or outputs. For example, if you have as separate inputs the full-time-equivalents of senior programmers and junior programmers, you may want to incorporate a weight restriction so that senior programmers are considered to be at least "costly" or important as junior programmers.

## F. Collect the Data

Collecting the data is often a time-consuming step. Bear in mind the rule of thumb that you will want to have at least two or three times the number of DMUs as you have inputs and outputs, as was noted in Section IV.B. This is a rough rule of thumb and depends upon many other factors such as the level of correlation among inputs and the correlation among outputs. High correlations will tend to reduce your need for DMUs. Also, the model that you use will affect your data requirements. In general, a variable returns to scale model will need more DMUs.

## G. Perform the Analysis

As of August 2000, there are a number of special purpose software packages that can be purchased for conducting DEA. These include DEA-Solver, Frontier Analyst, IDEAS, On-Front, and Warwick DEA. In general, these can accommodate data from spreadsheets and provide a wealth of different DEA models.

## H. Examine Results

Since DEA is an extreme point technique, a single outlier can have a major impact on your results. These outliers can be very useful in what they can teach us about best practices, or they may reflect a unique and noncomparable situation that others should not be evaluated against. Even worse, it may reflect a measurement error. Therefore, it is important to carefully examine each of these efficient DMUs to determine if that seems reasonable and accurate. The generalized sensitivity analyses described by Cooper and colleagues make it possible to examine them all simultaneously in order to identify those which most affect the results of the other DMUs. You may also find that DMUs are efficient by providing an unrealistic weighting scheme. In this case, you may want to consider going back and modifying the model to incorporate weight restrictions.

Also, check the DMUs with the lowest efficiency scores to see if these scores seem reasonable. In particular, note the DMUs that they are being compared against. As a thought experiment, it can be helpful to have one person play the role of a low-scoring DMU and the other person play the role of the efficient DMU the low scorer is being compared against as can be easily done from the printouts in most DEA computer software. If the low-scoring DMU can come up with a good argument for something that is being left out of the model, this provides a good starting point for going back and revising the input-output model selected.

### I. Repeat Above Steps as Needed

It is almost always necessary to go back and repeat the process to find a better model. Carefully examining the results will help point out shortcomings in the DEA model that you have built and help in diagnosing ways to fix it.

### J. Use Results to Improve the System

One of the most important results from a DEA study is an indicator of who the most efficient DMUs are, and for each of the inefficient DMUs, an indicator to which of these DMUs they are being compared against. The inefficient DMUs can then use this as a starting point in order to examine possible best practices that better meet their situation. For example, in a study of 100 information technology departments (or DMUs) with 10 efficient DMUs, the 90 inefficient DMUs each have their own target of performance constructed out of a subset of those 10 DMUs (perhaps three or four) that are in some way similar to their operation. This now allows a more focused investigation by the manager of the inefficient information technology department of those three or four information technology departments to learn how they operate.

## V. CONCLUSION

As we struggle with the problem of measuring and evaluating the benefits of information technology projects or better ways to conduct software development projects, it has become apparent that we need better tools for analysis. DEA provides a powerful technique for evaluating complex processes and systems such as those encountered in information systems. It provides a quantitative and robust methodology for evaluating software development projects, teams, or information technology departments relying on comparison to their peers and can therefore be added to the toolkit of techniques.

## ACKNOWLEDGMENTS

This document has evolved from a web page started in 1994 on the subject of DEA. The author appreciates the many constructive comments and encouragement from colleagues over the years. In particular, this version has benefited from detailed suggestions by William W. Cooper.

## SEE ALSO THE FOLLOWING ARTICLES

Decision Making Approaches • Information Measurement • Success Measures of Information Systems

## BIBLIOGRAPHY

Ali, A. I. (1994). Computational aspects of DEA. *Data envelopment analysis: Theory, methodology and applications* (A. Charnes, W. W. Cooper, A. Lewin, and L. M. Seiford, eds.), pp. 63–88, Boston: Kluwer Academic Publishers.

Banker, R. D., Charnes, A., and Cooper, W. W. (1984). Some models for estimating technical and scale inefficiencies in data envelopment analysis. *Management Science,* 30(9): 1078–1092.

Banker, R. D., Datar, S. M., and Kemerer, C. F. (1991). A model to evaluate variables impacting the productivity of software maintenance projects. *Management Science,* 37(1).

Banker, R. D., Kemerer, C. F. (1989). Scale economies in new software development. *IEEE Transactions on Software Engineering,* 15(10): 1199–1205.

Charnes, A., Cooper, W. W., Lewin, A. Y., and Seiford, L. M., eds. (1994). *Data envelopment analysis: Theory, methodology and applications.* Boston: Kluwer.

Charnes, A., Cooper, W. W., and Rhodes, E. (1978). Measuring the efficiency of decision making units. *European Journal of Operational Research,* 2(6): 429–44.

Coelli, T., Rao, D. S. P., and Battese, G. E. (1998). *An introduction to efficiency and productivity analysis.* Boston: Kluwer Academic.

Cooper, W. W., Seiford, L. M., and Tone, K. (1999). *Data envelopment analysis: A comprehensive text with models, applications, references and DEA-solver software.* Boston: Kluwer Academic Publishers.

Fare, R., and Grosskopf, S. (1996). *Intertemporal production frontiers: With dynamic DEA.* Boston: Kluwer Academic Publishers.

Fare, R., Grosskopf, S., and Lovell, C. A. K. (1994). *Production frontier.* Cambridge, MA: Cambridge University Press.

Fried, H. O., Lovell, C. A. K., and Schmidt, S. S., eds. (1993). *Measurement of productive efficiency.* New York: Oxford University Press.

Mahmood, M. A. (1994). Evaluation organizational efficiency resulting from information technology investment: An application to data envelopment analysis. *Information Systems Journal,* 4(2): 93–115.

Mahmood, M. A., Pettingell, K. J., and Shaskevich, A. I. (1996). Measuring productivity of software projects: A data envelopment analysis approach. *Decision Sciences,* 27(1): 57–80.

Paradi, J. C., Reese, D. N., and Rosen, D. (1997). Applications of DEA to measure the efficiency of software production at two large Canadian banks. *Annals of Operations Research,* 73: 91–115.

Thanassoulis, E. (2001). *Introduction to the theory and application of data envelopment analysis.* Dordrecht: Kluwer Academic Publishers.

# Data Flow Diagrams

**Sangjin Yoo**
*Keimyung University*

## GLOSSARY

**database** A collection of data organized to service many applications at the same time by storing and managing data so that they appear to be in one location. The major benefit of using a database instead of a file system is that the system can remove the data redundancy and consistency of the data in a system.

**database management system (DBMS)** Special software to create and maintain a database and enable individual business applications to extract the data they need without having to create separate files or data definitions in their computer programs.

**data dictionary (DD)** An automated or manual tool for storing and organizing information about the data maintained in a database

**data flow analysis (DFA)** A system design and analysis approach. It is supported by a technique referred to as DFD. However, the DFD focuses more on the process, but the DFA more likely focuses on the data.

**data flow diagram (DFD)** A primary tool in structured analysis and design information systems that graphically illustrates the systems's component processes and the flow of data between them.

**entity relationship diagram (ERD)** A methodology for documenting databases illustrating the relationship between various entries in the relational databases.

**field** A grouping of characters into a word, a group of words, or a complete number, such as a person's name or age.

**file** A group of records of the same type. In the similar above example, if we collect the students' information and group that information we will make a student file.

**flowchart** A graphical design tool that depicts the physical media and sequence of processing steps used in an entire information system.

**logical design** Lays out the components of the information system and their relationship to each other as they would appear to users.

**physical design** The process of translating the abstract logical model into the specific technical design for the new system.

**programming** A stage in the systems life cycle that translates the design specifications produced during the design stage into software program code.

**record** A group of related fields. For example, last name, first name, middle name, and social security number fields are all representing a specific person. In this example the group of those fields represents or is called a record.

**structured analysis** A method for defining system inputs, processes, and outputs and for partitioning systems into subsystems or modules that show a logical graphic model of information flow.

**structured design** A software design discipline, encompassing a set of design rules and techniques for designing a system from the top down in a hierarchical type.

**structured programming** A discipline for organizing and coding programs that simplifies the control paths so that the programs can be easily understood and modified. Uses the basic control structures and modules that have only one entry point and one exit point. This concept is in contrast to the concept of event-driven programming in Visual Basic Programming Language.

**structured query language (SQL)** The standard data manipulation language for relational DBMS.

**system analysis** The analysis of a problem that the organization will try to solve with an information system.

**system design** Details how a system will meet the information requirements as determined by the system analysis.

## I. INTRODUCTION

Since at least 1960, systems analysts have faced the task of describing business processes in order to build performance-enhanced information systems. This is true whether the analyst is designing a system to automate a structured task (such as order entry) or to support unstructured objectives (such as business strategy formation).

However, system analysts, at that time, have not had the proper modeling tools for designing a business process function. Unsuccessful systems are often exposed as dramatic and expensive failures and they usually have been held responsible for the organizational blame. For these reasons, systems analysts have put considerable effort into developing ways of understanding processes that can translate directly into information system specifications. These specifications, often demanding detailed descriptions of database structures, records, and fields, are explicit and structured like the entity-relationship diagram (ERD), which is not focused on the process, but on the data to describe clearly and precisely the data with which software applications will work.

The systems analysis challenge arises from applying internal organizational process knowledge to a computer-system-recognizable knowledge in order to accomplish useful work. Based on this kind of mismatch, systems design processes are difficult to describe, for at least the following reasons:

- Processes tend to be understood differently by each person working within them (ask three different people how “the process works”; get five different answers).
- Processes sustain many variations. An order entry process, for example, may work one way for first-time customers, one way for large customers, one way for small customers, one way for former customers, and one way for customers whose purchasing agents are personal friends of your CEO’s spouse.
- Processes can prove very complex. Again, an order entry process that seems quite simple at one level proves very complex in large organizations that operate through many functional departments spread across a wide geographical area.
- An existing business process is not guaranteed to be optimally effective. This means that not everything in every organization works or that processes are perfectly fit to the organization. Thus, most managers look to information systems as a tool for fixing those problems that exist within current processes. In other words, they look for ways to use systems to redesign processes.

The management implications of systems design arise largely from *the interaction of systems design and process design,* e.g., from an in-depth understanding of how information systems can be designed to foster constructive change within organizational processes. Each influences the other: in most systems design projects, processes influence systems and vice versa. Understanding these issues at more than a superficial level can greatly improve the success of any business design, process design, or system design ideas that you have.

Most traditional system design techniques do not arise from perspectives that managers might typically use to describe a business. Instead, they are developed from a systems engineering perspective: techniques were evolved by software engineers who were trying to understand business processes in order to build information systems that worked.

This point also might seem obvious, at least until you try to build a systems prototype of any meaningful size. In such a project sooner or later a “management” perspective (e.g., a focus on the business demands associated with the process in question) will begin to conflict with an “information systems” perspective (e.g., a focus on the demands imposed by the technology that information engineers are trying to use to support the process). The business demands of the process (e.g., to deliver a product or service quickly and without error, based on often highly ambiguous customer preferences) tend to oppose the requirements of an information system (which can process data extremely quickly, but must receive that data in a highly structured, unambiguous form). For example, according to the evolution of the global connected on-line real-time environments, most customers are very sensitive to their supplier’s on-line security and privacy concerns. If customers try to track their products via the Internet, the information system of the company should check the customers' identities before supporting the customers' requests through the Internet or other telecommunication technology. However, there are lots of ways to identify the customer himself utilizing fingerprint, ID, Password, etc. In this case, the managers want to provide any kind of services their customers want regardless of the system support ability, while restricting identification methods to their customers use due to instability of information technology. Those two parties should resolve the problem between the managerial and technical gap in system design and analysis. In database management systems (DBMS), most traditional DBMS such as relational, hierarchical, etc. cannot support the logical data relationship. For example, if a user inputs several constraint conditions of data, the DBMS cannot provide the correct result even though it uses a complex SQL query method, because those inputs do not have a specific range, instead having an ambiguous range of value in the case of finding the shortest path or goal programming of the management science theory.

The effect of these conflicting perspectives is illustrated in Fig. 1. It suggests that many characteristics of existing business processes can be left ambiguous, especially with regard to how those processes achieve explicit business goals.

Information systems inherently micromanage: the technology constraints imposed by hardware and database design make it impossible for them to do otherwise. It sounds overly obvious to point out that databases must have fields and those fields must be defined—but when information systems are being used to support cross-functional processes that extend beyond any but the most operational levels, the impact of such requirements forces systems analysts to reach a level of process description that many managers never attempt. The input, processing, and output requirements of information systems drive a need for understanding process characteristics at multiple levels and in great depth.

From this point of view the success of systems design hinges on finding techniques that effectively reconcile an organization's naturally ambiguous understanding of the processes that achieve strategic goals with an information system's naturally structured approach to handling data. One approach to resolving this conflict involves understanding the structures inherent in business data, and gave rise to entity-relationship (ER) modeling. Another complementary perspective focused on how data move through the tasks that make up business processes. This approach has come to be called data flow analysis (DFA), and is supported by a technique referred to as data flow diagramming (DFD). However, the DFD is more focused on the process not on the data that the DFA tries to focus on.

## II. FUTURE DIRECTION OF DFD METHODOLOGY

About twenty years ago, systems analysts—Gane and Sarson, DeMarco—began to resolve the conflicts between managerial and information systems design perspectives in an elegant way. This solution described processes by focusing almost exclusively on the *data* that each process generated (→ data-oriented process activities). This notion suggested that by focusing on data movement and the process*ing* activities that occurred within a business process, one could simplify process descriptions in ways that supported successful information systems. To a large extent, this idea proved valuable: systems analysis techniques based on this perspective now dominate most software systems engineering work. Focusing on the data had the following advantages:

1. A focus on data minimized the distractions generated by other process characteristics. For

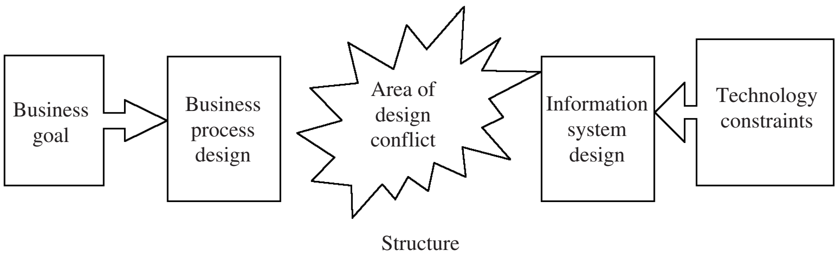


**Figure 1** Conflicts inherent in business process design and information system design. See http://roger.babson.edu/Osborn/doit/readings/dfdsumry.htm.

example, knowing what data an order entry process required enabled system designers to build systems that would support order entry no matter who did the work, no matter where the work was done, and no matter when it was done. The system design, from this perspective, depends less on attributes of the existing process than on the data within the process. In this sense, a focus on data alone enabled systems analysts to move quickly beyond process details to recognize the minimum necessary characteristics of a process that their systems would have to support.
2. A focus on data made it easier to recognize potentially new process designs. Abstracting a general picture of the data used in a process from the specific details that represented examples of that process tended to provide new insights for process redesign. Consider the ordering example we discussed above. In the more complex version of that process, accounting matches two types of purchase orders with a shipping manifest. A systems analyst using data-driven techniques would quickly recognize that all three forms carried largely the same data, ask why they were all needed, and ask why the matching step needed to take place. In organizations where Accounting waited days or weeks to obtain copies of all three forms prior to sending out an order (not really so unlikely in many businesses), such a process innovation could have important business benefits (e.g., shorter delivery times).

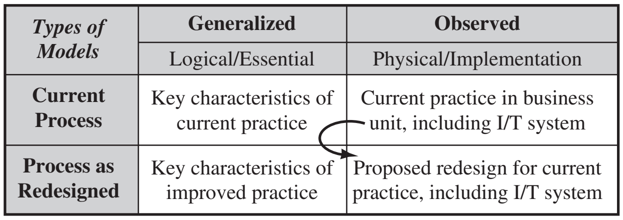
In effect, focusing on data enabled systems analysts to extract only those details most necessary to the development of the information system from the masses of detail collected to describe organizational processes. This simplified process view often offered insights for potential process redesigns. Insights arose for at least two reasons:

1. Developing software-based process support forces analysts to examine the logic by which organizations make choices within processes. Electronic support for order entry, for example, requires that a system "know" how orders are officially approved, how products are confirmed to be backordered, how packages are approved for shipment, etc.
2. Developing the algorithms (e.g., the specific instructions) that software can use across a range of process variations forces analysts to generalize generic process characteristics from observed process specifics. Orders from both large and small customers, for example, are likely to have features in common as well as specific differences based upon customer size: an efficient software system would know how to handle common characteristics using the same software code, thereby minimizing the amount of customized code that would have to be produced to manage process variations.

This interaction between process descriptions and the requirements of software engineering led to a standardized way of analyzing business processes. This perspective understood processes from the point of view of the data and processing steps that they generated. It led to a four-step analysis of data flows and their related processing steps:

1. Data flows and processing steps were observed within the process as it was currently practiced in the organization. The resulting model was often referred to as the "physical" or "implementation" model of the system.
2. The underlying logic represented by observed process steps was abstracted to build a generalized model of the process. This model was often referred to as the "logical" or "essential" model of the system. "Logical," in this sense, referred to the programming logic that analysts would have to use to write the software code needed to build the system.
3. After closely examining patterns of data flows and processing that emerged from the physical and logical models, analysts could often suggest ways of executing process logic more effectively. In other words, they could suggest process improvements. The logical model of these improvements would comprise the key characteristics of a new process that, if implemented, would enable improvements in business performance based on a combination of process redesign and newly available information system support.
4. If management decided to adopt the new process, the combination of process design and information system support would become the basis of new practice within the organization, giving rise to a new set of physical process models.

A summary of this approach to process analysis is shown below in Fig. 2. It suggests how a systems analyst could analyze data flows *(DFA-perspective)* to move from observations of current practice in an organization to understanding (1) key characteristics of current prac-



| *Types of Models* | Generalized | Observed |
|---|---|---|
| | Logical/Essential | Physical/Implementation |
| **Current Process** | Key characteristics of current practice | Current practice in business unit, including I/T system |
| **Process as Redesigned** | Key characteristics of improved practice | Proposed redesign for current practice, including I/T system |

**Figure 2** Types of data flow process models: physical, logical, and process redesign. See http://roger.babson.edu/Osborn/doit/readings/dfdsumry.htm.

tice, (2) key characteristics of improved practice, and (3) potential redesigns that could become future practice to support improved business performance. The curved arrow in the figure suggests the analytical progression that would support such conclusions.

We will refer to the analytical perspective that focuses on data movement and the processing implications of work tasks as DFA. The following sections describe a graphical approach to understanding data flows that provides a way to build data flow models quickly and consistently. These models offer a useful way to compress large amounts of process information into a two-dimensional space that assists in both understanding the logic of a process and identifying key data entities within the process. For this reason, DFA can be seen as a technique complementary to ERD. Indeed, the DFA advanced data flow diagrams methodology, can be seen as a data-driven process representation that is one step further away from the database-design-specific characteristics of ERD.

In conclusion, while DFDs do depict data, they are more aptly described by most authors as being process oriented. In DFDs, data is described/shown in chunks (data stores) and the structure and details of the data are not the primary focal point in a DFD, even though they are part of techniques related to DFDs such as the data dictionary (DD). In DFDs, processes are the focus of decomposition and refinement and hence the notion of DFDs being process oriented. The data flows to and from the processes are indeed important, but processes drive the construction and use of the diagram. At most, it is reasonable to say that DFDs are both process and dataflow driven. While DFA is a tool that analyzes the data movement or patterns in a process, ERD is more focused on the data not on the process utilizing the DBMS in most cases for analysis, design, and implementation of the data-oriented information systems (Fig. 3).

## III. BRIEF OVERVIEW OF THE DFDs

In this section we will briefly overview the general characteristics of DFD. Data flow diagrams are a network-style (or hierarchical) representation of a system. They are the cornerstones for structured system analysis and design. They provide one technique for isolating the data stores used by a process and the major data entities that those stores contain.

In short, DFDs offer a graphical technique for summarizing the movement of data between the processing steps that occur within a business process. They isolate the collections of data, or data stores, which accumulate during a process, and identify the sources and/or destinations of data that arise outside process boundaries. In addition, there are various DFD drawing CASE tools such as PowerDesigner (PowerSoft, Inc.), Data Explorer (IBM), Khoros, Wavefront Data Visualizer, Edge Diagrammer (Pacestar Softwave), and SilverRun BPM.

### A. Key Characteristics of DFDs

Some key characteristics of DFDs are

- Two-dimensional summary. DFDs offer a way to summarize the data flow characteristics of a process on a single page. As such they can provide a useful and concise summary of a system, which can be described by a single business process-oriented picture.
- Completeness. DFDs offer a way to check the completeness of your process model, particularly

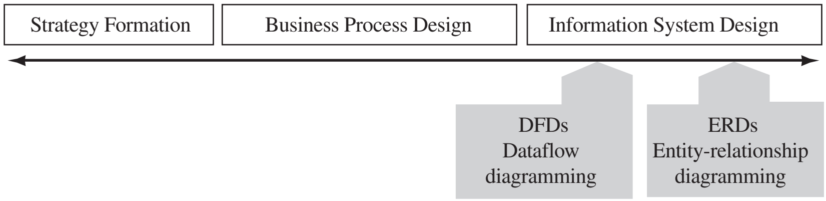


**Figure 3** Complementary data-driven system design techniques and process design. See http://roger.babson.edu/Osborn/doit/readings/dfdsumry.htm.

regarding your understanding of the data that would be required by an information system and process flow steps within the system (e.g., Are all the data that would be needed for input actually available? Does each processing step produce data that could be used by subsequent steps? Are all data generated usable and the consequence processes effective by an information system where necessary?). DFDs can provide a fast way to generate further questions that need to be asked about the process.

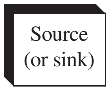
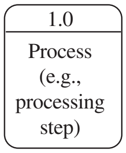
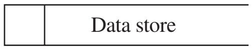
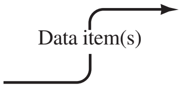
- Process*ing* not processes. DFDs refer to "process" steps. It might be more useful to think of DFD "processes" as process*ing* steps rather than process activities. In essence, DFDs ask one to refer to the information systems implications of any processing work that occurs during the tasks that comprise a business process. DFD terminology tends to confuse the term "process" in its connotation with *business* process with the term "process" that refers to a *computational* process executing within software (e.g., a software algorithm). Whether this represents the presumption among information engineers that *everything* is just a version of a computational process; the point here is that it is safer to think of DFD "processes" as processing steps.
- Patterns. DFDs can provide shorthand for understanding patterns that exist within the data flows supporting business processes. They can show, for example, where large amounts of data are collected, stored, transferred, generated, used, and delivered. They can highlight areas of potentially extraneous activity, and can suggest process components that do not receive the information support that they deserve (or need).

## B. Symbols Used in DFDs

Table I summarizes the graphical symbols used in DFDs. The vocabulary used by DFDs is very simple, comprising only four symbols such as data flows, processing steps, sources/sinks, and data stores to represent any system at any level of detail.

An example can probably provide the best way to understand how a DFD can summarize knowledge about a process. The diagram in Fig. 4 illustrates how a systems analyst might use a DFD to describe data flows, processing steps, sources, sinks, and data stores

**Table I** Symbols Used in Data Flow Diagrams (DFDs)

| Symbol | Description |
|---|---|
|  | **Source** (or **Sink**). Source is short for "data source"—a source represents any source of data that is identified as outside the boundary of the process that the DFD is modeling. Similarly, a "sink" is any destination for data that is outside the boundary of the process that the DFD is modeling. |
|  | **Processing step.** A DFD *process* represents an activity that processes data; presumably the processing is important enough to play a significant role in the business process that the DFD is modeling. This processing focus is relevant even when the description of the process does not sound like data processing—e.g., "withdraw funds from account" in a DFD describing banking actually refers to the processing that must accompany the withdrawal.<br>Note that the process is labeled with an identifying number—in this case, 1.0. Thus number is used in developing levels of **DFDs** that describe increasingly detailed levels of a process (e.g., process 1.0 breaks apart into three processes labeled 1.1, 1.2, 1.3, etc.). |
|  | **Data store.** A collection of data needed by the business process. A data store is *not* the same thing as a "database"—instead, it represents a more abstract way of referring to any accumulation of data that is used by the process. The contents of a data store will very likely suggest some databases that could be derived from that store, but data stores tend to be process-specific rather than database-specific. One of the main purposes for which we will use **DFDs** is to identify entities within the contents of data stores. Those entities then become the basis for database tables developed using **ERDs.** |
|  | **Data flow(s).** In **DFDs,** data flows are represented by arrows. The arrows are labeled with the data that move along the arrow (the arrowhead indicates the direction of movement). |

See http://roger.babson.edu/Osborn/doit/readings/dfdsumry.htm.

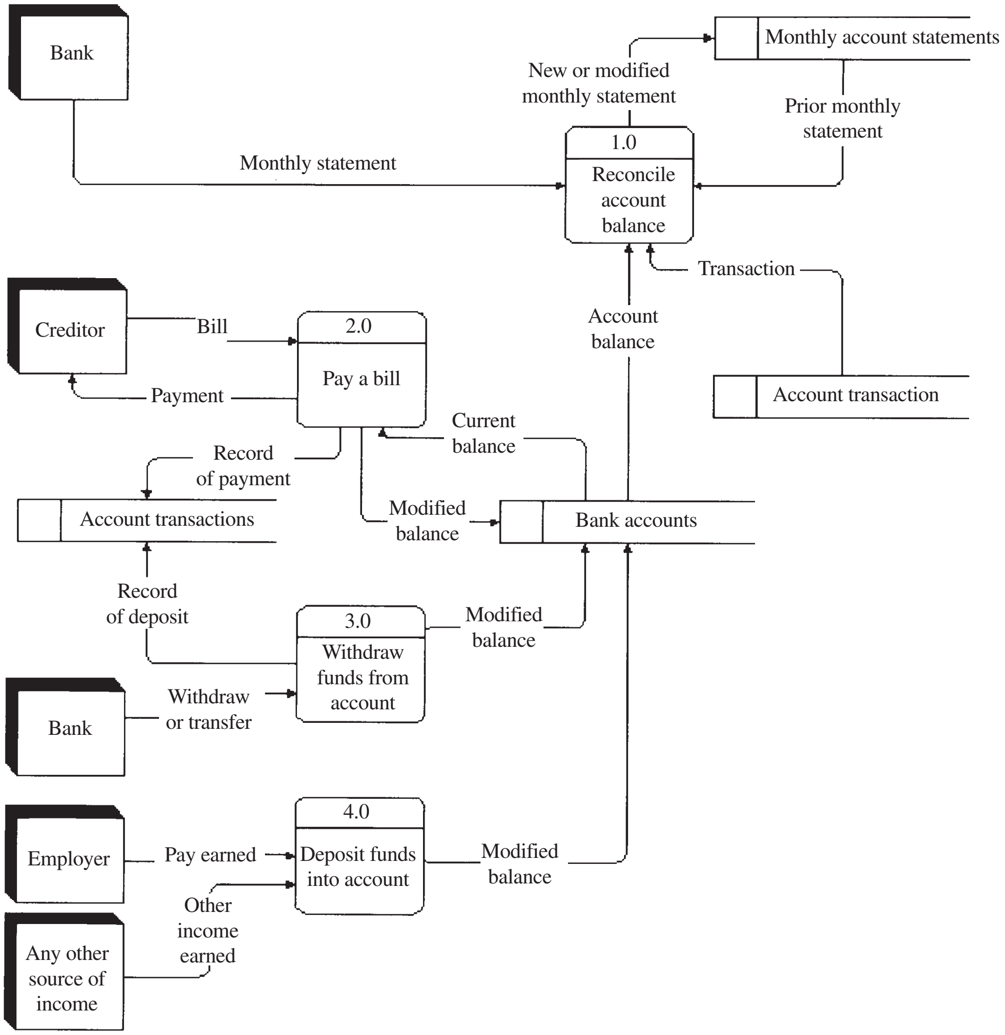


**Figure 4** A sample data flow diagram—bank operations. [Adapted from Whitten, Bentley, and Barlow. (1994). *Systems Analysis and Design Methods,* 3rd ed. Burr Ridge, IL: McGraw-Hill].

used to describe an individual's relationship with his or bank. The DFD describes the processing transactions that take place as the individual and the bank jointly manages a set of customer accounts. The process(es) described include most operational aspects of handling the customer's money, including making deposits, funding withdrawals, paying bills (perhaps a service provided by electronic banking), and reconciling account balances. Some of the data stores can be translated almost directly into database specifications (e.g., "account transactions"). Others show how the notion of data stores can apply equally well to data that are captured in paper form (e.g., "monthly account statements"). Throughout, the data flow arrows identify data items and groups of data items that move between sources, data stores, and processing steps. Note that processing steps represent those points at which data from sources and data from data stores come together, resulting in some change in specific data items maintained by the bank.

## C. Advantages of DFDs

Here are some of the advantages of using DFD analysis.

- *Data flows and process consequences.* Note how this representation of the data characteristics of banking operations enables us to start at any point in the operation (e.g., deposits, withdrawals, or bill payment), and follows the consequences of that activity through to the point where all appropriate account balances have been adjusted and reconciled. Wherever we start in the process, we can understand the processing steps that the bank would need to take to complete the relevant transaction(s) and to inform its constituents of the results.

- *Data inputs and outputs.* The DFD also makes it possible to understand what data are needed to provide appropriate inputs to any processing step. If, for example, we were to build an information system to support this individual's banking activities (in the days before Quicken and/or Microsoft Money), we would need to understand exactly what data items are represented by data flows such as "Monthly Statement," "Pay earned," "Withdraw or Transfer," and other arrows shown in the diagram.
- *Simplifying complexity by isolating process components.* Note how the DFD would make it easier to capture the detail of such data flows. By isolating "Withdraw or Transfer" within the larger scheme of the banking process, the DFD makes it possible to consider the details of the data items included in this flow without reference to the flows affecting other processing steps. All of the flows affecting withdrawals (e.g., processing step 3.0, "Withdraw funds from account") are isolated as entering or leaving processing step 3.0. At the time that DFDs were developed, this shift toward modularizing data flows and processing elements represented a major step forward in enabling systems analysts to add useful structure to process representations rapidly and easily.

### D. Three Approaches to Refinement of DFDs

1. Start by identifying the input and output data flows (arrows), and the top-level data transformer (bubble). Decompose the data transformer and specify the input and output data flows for each of the components.
2. Start with the output data flow from the system and try to identify the final transformation that has to be done to attain that data flow. Then try to identify the previous transformer, etc.
3. Same as method 2, except that we start from the input flow to the system and work out the sequence of transformations that should be applied to it.

## IV. COMMON TYPES OF DFD MISTAKES

DFDs look easy on the surface. After all, what's hard about writing down a few bubbles and arrows? In practice the techniques proves to be somewhat more difficult than one might initially anticipate. Obtaining appropriate names for both processing steps and data flows can require careful thought. As one rule of thumb, imagine that you are producing a diagram that must pass this test: you will finish the DFD, then hand it to someone (of reasonable intelligence) who will then proceed to describe the process back to you based upon what he or she sees in your diagram. If this process recitation captures your original process description (and, of course, the appropriate characteristics of the business process itself), your DFD is reasonably accurate.

With such problems in mind, this section considers some of the common mistakes that occur when one first tries to build DFDs.

Simply said, the DFD must focus on the flow of data, not the control of data. Every DFD should consider following basic rules: every symbol is labeled, no processes lack input data flow, no processes lack output data flows, data does not flow directly between sources, sinks, or stores, and data flows entering and leaving lower level diagrams match those entering and leaving parent processes. The following section will look at common mistakes in drawing DFDs more detail.

### A. Illegal Data Flows

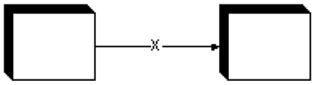
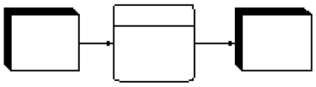
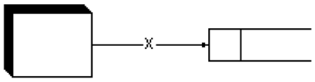
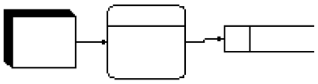
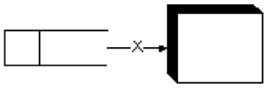
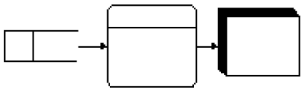
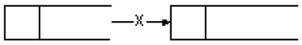
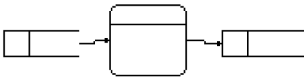
One of the patterns of DFA is that all flows must begin with or end at a processing step. This makes sense, since presumably data cannot simply metastasize on its own without being processed (in spite of the circumstantial evidence we might have gathered in our own business experience). This simple rule means that the following mistakes can be fairly easily identified and corrected in a DFD. The symbols shown below are purposefully left blank (e.g., without captions) so that it is easier for you to recognize patterns such as these in your own DFDs (Table II).

### B. Diagramming Mistakes: Black Holes, Gray Holes, and Miracles

A second class of DFD mistakes arises when the outputs from one processing step do not match its inputs. It is not hard to list situations in which this might occur.

- A processing step may *have input flows but no output flows.* This situation is sometimes called a *black hole.*

**Table II** Common Data Flow Diagramming Mistakes

| Wrong | Right | Description |
|---|---|---|
|  |  | A source or a sink cannot provide data to another source or sink without some processing occurring |
|  |  | Data cannot move directly from a source to a data store without being processed |
|  |  | Data cannot move directly from a data store to a sink without being processed |
|  |  | Data cannot move directly from one data store to another without being processed |

Adapted from Figure 9.9, p. 360 in Whitten, J. L., Bentley, L. D., and Barlow, V. M. (1994). *Systems Analysis and Design Methods* (3rd Ed.). Burr Ridge, IL: Irwin.

- A processing step may *have output flows but no input flows.* This situation is sometimes called a *miracle.*
- A processing step may have *outputs that are greater than the sum of its inputs,* e.g., its inputs could not produce the output shown. This situation is sometimes referred to as a *gray hole.*

When one is trying to understand a process during the course of an interview (and consequently drafting DFDs at high speed), it is not hard to develop diagrams with each of the above characteristics. Indeed, scanning DFDs for these mistakes can raise questions that provide questions for use in further process analyses (e.g., "Where do you get the data that allows you to do such-and-such...").

The following diagram (Fig. 5) illustrates these common DFD mistakes. By tracing the inputs and outputs affecting each processing step, you can avoid them in your own diagrams.

## C. The Differences between DFDs and Flow Charts

The DFDs are not flow charts. A last class of DFD mistakes is somewhat more difficult to identify. Many of us have had prior experience developing flow charts. Flow chart diagrams can be useful for describing programming logic or understanding a single sequence of process activities. It is important to recognize, however, that DFDs are *not* flow charts. Flow charts often show both processing steps and data "transfer" steps (e.g., steps that do not "process" data); DFDs only show "essential" processing steps. Flow charts might (indeed, often do) include arrows without labels, while DFDs never show an unnamed data flow. Flow charts show conditional logic and DFDs don't (the conditional decisions appear at lower levels, always within processing steps). Flow charts show different steps for handling each item of data, while DFDs might include several data items on a single flow arrow.

With these distinctions in mind, the following diagrams suggest some DFD drafting mistakes that might be influenced by prior experience with flow charts (Fig. 6).

In the example above, a processing step is included that does not actually change any data. This step (titled "route transaction") might appear on a flow chart but would not appear on a DFD.

In Fig. 7, the left side tries to represent the disposition of credit receipts after a credit card purchase has been approved. Branching, whether relating to data or to conditional decision-making, might appear on a flow chart but would not appear on a DFD.

How does one decide what goes on a DFD? One answer lies in understanding the difference between a physical and logical model of a process. The logical model describes only those processing steps that are essential to completing the process. These may not be

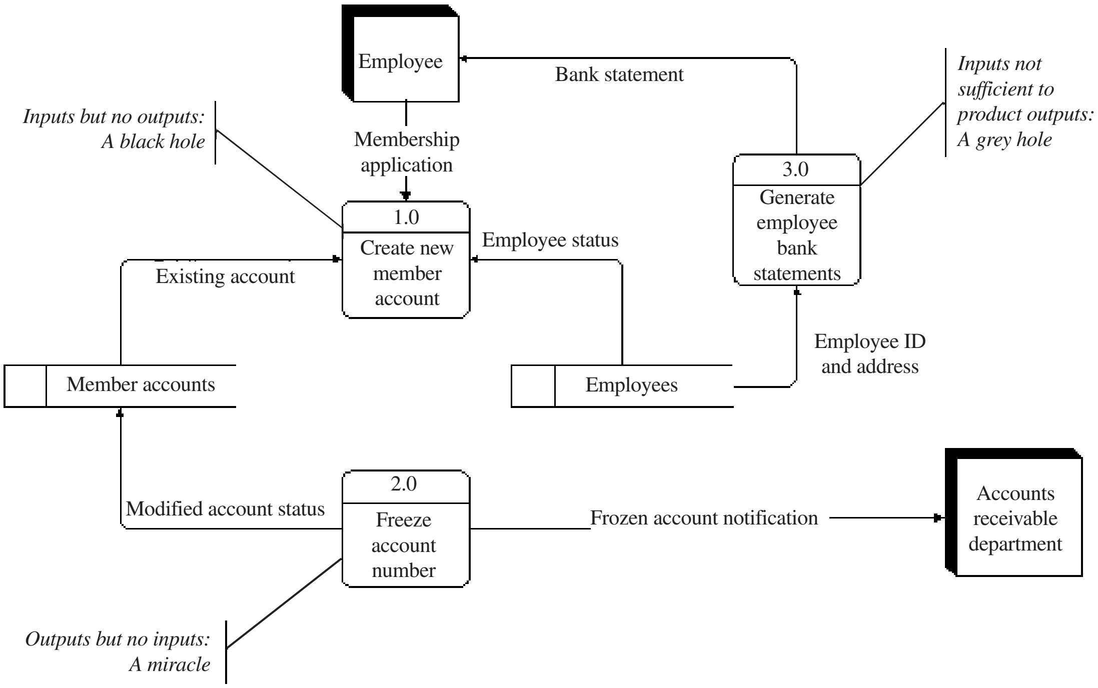

Employee
Bank statement
Inputs not sufficient to product outputs: A grey hole
Inputs but no outputs: A black hole
Membership application
3.0
Generate employee bank statements
1.0
Create new member account
Employee status
Existing account
Member accounts
Employees
Employee ID and address
2.0
Freeze account number
Modified account status
Frozen account notification
Accounts receivable department
Outputs but no inputs: A miracle

**Figure 5** Common DFD drafting errors: black holes, gray holes, miracles. [Adapted from Figure 9.5, p. 356 in Whitten, J. L., Bentley, L. D., and Barlow, V. M. (1994). *Systems Analysis and Design Methods* (3rd Ed.). Burr Ridge, IL: Irwin.]

immediately obvious during early steps in process analysis, so be prepared to sketch multiple drafts of a DFD. One of the reasons that this technique was developed was to enable systems analysts to sketch meaningful process descriptions on a single piece of paper during discussions with business managers (even an envelope or a napkin, no kidding). The technique is *designed* for rapid diagramming and multiple iteration. Don't be dismayed if your first draft has mistakes—those mistakes are one of the ways that you know how to ask more insightful questions of the process.

As a final aid in developing DFDs, consider the following description of processing steps (Table III). It

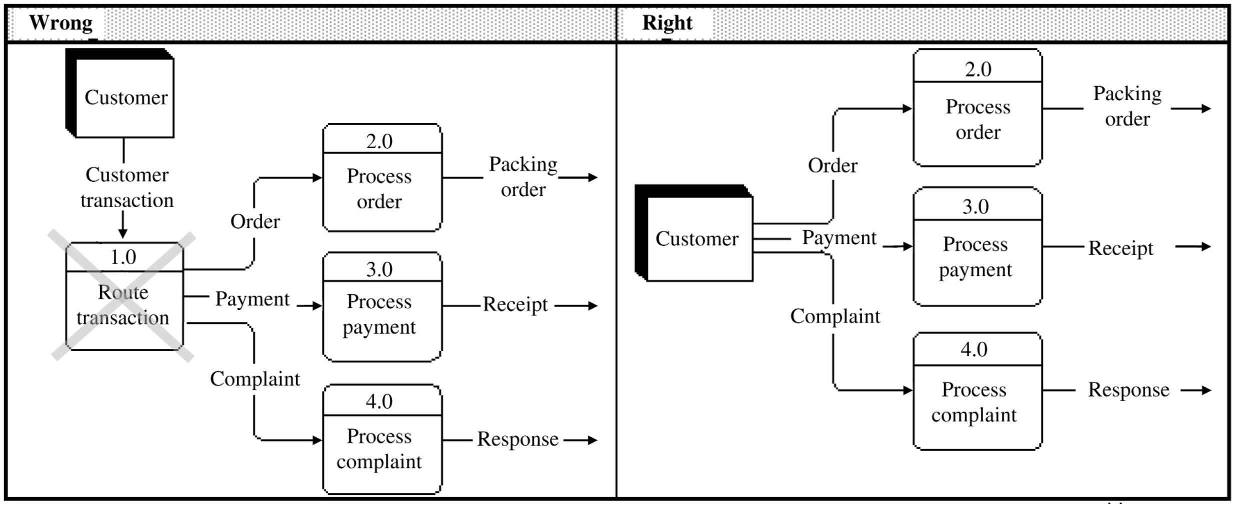


**Figure 6** Processing steps that do not change data do not belong in DFDs. [Adapted from Figure 9.6, p. 357 in Whitten, J. L., Bentley, L. D., and Barlow, V. M. (1994). *Systems Analysis and Design Methods* (3rd Ed.). Burr Ridge, IL: Irwin.]

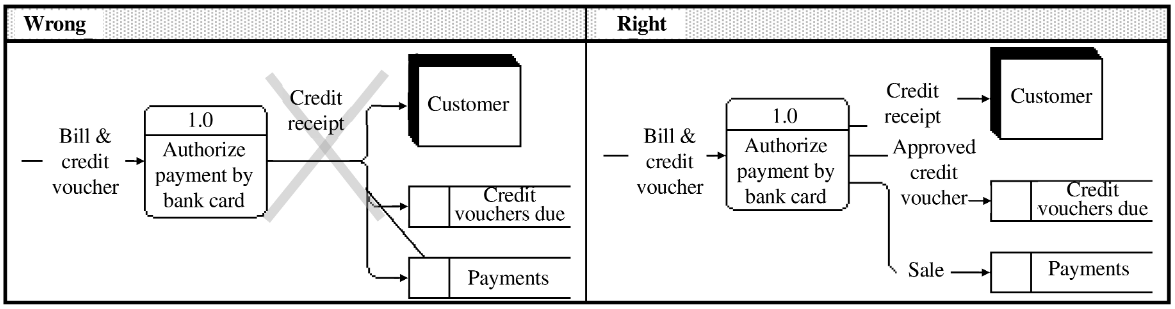


**Figure 7** Conditional/diverging data flows should be replaced by individual flows. [Adapted from Figure 9.8, p. 359 in Whitten, J. L., Bentley, L. D., and Barlow, V. M. (1994). *Systems Analysis and Design Methods* (3rd Ed.). Burr Ridge, IL: Irwin.]

suggests five characteristics of the processing step that changes data (and so deserves to be included on a DFD).

## V. LEVELING OF DFDs

### A. Definition of DFD Leveling

If we have hundreds of system processes that to be drawn on a single paper, it is a very tough job even though you can draw it clearly on a paper. Furthermore, hundreds of processes on a single document become incoherent, make it difficult to follow each process, and/or are clearly no better of than using an essay. To resolve these kinds of problems, we can use a set of leveled DFDs based on the top-down partitioning (or known as hierarchical) and "decomposition principle." In short, processes are decomposed by exploding them into lower and lower levels until they are ready for detailed design. In other words, a complicated problem or a single design domain can be chopped into a very simple unit domain that can be easily understood and maintained by the system analysts or users. DFDs should have between three and nine processes on a single level. Detailed designs of atomic processes are specified in descriptions called "mini-specs," usually in pseudo-code. Furthermore, detail that is not shown on the different levels of the DFDs such as volumes, timing, frequency, etc. is shown on supplementary diagrams or in the DD. For example, data store contents and/or definition of terms, symbols, etc., may be shown in the DD.

DFD-leveling refers to using multiple pages to show data flows at varying levels. For example, one page of a DFD might show processing steps 1.0, 2.0, and 3.0. A second page might show the subprocesses within step 1.0: it would have the same total inputs and outputs as 1.0, but would show an additional level of detail in describing processing steps 1.1, 1.2, and 1.3 that make up process 1.0. A third page might show steps 2.1, 2.2, and 2.3; a fourth page steps 3.1, 3.2, 3.3, etc. When decomposing data in this way, it is important to make sure that the

**Table III** **Processing Steps to Be Considered in Drafting DFDs**

| Include processing steps that | Example(s) |
|---|---|
| Perform computations | A processing step that develops charges associated with a product or service, e.g., "price consulting engagement" |
| Make decisions | A processing step that qualifies a potential customer as a good prospect based on demographics, income level, and the number of times that the individual has responded to company product trials |
| Split data flows based on content or business rules | A processing step that separates approved orders from rejected orders based on credit rules |
| Filter and/or summarize data flows to produce new data flow(s) | A processing step where specific data items may not change, but the structure of the data does, e.g., filtering invoice data to identify products that were in highest demand during the past two weeks |

Adapted from p. 355 in Whitten, J. L., Bentley, L. D., and Barlow, V. M. (1994). *Systems Analysis and Design Methods* (3rd Ed.). Burr Ridge, IL: Irwin.

lower level DFD receives exactly the same "incoming" data flow as the higher level process which it describes, and passes the same "outgoing" data flow as that process.

## B. Assigning Names on the DFD

The DFD symbols identify relationship among components of a system. For a DFD to complete its mission as a communication vehicle, these components must be given clear and meaningful names that support the description of the system. The naming of system components should follow these clear, simple rules:

- Data flow and data stores should receive names that describe the composition of the data flowing through the system with nouns.
- Process symbols should be named with strong verbs to indicate the basic transformation or process that is taking place.

Care must be taken to find names that accurately reflect the data and the processing involved. Difficulty in finding names is often a sign of a more serious problem: it may indicate a lack of understanding about what is happening. More detailed information may be necessary before the diagram can be completed.

## C. Context Diagram

The starting point is the context diagram, which, in effect, defines the scope of the system. It highlights the net inputs and sources as well as the net outputs and destinations (sinks) of data for the system.

In this level, system analysts or developers define the scope of the system using *context diagrams,* which offer a way to describe the scope of a DFD by identifying (at its highest level) the boundaries of the system. The context diagram is a special kind of DFD.

If we build a data flow model of a system in a top-down fashion, we start with a context diagram, in which the system is represented as one process. Furthermore, the context diagram shows all external entities, which interacted with the system and all data flows between these and the system. The purpose of the context diagram is to depict how the system is connected to, and interacts with, other entities that make its data environment. It especially gives the system context on the diagram where the boundaries will be between the system and the rest of the world. Before getting started on level 0 diagram, remember the need to focus on major business operations or occurrences rather than on specific, narrow, physical processing functions.

We can illustrate the leveling process using an automatic ordering example. The automatic ordering system provides a quick, efficient service to customers by allowing for almost instantaneous order and dispatch. The only entities "outside" the system (who are in contact with the organization using the system, but not part of it) are the customer and the credit supplier (a credit card company). The example is shown in Fig. 8.

## D. Top-Level/Level 0 DFD

Top level diagram (or level 0 diagram) provides a more detailed description of the system (Fig. 9). It is used to describe, at a high level, the overall processing in the system. The scope of the system presented in this level remains the same as in the context diagram. The difference is that the single and central process, the entire system, will be partitioned into a series of components, or major subsystems within overall system.

This level of DFD usually shows the major processes such as departments or critical functions. It still has the same external entities (sources and sinks). There are several hints to start with this level. First, make a list of the major data stores currently in use. Second, list the major business occurrences or events within the system. Third, draw a segment, or fragment, of a DFD for each of the identified events. And the final step in the preparation of a level 0 diagram is to assemble the fragments into a single DFD.

## E. Level 1 DFD

This level expands the processes in the level 0 diagram basically. When constructing DFDs at any level, it is good practice to name the data flow first, then to name

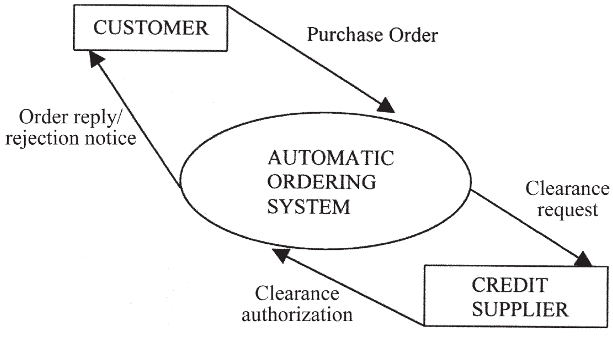


**Figure 8** A context diagram.

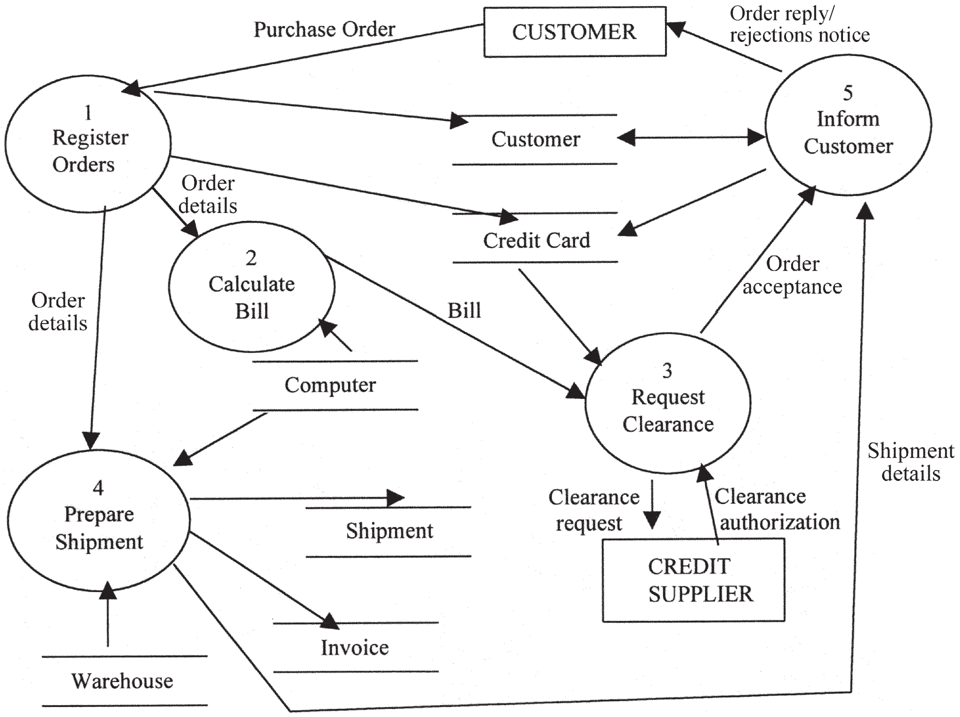


**Figure 9** The top-level DFD.

the processes (Fig. 10). This helps to keep the concentration on the flow and transformation of the data.

## F. Higher Level DFD

The level 2 DFD expands the processes in the level 1 diagram. The level 3 DFD is an expansion of the process in the level 2 DFDs. The level 4 DFD is an expansion of the process in the level 3 DFDs. And the level 5 DFD is an expansion of the process in the level 4 DFD etc. Levels stop when the analyst feels enough detail has been reached. All processes may require different levels of abstraction. In other words, eventually we will decompose the DFD diagram to a point where the DFD cannot be decomposed any further. This point is usually reached when the processes themselves only describe simple tasks. When we reach this point, the contents of the process “bubble” need to be described. In Fig. 10, such a point has been reached.

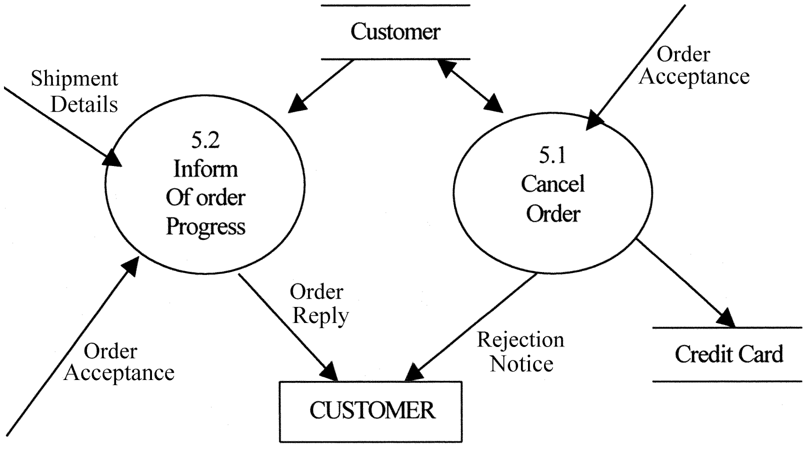


**Figure 10** Level 1 DFD: simple processes.

## VI. SUMMARY

Because of simplicity and clarity, DFDs are very effective for enhancing communication between users and systems designers. In addition, the resulting models form a good foundation from which to begin design work. DFDs stress processes or transformations that are applied to data. They do not emphasize data access paths, nor do they emphasize control and timing for events. In fact, they hide these items. This is good, since it enhances the ability of the analyst to discover redundancies and inaccuracies in the way data are processed by the current system and eliminate them from the new system. However, it means that DFDs *alone do not constitute a complete model,* even for analysis. Supporting documentation—in the form of the DD, process-descriptions (or often-called, mini-specs), and some narrative—is necessary.

If DFDs become too complex, it becomes difficult to trace data flows and transformation and their purpose (simplicity) is defeated.

Data flow diagrams emphasize functionality and movement of data. Functions and processes are isolated, and the data exchanged between them are specified in detail. Data flow diagrams are thus process or function oriented.

Data flow diagrams can assist in:

1. Isolating the component parts of a business process, reducing the analytical complexity involved in determining the specifications that process support software would have to meet
2. Shifting the focus of a process description to the data flows and processing steps that the process represents
3. Identifying data-related process characteristics that could be candidates for process design improvements
4. Identifying data stores that isolate entities that could be further developed using ER analysis

## SEE ALSO THE FOLLOWING ARTICLES

Database Administration • Database Systems • Flowcharting Techniques • Structured Design Methodologies • Structured Programming • Structured Query Language (SQL) • Systems Analysis

## BIBLIOGRAPHY

Fichman, R. G., and Kemerer, C. F. (October 1992). Object oriented and conventional analysis and design methodology—Comparison and critique. *MIT, IEEE Computer,* pp. 22–39.

Kozar, K. A. (Spring 1997). Representing systems with data flow diagrams. Available at http://spot.colorado.edu/~kozar/DFD.html.

Laudon, K. C., and Laudon, J. P. (2000). *Management information systems: organization and technology in the networked enterprise,* 6th Ed. Upper Saddle River, NJ: Prentice Hall.

Terence T. O. (August 1998). Dynamic modeling and performance analysis of information systems. Available at http://www.sbaer.uca.edu/docs/proceedingsII/98dsi0822.htm.

# Data, Information, and Knowledge

**Eduardo Gelbstein**
*International Computing Centre, United Nations*

## GLOSSARY

**data** Facts and figures that can be used as a basis for reasoning, discussion, or calculation. Data, particularly raw data collected from observations and experiments, may include redundant or irrelevant items as well as useful facts and figures, and must be refined to be meaningful.

**data resource management** A discipline the purpose of which is to define what data is available and where it can be found, in addition to defining controls and quality mechanisms.

**information** Data (possibly from various sources) presented in a context that is new or pertinent to a specific situation.

**knowledge** Often described as the ability to use information so as to be able to do something with it.

**metadata** The description of the characteristics of data.

The growing availability of measurements, databases, texts, images and other records in digital form, together with the extent to which networks such as the Internet have been deployed, provide humanity with access to volumes of data and information with precedent in human history.

Sections of this article present definitions, differences and relationships between data, information, and knowledge as well as a discussion as how data and information can be used.

This article also presents a discussion of quality issues and some of the dilemmas for the Information Age.

## I. INTRODUCTION

Man has left records of his observations of the world around him for at least 30,000 years. Early records include cave paintings and inscriptions in clay and stone dealing with mundane topics such as commercial grain transactions and ownership of land. From these early records, the history of writing progressed from clay and stone to papyrus, paper, books, and maps, and went beyond language to include music (the phonograph), images (photography), and digital techniques which now encompass all forms of records.

The processes of observation, analysis, documentation, and collection have continued ever since. As literacy spread, science became established as a discipline, and data, information, and knowledge began to accumulate at an ever-growing pace that continues to accelerate.

By the latter half of the 20th century, humanity had access to volumes of data and information much greater than at any other time in human history. The way in which a knowledge worker used these "materials" has more in common with the work of a craftsman than with that of an assembly line operator in the Industrial Revolution. Those who use data and information need to have a very good knowledge of their tools and must also have a thorough understanding of what to do with these "raw materials."

Editors, publishers, librarians, web site designers and other professionals have created and continue to create collections out of this raw material to facilitate the search and use of such material. By their very nature, these collections imply choices made by their creators and in many instances researchers may prefer or need to have access to the original source for that data or information.

People working with data and information do so for a variety of purposes, transforming the data and information for re-use and distribution beyond the boundaries of their workplaces, and, often, beyond geographical boundaries as well.

## II. CONCEPTS AND DEFINITIONS

Data is defined as facts and figures that can be used as a basis for reasoning, discussion, or calculation. Data, particularly raw data collected from observations and experiments, may include redundant or irrelevant items as well as useful facts and figures, and must be refined to be meaningful.

Data resource management (DRM) is a discipline the purpose of which is to define what data are available and where they can be found, in addition to defining controls and quality mechanisms. All of these create a framework where every user of these data can determine their suitability for any particular situation. Data resource management includes many separate activities, among them:

- Enterprise data modeling, including definitions and relationships
- Data standards and data dictionaries
- Data administration
- Data quality assurance and quality management
- Metadata management (see below)
- Creation and maintenance of data warehouses and data marts

The emergence of distributed and end user computing has made DRM much more complex than it was in the days of centralized computing. More than ever, it remains a critical activity.

A good understanding of how and when data were acquired, and other pertinent details, is essential in determining the suitability of any data. The description of the characteristics of data is called metadata. This metadata is a prerequisite for the measurement of data quality.

The relationship of metadata to data is comparable to the relationship between a product description and a manufactured product such as a personal computer. Another example of what metadata represents is found by looking at a quality map. Its metadata will include details such as scale, type of projection, date of production, the identity of the publisher, and other details that will tell the person looking at this map whether or not it is appropriate for the intended use.

Information consists of data, possibly from various sources, presented in a context that is new or pertinent to a specific situation. The way in which data are organized will depend on an individual's needs, how the data are interpreted and the purpose for which the information is sought. These data may also exist in the form of *documents*.

In general, a document is a record or the capturing of some event or thing (for example, philosophical thoughts, dissertation) so that the information will not be lost and can be communicated. A document is frequently written, but it can also contain or be made with images and sound. A document usually adheres to some convention based on similar or previous documents or specified requirements.

Examples of documents are sales invoices, wills and deeds, newspaper issues, individual newspaper stories, oral history recordings, executive orders, and product specifications. Document collections can be found in filing cabinets, archives, libraries, etc.

Printed or nonelectronic documents can be converted into electronic form and stored in a computer as one or more files. Often, a single document becomes a single file but an entire document or individual parts may be treated as individual data items. As files or data a document may be part of an organized collection or a database. Electronic document management (EDM) deals with the management of electronically stored documents.

A key objective of data information resource management is that of ensuring that data, documents, and other forms of information can be shared, which in turn creates requirements for compatibility, rationalization, and various other standards. Sharing, in turn, requires rules concerning the release of data, documents, and information (DDI), definition of access rights, and all other aspects of information security. Information science includes this and many other related topics.

Knowledge is often described as the ability to use information so as to be able to do something with it. In philosophy, the theory of knowledge is called *epistemology* and deals with such questions as how much knowledge comes from experience or from innate reasoning ability; whether knowledge needs to be believed in order to be used; and how knowledge changes as new ideas about the same set of facts arise.

In information technology, knowledge is the possession of information and/or the ability to quickly locate it. This is essentially what Samuel Johnson (1709–1784) stated: "Knowledge is of two kinds: we know a subject ourselves, or we know where we can find information upon it." In practice, this knowledge also implies the possession of experienced "know-how."

The main differences between information and knowledge are as follows.

Information is considered a self-contained item. The question "where can this information be found?" is completely legitimate. Furthermore, it can be found, bought, written down, collected, compared, put on a database, etc., by any interested party.

Knowledge is considered a personal attribute. The question "who knows this?" is more appropriate than "where can I find this knowledge?". Knowledge is invariably hard to acquire, hard to share.

The author of a book or an article documents part of her/his knowledge and invariably assumptions need to be made as to what is expected as "common knowledge" to the reader.

Among the factors that influence how knowledge is transferred from one person to another are the degree to which the recipient already has already acquired a suitable conceptual framework and relevant past experiences.

## III. USING DATA AND INFORMATION

The discussion of this topic is divided in four segments:

1. Defining the need, the search, and the usage
2. Information complexity
3. Context
4. Turning information into knowledge

The abbreviation DDI will be used throughout this section.

### A. Defining the Need for Data or Information, Its Search and Usage

One of the many adages to appear in the early days of the information systems and technology industry is that

. . . the information you have is not the information you want
. . . the information you want is not the information you get
. . . the information you get is not the information you need
. . . the information you need is not available
. . . in any case, should you (by chance) acquire the information you need
. . . it changes the nature of the subject and then . . .
. . . the information you have is not the information you need.

Defining a *need* for DDI is an important exercise, and the three questions below can be used to facilitate this process:

1. What DDI do I need to provide to the people with whom I collaborate and to the people on whom I depend. In what form, how often, etc.?
2. What DDI do I need myself? Who/what can deliver it and in what form?
3. What DDI are *critical* to me?

These needs are always influenced by context, such as the nature of the situation being considered, prior knowledge, and purpose.

A *search* for DDI can be carried out in many different ways, ranging from looking through the contents of boxes in a dusty archive to using a search engine such as those available through the World Wide Web. The role of technology is to provide better, faster, more powerful, and convenient access and/or processing. At certain points in the technology cycle disintermediation may occur, but this is just a side effect. The person with a specific need can define and refine the search and assess the results as they are delivered.

The library card catalog was largely a disintermediating technology in many contexts: people with simple needs or with some minimal training could by and large understand the tool and find what they wanted, and because of standardization, transfer skills easily to other contexts (i.e., other libraries). Computer systems both in libraries and commercial databases brought increased need of mediaries, because search languages, field tags or names, and command languages were obscure and varied, as well as because of new cost structures. We may be seeing now increased disintermediation, because of simpler, more standard interfaces, much cheaper computer costs and communication/network technologies that make intermediation more complicated than before. But that is not the primary role of technology.

The success or otherwise of a search will be determined by

- The interest in, and value of, the DDI being looked for
- Knowledge of what is or might be available
- Accessibility of the internal and external sources
- Familiarity with search processes and tools
- Level of quality required of the DDI

Different criteria and techniques will apply depending on how much is known about what is being looked for and how much is known about different ways to search. In addition, these will be influenced by:

- Whether the DDI is in printed or digital form and, if in digital form, it is structured (such as a database) or unstructured (a text or compound document consisting of text, audio, image, etc. files); text documents may be in different languages and/or use different character sets
- The volume of the DDI to be located and retrieved, the time sensitivity of the results, the cost of access, any restrictions about the re-use, copy or further dissemination of the material, etc.
- The skills of the person in expressing the search such as using a query language and in formulating a search strategy, the functions supported by the query language and the database (such as full text or keyword search) and the availability of tools such as control lists, dictionaries, and thesauri.

## B. Use of DDI

As DDI can be put to use in many different ways, the use of data, documents and information is always driven by situational factors. The effectiveness with which they are used is influenced by the prior knowledge, experience, and cognitive abilities of the user, the functionality and quality of the tools available, and the knowledge of how these are used.

## C. Analysis

There are many ways to combine DDI from multiple sources and manipulate them with specific objectives. Such objectives may include summarizing large volumes of data using statistical techniques, pattern recognition, data mining and discovery, modeling complex phenomena, simulation, and virtual reality.

## D. Business Intelligence

Making sound business decisions based on accurate and current information takes more than intuition. Data analysis, reporting, and query tools can help business users synthesize valuable information from a sea of data—today these tools collectively fall into a category called "business intelligence" (BI).

This relies on the use of computer systems and technologies for gathering, storing, and providing access to enterprise and external DDI to support the process of making informed business decisions. The BI applications include the activities of decision support system (DSS), query and reporting, on-line analytical processing (OLAP), and forecasting.

In addition to supporting decision making, BI also covers the integration of multiple sources to deliver situation reports and provide the basis for risk management.

## E. Dissemination

For DDI to be shared, the appropriate publication processes need to be in place. The processing of DDI from multiple sources, to condense it, present it in a more readily usable format (which may include the visualization of data and trends) and other editorial techniques will put DDI to a productive use for an organization and/or its wider external community.

### 1. Information Complexity

The sheer volume of data, documents, and other forms of information available causes their use and management to be a significant challenge. The problem is compounded by the growing need to integrate information from multiple sources, many of which are external to the organization. These components need to be validated, rationalized, and combined with other components. These activities require that the precise meaning of each one of these components be clearly understood and that the relationships between the various components be clearly defined. External information components may present issues of language and intellectual property.

There are large quantities of data, documents, and other forms of information generated *within* an organization, residing in a variety of information systems such as technical and production systems databases, data warehouses, document management systems, enterprise resource management (ERP) systems, electronic (and other) archives, etc.

As information technology enables or even encourages the decentralization of these resources, it becomes harder for any organization to "know what it knows" and be able to track, access, share, and exploit the assets it has.

These internal sources are complemented by an even larger number of external sources of DDI in digital form, and a still larger number of sources in nondigital formats. The digital sources can be classified in groups such as:

- Commercial information providers: these can be general-purpose providers such as real-time stock market and other financial information, newswires, and services specializing in specific industries
- Other commercial sources include industry associations, vendors, and other service providers
- Digital libraries, professional and other journals, on-line newspapers (including archive services)
- Governments, nongovernmental and other not-for-profit organizations
- Personal and special interest group websites and many more

As a result of the wide choice of material, a number of issues face the user of these sources: the ease or otherwise of integrating these DDI, made complex by the lack of standards; the cost of obtaining these DDI and associated copyright; and the growing importance of assessing quality before using any of this material.

## 2. Information in Context

The ability to access information, particularly in digital form, makes it easy to fail to take into account important factors without which it may be hard to understand what the actual DDI might mean and why it is important.

The elements of this context include, at the very least, the following:

- DDI and explicit knowledge already available on a particular subject
- Tacit and common knowledge specific to the environment and subject matter
- Knowledge about the organizations, institutions, communities, and individuals, that create and maintain the DDI at their sources

One of the side effects of implementing information systems, information management, and information technology operations particularly well is that they become "invisible" to the end user. In the case of a digital library, for example, it becomes easy to accept the illusion that there are no library collection managers, librarians, and catalogers, publishers, editors, referees, authors and information technology infrastructure operators, and that things happen automatically.

## 3. Turning Information into Knowledge

The main purpose of DDI is to enable the sharing of information and thus support the creation and development of knowledge. This, in turn, drives innovation and accelerates change as a result. Innovation and change create new DDI and knowledge in a self-reinforcing cycle. New knowledge may modify or displace previous knowledge.

Knowledge is not lost when it is shared. In fact, the opposite is true, as knowledge increases as a result of interactions. However, the sharing of knowledge cannot be taken for granted due to a number of reasons.

*Cultural reasons* such as behavior are learned early in the educational cycle. For example, sharing knowledge during examinations is called cheating. Moreover, Francis Bacon (1561–1626) stated that "knowledge itself is power"—and human history shows that power is not something people give up willingly.

There are *practical reasons* relating to the fact that there are three distinct kinds of knowledge:

1. Knowledge *about* something: The easiest to acquire as it only requires that the recipient becomes aware of a particular subject
2. Knowledge *on how to do* something: More difficult as it requires considerable learning, and the acquisition of both explicit and tacit knowledge, the latter being best achieved by working with a mentor or in a community of interest; in addition, learning how to do something also requires practice
3. Knowledge of the *why* of something: Understanding the underlying principles and/or theories differs from the above insofar as it requires the learning and comprehension of conceptual as well as practical matters.
4. *Knowledge management* is a relatively new concept in which an organization systematically gathers, organizes, shares, and analyzes its knowledge to support its objective of getting the knowledge from "those who have it" to "those who need it", in turn supporting the organization's business objectives as well as creating new knowledge

It is of interest to note that the value of data, information and knowledge has so far defied quantification in standard accounting terms. Nevertheless, knowledge can represent serious money as evidenced in, for example, risk management in the financial sector, or in reduced capital tied up in inventories as a result of implementing "just-in-time" inventory management systems.

## IV. DDI QUALITY ISSUES

This article has referred to the very high volumes of DDI available today, and to the need for quality.

Data quality exists when it enables its users to accomplish their objectives. In the field of total quality management, "quality" is defined as "the ability to consistently meet the customers' expectations" where "expectations" means "conformance to requirements." The importance of quality can be illustrated through an analogy: poor quality of data and information is the Information Age equivalent of manufacturing scrap and defective units of the Industrial Age. Poor quality DDI is a waste of resources and really bad quality can render an organization dysfunctional or become the subject of litigation. Ensuring the quality of DDI is a complex matter and requires considerable resources.

The components of DDI quality include, at the very least:

- *Accuracy:* Best defined as the opposite of an error. An error is a discrepancy between the recorded value for an event or object and the actual value as measured or observed. Accuracy is always relative to the specification of how a measurement or observation is to be carried out and recorded.
- *Precision:* A term that describes the amount of detail that can be discerned from a measurement or observation. Precision is always finite because no measurement system is infinitely precise. Furthermore, the recording of data/information is frequently designed to reduce detail. In the case of a document, precision related to the degree of relevance to a particular topic.
- *Completeness:* Refers to a lack of errors of omission in a set of DDI or in a database. It is assessed relative to their specification, which defines the desired degree of generalization and abstraction (selective omission). This may also be affected by policies on archival/long-term preservation.
- *Timeliness and currency:* These are defined by the time it takes for updates to become visible to users' environments. In operational environments, real-time updates are usually available immediately. Therefore, data timeliness is instantaneous. When digital data is extracted from a data warehouse, the frequency of updates becomes the defining factor.
- *Consistency:* Refers to the absence of apparent contradictions in a set of DDI. It is also a measure of the internal validity of a set of DDI or of a database. It is assessed using information that is contained within the set or the database.
- *Traceability:* Defines the extent to which the history of changes to the DDI can be followed back in time.
- *Security:* Identifies what DDI a given individual or user group may access.

When DDI do not meet the users' expectations, the predictable result is that these DDI will not be used. Instead, empowered users with knowledge of end user computing will create their private stores of DDI in electronic form or develop applications that create multiple copies of the "same" data in an uncontrolled environment. Others will build their own collections of paper documents and other forms of private archives. To an organization, these represent unnecessary costs in building and maintaining nonvalue-added items.

Finding examples of poor DDI quality is relatively easy. Poor quality DDI may also exist in corporate databases and other sources. Typically, there are islands of information in many corporate environments as a consequence of what has been described in the previous paragraph. In addition there is legacy data that can no longer be accessed because the application that created it has been withdrawn, there are documents written with word processing software that is no longer supported, etc. Similarly there may well be printed documents that are not included in corporate records.

In the World Wide Web there are numerous sites with out-of-date information, untrustworthy content, broken or incorrect links, etc. These have been referred to as "garbage on the information superhighway."

There are many potential sources of poor quality. These include:

- Lack of accountability for the creation, updating, and maintenance of DDI
- Lack of standards and definitions, which cause DDI to be mismatched with the intended purpose for their use
- Mistakes in data entry or conversion
- Lack of validity and consistency checks
- Missing attributes (a customer record showing "J. Smith" should indicate if it is Mr., Mrs., Ms., Dr., Prof., etc.)
- Homonyms that are incorrectly interpreted (St. could be Saint or Street)
- Lack of data quality audits

## V. GOOD PRACTICES

The following are offered as a complement to the previous sections and are intended to highlight the differences between computer systems and human beings. Understanding their strengths and weaknesses is an important component of how DDI will be used.

### A. Things That Cannot Be Expected from Computers

Computers as they are known around the year 2000 are able to process algorithms and logical sequences. They have, at best, crude sensory and artificial intelligence capabilities. As such they cannot understand context or situations.

Therefore, DDI are not necessarily "right" because they are delivered by a computer, and the user of these must be fully aware of the quality issues discussed in this article.

It is critically important that any DDI be relevant to the specific purpose for which they are collected. DDI collected for one purpose may be totally inappropriate for a different purpose.

### B. People Do Not Think and Act as Computers Do

Unlike computers, people are emotional beings who are extremely good at recognizing context, but whose knowledge is limited. There is no simple mechanism for people to know what they don't know. These few factors have important consequences:

- People cannot give good answers to the question of what information they need
- A person may have access to large amounts of DDI and/or knowledge; this does not guarantee that these will be used objectively in making decisions or taking actions
- A person who has a particular item of DDI needed by another person will not necessarily give it freely
- Whatever is encoded into a computer system by one or more people is not necessarily complete
- Implementing new computer systems does not guarantee that DDI will flow freely across an organization, or that this information will be used effectively

## VI. DILEMMAS FOR THE INFORMATION AGE

There are frequently voiced complaints that there is too much data and not enough information. Others claim to suffer from an information overload and maintain that the technologies of mobile computing, integrated messaging, etc., are demanding an increasing amount of their time. Some wish that it was possible for their company to "know what they know."

At the same time, there is often a feeling remaining that much of the information needed either within an organization or from external sources remains difficult to find and extract.

All of these are related to a number of human factors.

### A. The Need for a New Literacy

Teaching the basic skills of reading, writing, and arithmetic was a by-product of the needs of the Industrial Revolution and the businesses that emerged from it. Of course, these skills continue to be needed today.

Additional skills are needed to be able to operate effectively in the Information Age. In particular, the ability to formulate information needs, the ability to navigate through large amounts of data and information, and the ability to analyze, recognize, interpret results and extract meaning from all of these becomes increasingly important.

Failure to acquire these skills will reinforce the incongruity of an "information illiterate" user with a supercomputer on his desk, if not in his hand, connected to every other (super) computer in the world.

### B. Cultural Resistance

The best processes, information flows, formal organizational charts, etc., are not sufficient to ensure that the "unencodable," factors such as tacit knowledge, peoples' networks, and organizational politics can be or will be formalized in information systems.

### C. Wider Sharing of Information and Knowledge

While this is highly desirable in a world that is truly networked through ubiquitous technologies, the ownership of information and knowledge, formalized through patents and copyrights, constitutes a major competitive factor which may inhibit or prevent its sharing.

Data, information, and knowledge can be put to creative use to improve business performance, enhance quality of life, and develop a better understanding of the universe.

They can also be used to create misinformation, disinformation, and fraudulent, potentially dangerous and destructive products or processes. Depending on each individual's values and framework, some of these will be good business opportunities and others should be avoided at all cost.

Finally, while it is true that in the last few years mankind has collected large volumes of data, information, and knowledge about an unbelievably large range of subjects, extending from the nature of the universe to the mapping of the human genome, this does not answer many questions that continue to be studied. For example:

- Why do people dream?
- Why are some people more susceptible than others to particular diseases?
- Is there life beyond earth?
- Is poverty inevitable?
- Are we ever going to catalog all the knowledge that exists?

## SEE ALSO THE FOLLOWING ARTICLES

Computer History • Data Mining • Decision-Making Approaches • Decision Support Systems • Digital Divide • Informatics versus Information Systems • Information Measurement • Knowledge Acquisition • Knowledge Management • Virtual Reality

## BIBLIOGRAPHY

Borges, J. L. (1944). *The Library of Babel.*

Brown, J. S., and Duguid, P. (2000). *The social life of information.* Boston, MA: Harvard Business Books.

Haeckel, S., and Nolan, R. (1993). Managing by wire. Boston, MA. *Harvard Business Review.*

# Data Mining

**Marietta J. Tretter**
*Texas A&M University*

## GLOSSARY

**classification** A group of analytic methods for putting observations or customers into separate and relatively distinct groups using all available information. These groups could be something like good credit risk and bad credit risk.

**clustering** A group of analytic methods that put observations or customers into groups that may be related or overlapping.

**data mining** The process of extracting knowledge from very large data sets.

**decision trees** A group of analytic methods that succesively separate or split observations or customers into branches of related groups forming a tree of groups. To get the rule for the separation one starts at the end of the branch and follows the path back to the original data set. Trees result in generally easy to implement and understand models.

**discriminant analysis** A multivariate statistical technique that builds a regression like equation that allows distinction between two or more sets of data. To build the discriminant analysis model or equation on good and bad credit risks you would need to already have the two groups distinguished before running the discriminant analysis. You could then apply the model to unknown credit risks. Neural networks can also be used to perform this model building.

**flat file** Most statistical packages prefer to work on data in flat file format. This means that each observation with its related variables is stored as a single row of entries.

**input variables** The information or facts used to model or explain the target variable. If the target is a good loan risk then the inputs might be credit rating, income, own home, etc.

**linear versus nonlinear models** Linear means a straight line can best represent the relationship between two variables and nonlinear means that a straight line gives a bad fit of the relationship. Generally the utility of the various data mining techniques can be judged based on their ability to fit linear or nonlinear relationships. Most real world applications tend to be nonlinear.

**logistic regression** A traditional statistical method that is often quite powerful in building data mining models. It fits probability of occurrence rather than actual occurrence. In the good credit risk model you would be fitting the probability of a good credit risk.

**model** A relationship, usually expressed by a mathematical formula or a set of rules that relates the target variable to the other input variables of a customer or observation unit.

**neural networks** A group of analytic methods that compares all observations across all variables to get the strongest relationships between variables and observations. They produce robust models that are often difficult to interpret.

**observation** All possibly relevant information about a unit of interest. For example, if we are modeling customer behavior, the customer observation would include such variables as income, age, occupation, hobbies, credit history, address, own home, education level, etc.

**overfitting** Suppose you have two variables that are known to have a simple linear relationship. If those are real variables and you plot them you will get an approximate straight line, but there will be some variability due to measurement error or other reasons and not all the points will lie exactly on the line. A simple regression model will fit a straight line that is as close as possible to all the points but still a straight line.

A neural network, for example, can fit the exact, not quite straight line. This is called overfitting and often results in a model that accurately fits the data at hand but does not do very well on a new data set since it fit many of the inconsistencies in the original data set.

**regression analysis** A mathematical and statistical method for fitting a straight line model of a target and input variable.

**target variable** The single fact that is trying to be modeled in the data mining process. This could be something like good loan risk, large purchaser, etc.

This article is an introduction to **DATA MINING.** It defines data mining and differentiates it from traditional applications of statistical methods.

Several examples of uses of data mining are given in the context of current practices in business and industry. The role of model building in data mining is explained and several of the techniques used in data mining are briefly covered including some available software which is a very dynamic topic.

## I. WHAT IS DATA MINING?

The concept of data mining (DM) has existed for several years, although it has been clearly identifiable only within the last five years. When seeking information about DM it is easy to get confused because the area originates from three separate disciplines: information technology (IT), machine learning (ML), and statistics. Information technology contributed to the development, storage, and manipulation of large amounts of data. These data accumulating in quantity naturally led to the question of what sort of usable information might be extracted. Machine learning and artificial intelligence spent many years building algorithms that recognize patterns in some sort of data (numeric, verbal, symbolic). Statistics was founded by the need to summarize and analyze mostly numerical data. DM is a rough mixture of the techniques and terminology of the three areas and thus causes confusion because there are often three different ways to refer to the same thing. For example, variables are also referred to as attributes or inputs and observations are cases or collective inputs with targets.

What is DM? Actually the name is very descriptive. Quite simply, DM takes a relative mountain of information (data) and attempts to extract a few gems or nuggets of knowledge. In fact many of the software packages, societies, user groups, etc. involved with DM carry that idea in their titles: Gold Miner, KDNuggets, Diamond SUG.

## II. WHAT CAN DATA MINING ACCOMPLISH?

From just a few of its many success stories, DM can: help you sell more products; identify new chemical compounds for prescription drugs; identify fraudulent insurance claims; make your customers happier; help you to retain customers; send catalogs only to customers that would be interested; target merchandise only to customers who would be interested in buying it; identify diseases; map genes; improve the reliability of big yellow tractors; help you get your car fixed quicker; and help you browse a web site for only what you want to buy. The list of DM applications grows daily as do the number of DM software tools available to generate these applications. Many of the successful applications of DM have been in the context of marketing. It is so prevalent in this context that it has become a necessity rather than an option for large-scale marketing. Those applications, which are not marketing driven, are often still related to a profit motive in a business context. Where there are large data sets and large amounts of money to be lost or gained you will find DM applications. Ultimately a good part of what DM accomplishes is to make or save money even if the original intent, for example, is to identify diseases.

## III. WHAT DATA IS NEEDED TO DO DATA MINING?

To a statistician a lot of data would be 10,000 cases with perhaps 20 variables or pieces of information per case. To a data miner a billion cases with perhaps 3000 variables would be a reasonably large amount of data. The new frontiers of DM are working with terabytes of data. IT contributions to DM established data warehouses, which are now a main supplier of information to the data miner. IT also provided on-line analytic processing (OLAP) tools used in decision making based on data. However, OLAP remains a part of data warehousing and will not be considered further here.

Since many of the algorithms that build DM models have origins in statistics, the typical DM software expects data in a flat file consistent with how most statistical packages handle data. A flat file stores all the variables (information) associated with a case or observation (customer) in one row. For example, a human customer, John Smith, is a case and his variables might be name, address, age, income credit history, etc. All of the information for this one customer would comprise a row of data for DM. In a data warehouse this customer's information would be stored in several places in a relational format with links of purchases, returns, etc., back to the main customer record. Thus the data miner rarely builds models directly from the data warehouse or a subset data mart.

Up to 80% of the effort that goes into DM is involved in extracting, formatting, and cleaning the data before it is ready to be analyzed. Poorly prepared data will give poor models and poor predictions. Data are rarely perfect and often information stored in warehouses is incorrect or incomplete. A data set with 30 variables and with, say, 1% of the data missing for each variable could potentially result in 30% of the cases being unusable. Data may need many transformations before it is usable. Dates may need transformation to Julian dates; ratios may need to be developed; and qualitative or alphabetic data may need to be converted to quantitative or numeric data. In systems collecting terabytes of data, it may be reasonable to build DM primitives into the data warehousing process to create some of the transformations required in the DM model-building process. For example, the telecommunications industry stores terabytes of call and customer information. After much experience with this data they now typically generate often used data transformations from DM primitives built into the data warehouse. These speed up the model-building process because the DM analyst does not have to spend time repetitively producing necessary transformations. In order to prepare data for DM it is important to have domain knowledge as well as knowledge of the DM processes. Variables will have to be selected and those variables will possibly need transformation and cleaning. DM requires a large amount of relevant and clean data in, typically, a flat file format.

## IV. EXAMPLES OF DATA MINING

There is no shortage of DM examples both good and bad. A search of the Internet yields more than you can read. This includes one of the great myths of DM, beer, and diapers. The myth comes from a Market Basket analysis, which supposedly found that when people buy diapers they also buy beer. This of course can be rationalized in many different and believable ways. However, no one has ever been able to substantiate this claim. Another claim is a relationship between purchase of Barbie dolls and chocolate. This claim has been substantiated and many explanations have been proposed. Russell Stover Candy markets their chocolates in boxes with pictures of Barbie on them.

In the applications below, the following information is given: cases, inputs, target, and action. This terminology was chosen to keep the applications consistent and comparable and to fit the general DM model-building process explained in the next section.

### A. Retailing

Campaign management has produced very positive results for DM. The goal is to build a model that predicts which customers will most likely respond positively to a mail advertising campaign. The cases or observations consist of customers, prospects, or households. The variable that is to be modeled (referred to as the target) is response to a past or test solicitation. The idea is to build a model that explains why customers responded positively to a past campaign and then use that model to predict future campaign responses. The input data can be geographic, demographic, or psychographic information for each customer or case. These data may come from customer purchase records, questionnaires, and commercial data providers that sell demo-, geo- and psychographic data. Fingerhut, for example, is a long-established mail-order retailer that has used DM very effectively to increase sales. When you order from a mail order catalog you will quickly find that you will start receiving many more catalogs from the same or different vendor. The companies that produce these catalogs generally use DM. Ideally if their DM were perfect, you would only receive the catalogs you are interested in.

Customer relationship management (CRM) is the evolving and more sophisticated application of DM to retailing. CRM is an attempt to create and maintain the type of customer relationships between retailer and customer that existed in the days of the Mom and Pop general stores. These stores knew their customers well. For example the butcher, Pop, knew that Mrs. Jones would come in and buy a roast every Wednesday. On Wednesday he would make sure he had the roast she likes and might suggest items to go with the roast. With people shopping on the Internet and/or catalogs it is once again possible with DM tools to know customers like the Mom and Pop stores did.

CRM is a collection of DM and data warehousing tools that let you collect and analyze customer data. The goal of CRM with respect to the customer is to make sure you have what they like, offer them new or related items that they will like, and generally keep them as a loyal customer. The customer must be made to feel that they are valued because you understand and anticipate their needs and desires.

### 1. Credit Scoring

Before DM, the equivalent of credit scoring was often done using a multivariate statistical technique known as discriminant analysis. When neural networks started to become popular many people replicated the results of discriminant analysis with various neural networks. DM can actually use these methods as well as an arsenal of other tools to produce improved results. In credit scoring, a bank, for example, is interested in identifying good credit risks based on a model built using cases from past loan recipients. The cases would consist of both good and bad credit risk customers. The variables or inputs on each existing credit customer would be application information including income, age, etc., and credit bureau reports. The model target or the variables modeled from the inputs could be defaulted, charge-off, serious delinquency, repossession, or foreclosure. If the model proves to be a good model then the bank can use it in screening future loan applicants. This model could also relate to credit card offerings. This model gives a credit score or number, which can be interpreted in terms of credit risk.

### 2. Customer Retention or Churn

Most businesses are concerned with keeping current customers. A large insurance company found itself in a situation where they were not getting new customers and they were losing existing customers. Obviously not a good circumstance. By using DM they modeled their existing customers and dramatically turned the situation around within two years. The rugged competition among phone companies has also made them very concerned with churn among existing customers. To handle churn, some companies have terabyte data warehouses that contain customer calling history. The data warehouses have imbedded DM primitives that process this information in real time. When a customer calls the phone company, a representative brings the customer history up on a computer screen along with a list of incentives to offer this customer. The phone companies also use this information to call customers and offer various incentive packages.

The observations or cases for churn prediction are existing and attrited customers. The input variables are payment history, product and service usage, and demographics. The potential target variables reflecting attrition are churn, brand switching, cancellation, and defection. The model, if useful, will predict candidates for customer loyalty promotions or campaigns. The promotion may include such things as better rates for loyal customers, exclusive offerings, improved service levels, etc.

### 3. Fraud Detection

Credit card companies, Internet merchants, medical insurance, and telecommunications are a few of the industries concerned with fraud. DM has proven to be very useful in detecting large-scale fraud. The cases or observations are past transactions or claims. The inputs are particulars and circumstances. One Internet merchant mined sales transactions to try to determine characteristics or patterns of fraudulent purchases on credit cards. What they found was that all of the fraudulent purchases were centered on just a few zip codes. In this case exploratory DM was more useful than predictive DM models in finding patterns. The target variables for a model would be fraud, abuse, or deception. The action taken as a result of a successful model is to impede or investigate suspicious cases.

The telecommunications industry runs DM primitives on terabyte data warehouses keeping track of all phone calls. In this way suspect patterns of phone calls are caught in real time. The DM primitives are developed and modified by preliminary model building and consistent updates on the model and primitives built into the data warehouse. As new fraud patterns are detected, the warehouse primitives are modified.

## B. Manufacturing

Manufacturers of heavy mechanical equipment must be concerned with quality control. For example, the engines in earth-moving equipment will be expected to function for many years and hours. If they do break down they will be repaired. Thus the manufacturer must insure the reliability of the engine components. When an engine breaks down, replacement parts must be supplied on a timely basis. One such manufacturer has data on every heavy equipment engine made and its repair history. By using DM on this data they can better design engines to eliminate weak parts. They can also predict which parts need to be stocked for repairs. The cases or observations are engines or components of interest. The inputs or explanatory variables are the various components of the engine and their replacement history (all have serial numbers for tracking purposes) plus any usage information available (this might include age, hours running, and maintenance history). The target variable is engine breakdown, time to breakdown, etc. A combination of predictive and exploratory modeling can be used in this case. Breakdowns could be predicted or weak components identified. The action is to insure the reliability of the engine.

### C. Other Applications

One of the relatively new applications of DM is the integration of CRM with supply chain management (SCM) and/or enterprise resource planning (ERP). Much of this effort is webcentric. There is little documentation in this area simply because the development has been so fast that little time has been available to write specific details. It is a pioneering area, which will see much development within the next two years. There are many other applications of DM that are not touched on here. One entirely different area is text mining, which remains mostly in the domain of ML.

## V. GENERAL PRINCIPLES OF DATA MINING MODEL BUILDING

It is not practical to do DM without a computer or appropriate software. Most of the DM software suites approach DM from a model-building perspective analogous to the process of building universally familiar multiple regression models or econometric models. The steps in building a DM model are

1. Identify the problem and what is to be modeled
2. Get the appropriate data
3. Clean the data and perform any transformations that might seem appropriate
4. Divide the data into training and testing sets
5. Build a model on the training data
6. Evaluate the model
7. Use the model
8. Monitor the model performance

Building a DM model is an iterative process. If the result of step 6 is not satisfactory then the process may go back all the way to step 1. You may need to rethink what the problem is or what you want to model. You may need to select different data and variables or you may simply need to eliminate some variables or add some transformations. You may also need to try different algorithms in building the final model.

There are two general types of models that can be built depending on the goals of building the model. Predictive models are built to predict or forecast the behavior of the modeled variable or target based on the inputs to the model. The idea is that if you know the inputs, for example, income, debt, age, then you can predict if a person is a good credit risk (target). Another type of model is descriptive. It does not try to predict what will happen but explains or describes what is happening in your data set. In the manufacturing application a predictive model would attempt to predict when the engine will fail. A descriptive model would describe the failure of the engine as it relates to the failure of various parts.

### A. Identify the Problem and What Is to Be Modeled

When you build a model you need a goal or something to aim for, a target. To focus the discussion, consider the possible targets for a strictly mail order company selling through catalogs. They might want to increase profit, sales, or decrease mailing costs. They might wish to obtain new customers, or increase the number of items ordered by current customers. They clearly want to get as much profit as possible out of each catalog mailing. The number of potential targets may be limited only by the imagination or understanding of the business. Suppose the company decides to look at increasing revenue for each catalog mailed. This is, in fact, not a simple problem. It is related to how many people respond to a catalog mailing and how much they order so it would probably involve at least three models. If this is the first attempt at DM it would be good to focus on just one model to begin with. Thus a reasonable target would be to model the best customers who can be identified as those that spend the most from the catalog.

### B. Get the Appropriate Data

The target to be modeled is the highest spending catalog customer. Since this model is only for a particular mail order company, the primary customer data must come from this company. If the company has not collected this data in some sort of data warehouse little can be done in building the model until that is accomplished. If the data is available then the next step is to decide what customer data is needed. They might want to start with all customer data and do some descriptive data analysis including means and histograms to decide how they define the best customers—spending cut off. They may also wish to purchase data from a credit bureau and from data vendors that might supply additional information on their customers spending habits from other catalogs. Ideal in-house customer data would be type of purchase, amount, frequency, or time lapse between purchases, record of returned merchandise, etc.

### C. Clean the Data and Perform Any Transformations That Might Seem Appropriate

It might seem surprising that data would need cleaning if it is all in-house data. Many things can happen to data in the best of circumstances. For example, the same customer may be listed under several different variations of their name. They may often return merchandise and these data may be stored separately or not linked to the customer's regular file. They can be missing for many reasons. Where there is manual data entry there can be errors. It is always useful to run descriptive measures of the data to spot missing or extreme errors.

Once the data is merged and cleaned it will then be necessary to look for initial transformations. For example, the frequency or time lapse for orders may be stored as dates and it will be necessary to convert those to frequency counts. Depending on the techniques used to build the model the data may need mathematical transformations. For example, it may turn out that a ratio of income-to-dollar purchases gives better results than income and dollar purchases alone. If merchandise has been returned then the amount purchased needs to be adjusted. Missing data is always a problem and care must be taken to address that problem consistent with the algorithms you might be using. Simply ignoring missing data is rarely a good solution.

Once these data are cleaned and merged by customer you can then decide which inputs or data will be relevant to modeling the target. Domain knowledge is very relevant for this process. The more the analyst knows about the catalog business, in this case, the more likely the model will be good at predicting customer purchasing behavior. Some DM software will automatically eliminate variables and even do transformations, but a person with domain knowledge and some skill at model building can almost always build a better model than this software alone.

### D. Divide the Data into Training and Testing Sets

The cost of building and using bad DM models is high. Thus it is wise to test the model in the building process by holding back some of the data from the model-building process. The original data set is randomly split into at least two parts (in the later stages of the model-building process a third split is used for validation of the final model). The larger part, 95 to about 65% of these data are used to train or build the model and the remainder are used to test the model to see if it holds up under a different set of related data. The reason for doing this is that if you work hard enough on the training data you will in most cases be able to optimize or overfit the model to the data. Overfitting means you did not capture what is really going on in the population of interest but you have fit all the consistencies and inconsistencies in your training data set. Neural network algorithms are particularly subject to overfitting.

For smaller data sets (3 or 4 thousand rows) there are more elaborate ways of validating and cross validating the model. For example, these data could be split into two equal parts. You would train on one set and test on the other. Then, train on the previous test set and test on the training set. The error rates of the two model-building sets can then be averaged. The idea, again, is to determine how the model holds up on a set of data different from the data set that was used to train or build the model.

### E. Building a Model on the Training Data

Building a model on the training data involves fitting the model you have chosen with a particular DM algorithm. In DM it is common practice to fit using several different algorithms or tools. The models from each algorithm are compared. It is very likely that you will iterate back to the variable selection and transformation phase several times before you are satisfied with a final model for the testing phase. DM software suites typically support the process of fitting several different algorithms, comparing results, and refining the model by iterating through the model-building process. The reason for this is that even if the data analyst is very familiar with the model-building process and has extensive domain knowledge, the data sets are usually complex enough that an experienced analyst may not be able to completely judge the underlying relationships in a model. Fitting several algorithms helps to uncover underlying relationships in the data. Some of the algorithms are better at fitting linear relationships and some are better at fitting nonlinear relationships. Most real data is complex and not simply fit by linear models. Neural networks are typically very good at fitting complex relationships and because of that are usually run as benchmarks for other algorithms. They are not always the tool of choice for the final model because they tend to be harder to deploy, harder to explain, and are also much more likely to overfit the data. Some suites also offer the choice of combining models and this is often an ideal way to completely capture all of the useful relationships in a data set.

## F. Evaluate the Model

Several of the DM algorithms generate statistical estimates and tests. However, with very large data sets some of these classical statistical tests are not too meaningful. The classic $p$ value, for example, will be astronomically small for variables that are hardly contributing to a model. Thus other tools are often used to quickly evaluate model utility. Two very useful tools are the confusion matrix and the lift chart.

### 1. Confusion Matrix

After training we can use the model to classify the catalog customers into spending categories. Suppose we had two categories in the original data set: high spenders and other. We could simply report an error rate of misclassification of existing customers, but the confusion matrix (in spite of its name) helps clarify that number (Fig. 1).

| *Prediction* | *Actual*<br>High spender | Other |
|---|---|---|
| **High spender** | 18 | 3 |
| **Other** | 2 | 77 |

**Figure 1** Confusion matrix.

After fitting the training data or test data to the model we could have simply reported the percentage of customers correctly classified into high spender or other (remember that these are our customers so we actually know which category they are in). If you look at the confusion matrix the diagonal entries of 18 and 77 are the correct classifications and this would give us a 95% classification accuracy since 95 out of 100 cases were correctly classified (the numbers were kept small for purpose of illustration). However, the confusion matrix also lets you answer the question of which case would most likely be incorrectly classified. Would we rather misclassify a *high spender* as *other* or would we rather misclassify an *other* as a *high spender?* I think most would agree that we want to misclassify or possibly lose as few high spenders as possible. We can see from the confusion matrix that this model tends to err in favor of the high spenders.

### 2. The Lift Chart

In the case of our catalog model, we would be interested in how our model might improve catalog sales. A very useful graph for this is the lift chart. Suppose we build a model of catalog customers that have responded to a previous catalog mailing. Note that the population is that of all customers ordering from that catalog. Suppose the response rate for this nontargeted catalog mailing to customers was 10%. We want to know if using the respondent model we can send out fewer catalogs to selected individuals and at the same time improve our sales. This would increase revenue from sales and also reduce mailing costs because we would be sending to fewer customers. The lift chart can help answer this question.

Once the model is fit we then apply the model to the test data and identify the respondents according to the model.

In the lift chart in Fig. 2 the dark bars refer to randomly mailing the catalog to a percentage of the responding customers. The corresponding light bars represent mailing that percentage to customers on a list that is rank ordered (scored by the model) by the likelihood of buying. The rank order is highest likelihood of buying to lowest. If 10% of the scored group were to receive a mailing we would take the top 10%. To read the lift chart we find that if we randomly mail to 10% of our responding customers we can expect a 10% response. However, if we mail to the top 10% of our scored customers, we can expect a 30% response rate, according to the model. We would have to mail to 30% of the random customer base to expect the same response rate. Thus we will get more for less. You can run a cost-benefit analysis to decide the

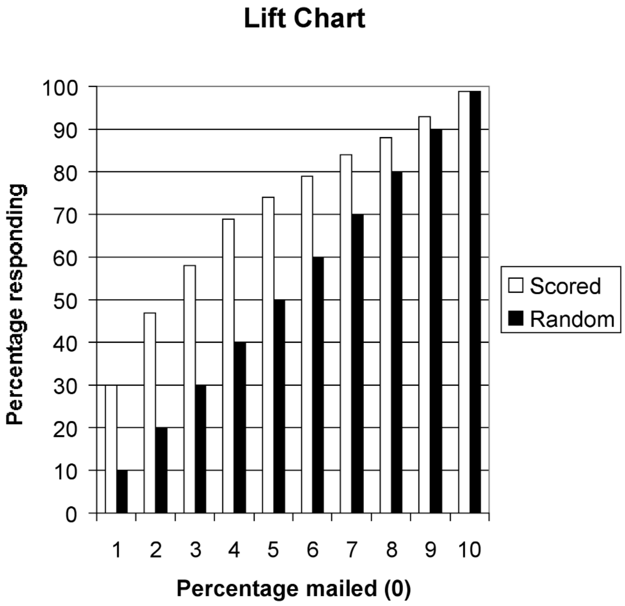


**Figure 2** Lift chart of scored versus random mailing.

trade-offs of mailing to different percentages of the scored data.

A caution flag must be raised here. Lift charts and confusion matrices are very informative. However, it must be remembered that you are fitting known results. When the models are applied to a new situation the results may be better or worse. It is important to monitor the performance of all deployed DM models. If the lift chart claims a 30% response rate and you only get a 5% rate on the next mailing or successive mailings, someone is sure to notice and hopefully that is the person in charge of the DM. The ultimate evaluation of a DM model should be that the model is understandable and makes good business sense. If it seems magic or mysterious, chances are it is not a good model no matter how good the confusion matrices or lift charts look.

## G. Using the Model

If we built a reasonable predictive model of high spenders for our catalogs then ideally we could use that information to identify high spenders who are not yet our customers. One must be careful with that assumption because we would be using the model outside of the group we built it on. We could also use the model to understand what comprises a high spender. Maybe it is income, region of the country, age, interests, or most likely a complex combination of things. Since we used frequency of buying we might also find that these customers buy in some pattern and thus try to match that pattern to catalog mailings.

DM software suites often make the catalog or ad mailing campaign as efficient as possible. Imagine a mail order company sending out as many as 100 different catalog mailings targeting different customer bases in one day. Also imagine that there are thousands of customers. The logistics of this seem overwhelming. However, if the customer data are scored this can be a totally automated process. The model will generate a score something like a test grade for each customer. This scored list of customers is then ranked from highest to lowest scores (higher is better) and catalogs are mailed as far or as deep down this list as desired. Attaching value to a customer response and cost to mailing will result in the lift chart showing return on investment. This can then be used to decide on the depth of mailing in a scored customer base.

The ultimate use of a model depends on the application. In fraud detection the model might contribute to understanding fraud patterns. The model may be a small part of a manufacturing process. The model may be applied to new data for further evaluation. The model may be used to recommend business strategies or actions. The use of a model is as varied as the number of applications of DM.

## H. Monitor the Model Performance

If you attend many presentations demonstrating the usefulness of DM you often have to wonder how the lift charts compare to the actual model performance. Not surprisingly, it is very hard to pin down that information. Whether or not a business brags about its failures, it is a certainty that if the lift chart is much better than the actual performance someone will most likely start to notice a loss in profits, increase in costs, etc. It is not in the best interest of the company to not monitor the actual performance of a DM model. Telecommunications companies go through a model-building process before setting up their terabyte data warehouses. They then use DM models to improve and maintain customer relations. Because of the sheer volume of data they try to use the initial model-building phase to build DM primitive functions into the data warehouse. For example, the data miner may always use a certain ratio of variables when building a model; it is easiest to have the warehouse generate that ratio rather than creating it every time in the model building process. However, since customers change, the telecommunications analysts monitor the situation and if the ratio were no longer useful they would alter the primitives.

There is a current trend for database providers to advertise automatic DM tools built into data warehouses. These tools are really equivalent to built in primitives and cannot take the place of an experienced DM analyst with modeling and domain knowledge. Generally these tools are intended to aid DM analysts and not replace them. DM requires a rich collection of tools and algorithms used by a skilled analyst to produce acceptable results. The DM process is also an ongoing process due to constant changes taking place in the data.

## VI. THE TOOLS OF DATA MINING

DM is a merger of three areas: IT; statistics; and ML. The tools or algorithms of DM make this fact most apparent. About five years ago few commercial suites of algorithms existed. With diligence you could collect together various pieces of software and do some DM. Machine learning and statistical software vendors began marketing various components for DM. Very quickly these components found their way into DM software suites which had a user-friendly interactive model-building environment. This area is far from settled and you will see software suites being bought out by larger software vendors on an overnight basis. Clementine, now marketed by SPSS, is a good example of this. Most of these suites contain algorithms from statistics and from artificial intelligence.

We will consider some of the more typical tools below. New techniques are being developed and need to be developed to work with terabytes of data, which is becoming more common partly due to the large amount of click stream data generated by the Internet. OLAP tools are not considered here because they are typically part of data warehousing.

### A. Logistic Regression

Regression tools have historically been used in business to fit various models. They are appealing because they are reasonably intuitive and the resulting equations, which model a target as a function of input variables, are understandable. In fact a person skilled in building multiple regression models will find the logic of DM model building to be very similar. Logistic regression is part of this regression tool kit that has found respectable use in DM.

Logistic regression is suitable for fitting models to a binary target with a buy, don't buy type of value which can be coded as 1, and 0, respectively. Logistic regression can be used to classify, for example, whether or not a person will buy. It can also be used to predict the continuous variable, the probability that a person will buy a product. The model is built on the logarithm of the odds ratio that an event will occur. If a person has an 80% chance of buying a product the odds of buying are 4 to 1.

One of the underlying reasons some tools are more powerful or useful than others is their ability to handle nonlinear relationships. If a target variable increases, as an example, at the same rate as an input variable, it has a linear or straight-line relationship. However, if the target increases, as an example, at the rate of the input raised to the power two then the relationship would not be fit by a straight line and would thus be nonlinear. Keeping in mind that most models have many input variables, it is not too surprising to find that most relationships are nonlinear.

Logistic regression is a very good modeling tool but it does assume that the target (dependent variable in regression terms) is linear in the coefficients of the input or predictor variables. That, combined with the fact that a successful logistic regression model requires the analyst to provide the variable transformations necessary for this linear relationship implies that the model is only as good as the skill of the analyst that built it.

### B. Neural Networks

Neural networks have their origins in artificial intelligence. They started out as an attempt to model the workings of a brain. Since we are not yet completely certain how the brain works, we can only assume that neural networks are an artificial representation of a biological brain. There are many different variations of neural nets that have been specialized to certain types of problems.

Looking at Fig. 3, a simplistic description of a neural network can be given. Figure 3 represents a neural network using data on customers with the information given on the input node labels. The targets are labeled on the output nodes. In this particular case we are trying to classify customers that have purchased from the target catalogs. One case or observation would consist of a customer's age, income, leisure activities, and from which of the listed catalogs they have purchased. Each case or set of data from one customer is fed through the neural network. As each input node goes to a hidden layer node, a weight is applied (multiplied), the weighted inputs to the hidden node are

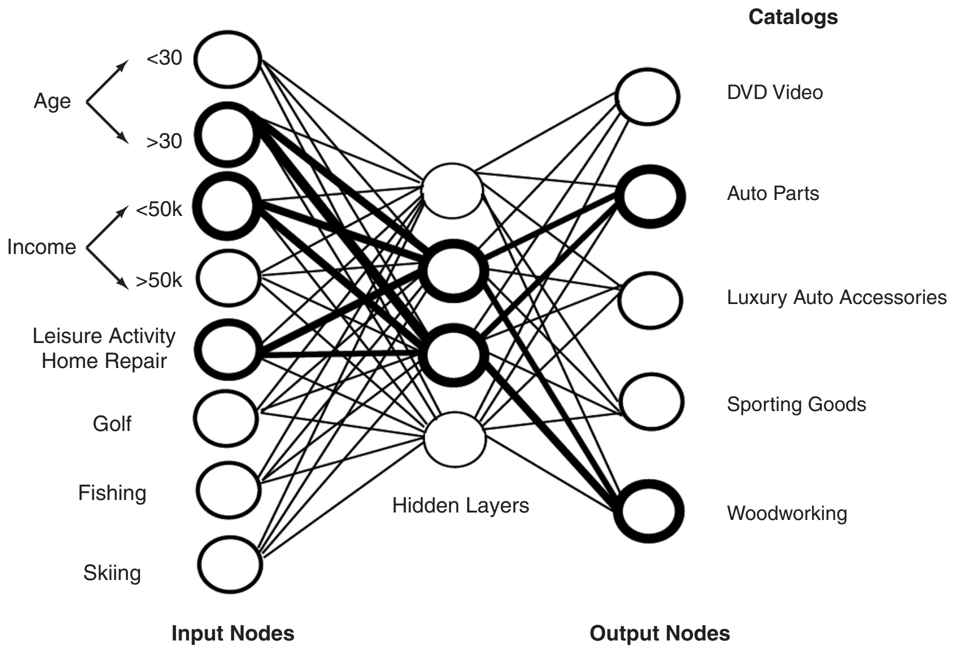


**Figure 3** Neural network identifying catalog purchasers.

added together, and a function applied (sometimes referred to as the activation function) which produces a result that is passed (fed forward), in this case, to the output node. This value is compared to the actual output value from the customer. An error is computed between the actual customer target value and the value from the network. If not within desired accuracy this error is fed back to the hidden layer where the weights are adjusted. This process continues for each case until the desired or target output is produced within the tolerance desired or until the error can not be reduced.

If there was no hidden layer and the activation function was linear, then the neural net would be equivalent to simple linear regression assuming there is only one output node.

In Fig. 3, the heavy dark lines indicate the connection between the customer profile and the catalogs from which they purchased. The typical person in this data set that purchased from the woodworking and auto parts catalogs was over age 30, makes under $50,000 per year, and repairs their home as a leisure activity.

There are advantages and disadvantages to neural nets. The advantages of neural nets are that they can fit nonlinear relationships by addition of hidden layers. They require less prior knowledge of possible interactions between variables. In logistic regression the analyst must specify the interactions before building the model whereas in neural networks the hidden layers take care of this without analyst intervention. Given enough time they can fit almost any data, however, the question is whether this fit will hold up under cross validation and testing.

The disadvantages are that they often overfit the data which means that they do not do well when applied to new data. If the data has a lot of measurement error the neural network will fit all of this error or noise and the model will not be robust when applied to new data. The weights from a neural network are not meaningful (they are in regression) in interpreting the relationships among the variables and in understanding the model.

One of the most valuable aspects of neural networks is to serve as a benchmark or goal for other methods. Recall that a good approach to model building in DM is to fit several different algorithms to the model and compare results. If a neural network is performing better than other viable models, it is a strong indication that the other models can be improved. The advantage of other models is that they may lend more insight into the situation being modeled or they may be easier to deploy and interpret than a neural network.

There are many variations of neural networks. One with an interesting potential is the Radial Basis Function neural network. This network does not have hidden nodes but maps the inputs to outputs using a weighted sum of basis functions. It is a very robust and fast computing function approximation technique. To date, most implementations use the Gaussian as the basis function. There is some evidence to indicate that other basis functions including a cubic spline will produce better results especially with time series data. This might be a useful research area for those interested.

## C. Decision Trees

Decision trees produce models that are easy to understand if the number of branches remain relatively small. If there are many variables then the tree can become complex and uninterpretable. Decision trees work by doing successive binary splits (some algorithms do produce more than two branches at each split). The first split will yield the biggest separation or distinction in two groups of data. Each subgroup is then split until some stopping criteria are reached. Algorithms differ partly on how they measure the separation distance between groups. Two different algorithms on the same data set would very likely give different branches and rules. Decision trees can result in very easy to understand decision rules.

For example, in Fig. 4, one rule for a good loan risk is: A good loan risk had 4 jobs or less in the last 4 years and has an income greater than $50,000. If the tree results in many branches then the rules would obviously become very complicated and perhaps unusable. There are methods to prune trees to keep them useful. Some of the different algorithms include chi-squared automatic interaction (CHAID), classification and regression trees (CART), and C5.0.

Decision trees produce rules. Another method referred to as rule induction also produces rules similar to a decision tree, however, it does not produce a tree. Since there is no tree and no forced splits at each level, a rule induction algorithm may produce better classification rules.

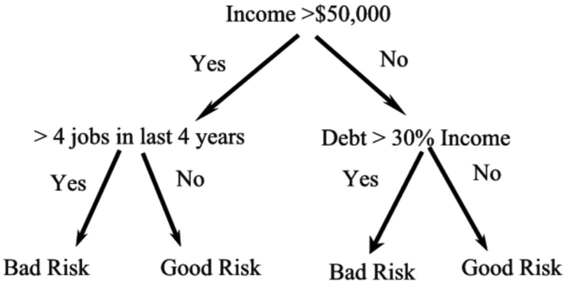


**Figure 4** Simple decision tree.

## D. MARS

Statistician Jerome Friedman invented CART and then in mid-1980 developed a method that attempts to overcome the disadvantages of CART. It produces a decision tree and can thus produce rules associated with the tree. Friedman's MARS was intended to eliminate at least two of the problems of CART including (1) dependence of all splits on previous splits and (2) reduced interpretability of splits due to interactions among variables. MARS does not produce a tree or rules but does find the most important predictor variables and their interactions. It also relates the dependence of the target on each predictor somewhat in the way that a multiple regression model would.

What does MARS stand for? It stands for a very complex algorithm, multivariate adaptive regression splines. There is only one well-known implementation of MARS available. The commercial implementation took several years to perfect, however, the code is still available as freeware from the Web. A recent possible successor to MARS is MART, multiple additive regression trees.

## E. K-Nearest Neighbor or Memory-Based Reasoning

This algorithm relates cases and groups them based on some distance measure. The distance measure that is used is very important and also can be difficult to define. For example, what would be the distance between red hair and blond hair in skin cancer data? Once defined, the distance must be summed between attributes to calculate and assign the cases and neighbors to groups. The algorithm is not complex but is computer-intensive as each new case requires that the distance be computed between each case. This method is also referred to as memory-based reasoning because often all of the data is kept in main memory to speed the calculations of the distances. Terabytes of data pose restrictions on this algorithm.

# VII. OTHER METHODS OF DATA MINING AND FUTURE TRENDS

The techniques described here represent general categories of DM algorithms. As the area becomes mature there will be many more algorithms developed. New algorithms are needed that deal effectively with terabytes of data. K-nearest neighbor methods can be improved to be faster and more efficient. Decision tree models may soon be made more accurate through the use of boosting and bagging. Radial basis functions may be adapted to terabyte databases better than traditional neural networks because of their computational efficiency. Genetic algorithms can be useful in improving the learning process of DM algorithms. Sophisticated visualization methods can improve understanding of the data and thus improve models. A current trend is to develop visual representations of variable relationships to help understand resulting models. Existing algorithms have helped to understand what DM can do. New and improved algorithms will help do more. Currently the businesses that use DM are enjoying a competitive edge. When everyone is using DM then new methods and applications will need to be invented to improve that edge. Some of the classic methods of decision theory are being applied to DM algorithms to help sharpen the DM edge. E-commerce is offering a tremendous challenge and opportunity for DM. There may be a point where no further advances can be made. That point seems far in the future.

## SEE ALSO THE FOLLOWING ARTICLES

Database Development Process • Data Warehousing and Data Marts • Decision-Making Approaches • Decision Theory • Hybrid Systems • Machine Learning • Neural Networks • Supply Chain Management

## BIBLIOGRAPHY

Adriaans, P., and Zantinge, D. (1996). *Data mining*. Harlow, England: Addison-Wesley.

Berry, M. J. A., and Linoff, G. S. (2000). *Mastering data mining*. New York: Wiley.

Berthold, M., and Hand, D. J. (1999). *Intelligent data analysis*. Berlin, Heidelberg: Springer-Verlag.

Friedman, J. H., Stone, C. J., Breiman, L., and Olshen, R. A. (1984). *Classification and regression trees*. Boca Raton, FL: CRC Press, LLC.

KD Nuggets. http://www.kdnuggets.com/. A noncommercial web site that lists the latest software, technology, publications, jobs, etc. in data mining.

Kennedy R. L., Lee Y., Van Roy, B., Reed, C. D., and Lippman, R. P. (1997). *Solving data mining problems through pattern recognition*. Upper Saddle River, NJ: Prentice Hall.

Mena, J. (1999). *Data mining your website*. Boston, MA: Digital Press.

Pyle, D. (1999). *Data preparation for data mining*. San Francisco, CA: Morgan Kaufmann Publishers.

Two Crows Corporation (1999). *Data mining '99 technology report*. Potomac, MD: Two Crows Corporation. www.twocrows.com.

Witten, I. H., and Frank E. (1999). *Data mining: Practical machine learning tools and techniques*. San Francisco, CA: Morgan Kaufmann.

# Data Modeling: Entity-Relationship Data Model

**Salvatore T. March**
*Vanderbilt University*

## GLOSSARY

**attribute** Name and specify the characteristics or descriptors of entities and relationships that must be maintained within an information system. Attributes are typically single-valued.

**data model** An implementation independent representation of the data requirements of an information system. Data models may be very broad in scope, representing a large number of application areas or they may be very narrow in scope, representing a single application.

**data modeling** A process by which the data requirements of an application area are identified and represented. Data modeling is often a component of information system development methodologies.

**data modeling formalism** A set of constructs and rules used in the representation of data. Example data modeling formalisms include the entity-relationship model and the binary-relationship model.

**entity** A category or grouping of objects (e.g., people, things, events) or roles of objects (e.g., customer, employee) each sharing a common set of characteristics or *descriptors* (e.g., all employees have employee numbers, names, addresses, pay rate, etc.).

**identifier** A combination of attributes and relationships whose values uniquely distinguish the instances of an entity. An entity may have multiple, alternate identifiers. Identifiers are often *artificial* in the sense that an identifying attribute is created for the entity and values assigned to its instances (e.g., employee number or customer number).

**relationship** A named association among entities. A relationship defines the entities involved and the roles played by each in that association. A relationship may associate instances of the same entity or different entities.

## I. INTRODUCTION

### A. The Purpose of Data Modeling

*Data modeling* is a process by which the data requirements of an application area are represented in an implementation-independent way. A *data modeling formalism* defines a set of *constructs* and rules used in the representation of data. The product of data modeling process is a *data model* represented in some data modeling formalism.

Effective information system development requires: (1) accurately representing the data and processing requirements of the application area; (2) validating those requirements with end users; and (3) transforming the data requirements into a database schema and the processing requirements into computer programs (in the target database management system (DBMS)/programming language environment).

A data model represents the "things" and "events" that are a part of the application (e.g., customers, inventory items, order placement and shipment), their characteristics (e.g., customers are identified by customer number and are described by customer name, credit limit, etc.), and their relationships (e.g., each order must be associated with a single customer). A data model can be validated by end users to insure accuracy of the data requirements. These can then be transformed into a database implementation.

## B. Data Models and Database Implementations

A data model does not specify the physical storage of the data. It provides a precise representation of the data content, structure, and constraints required by an application. These must be supported by the database and software physically implemented for the application. The process of developing a database implementation *(schema)* from a data model is termed *physical database design*. In short, the data model defines what data must be represented in the application and the database schema defines how that data is stored. The goal of data modeling, also termed *conceptual database design,* is to accurately and completely represent the data requirements. The goal of physical database design is to implement a database that efficiently meets those requirements.

Clearly there must be a correspondence between a data model and the database schema developed to implement it. For example, a data model may specify that each employee must report to exactly one department at any point in time. This is represented as a relationship between employees and departments in the data model. This relationship must have a physical implementation in the database schema; however, how it is represented is not of concern to the data model. That is a concern for the physical database design process. In a relational DBMS (RDBMS), relationships are typically represented by primary key-foreign key pairs. That is, the department identifier (primary key) of the department to which an employee reports is stored as a column (foreign key) in the employee's record (i.e., row in the Employee table). In an object DBMS relationships can be represented in a number of ways, including complex objects and embedded object identifiers (OIDs).

Numerous data modeling formalisms have been proposed; however, the entity-relationship (ER) model and variations loosely termed binary-relationship models are the most widely known and the most commonly used. Such formalisms have come to be known as *semantic* data models to differentiate them from the storage structures used by commercial DBMSs to define a database schema. Data modeling has become a common component of system development methodologies. A number of object-oriented system development approaches, such as the Unified Modeling Language, have extended data models into what has been termed *class diagrams*. These use the same basic constructs as data models to represent the semantic data structure of the system, but typically extended the representation to include operations, system dynamics, and complex constraints and assertions.

## C. Overview of the Article

This article describes data modeling and more specifically the ER and binary relationship models. Section II introduces basic data modeling constructs and Section III introduces alternative graphical notations. Section IV discusses more advanced concepts and constructs developed to enable data models to represent a richer set of semantics. Section V presents techniques for developing data models and criteria by which to evaluate and improve them. Section VI illustrates how data models are transformed into a relational database schema and discusses implementation efficiency issues. Finally, Section VII summarizes the article and presents directions for further research.

# II. BASIC ENTITY-RELATIONSHIP CONSTRUCTS

As originally proposed by Peter P.-S. Chen in 1976, the ER mdel has four basic constructs, entity, relationship, attribute, and identifier. These have become the basis for essentially all data and object structure modeling formalisms. They are first defined and then illustrated in an example data model.

## A. Entity

An *entity* is a category or grouping of objects (e.g., people, things, events) or roles of objects (e.g., customer, employee), each sharing a common set of characteristics or *descriptors* (e.g., all employees have employee numbers, names, addresses, pay rate, etc.). The individual members of a category are termed *entity-instances* or just *instances*. Each instance must be uniquely identified within the context of the domain being modeled (e.g., the employee with employee number 12314, the customer with customer number 5958). Chen's original proposal referred to the category as an *entity set* and to the instances as *entities*. The important consideration is to distinguish the category or type from its instances.

Defining entities and their instances is not always as simple as it sounds. Consider, for example, a product identified by product number 1242. Does this product number identify a single "thing" (instance) or a category of "things" (entity)? If it is a category, does the context require differentiating the instances? The answer is often domain dependent. Suppose product number 1242 is defined as "box of 24 No. 2 red pencils." Clearly there are numerous units (instances) of that thing in inventory; product number 1242 does not refer to a specific one of them but simultaneously to any one of them and to the collection of them. Characteristics such as weight and volume are likely to be ascribed to it (referring to each unit) as well as characteristics such as quantity on hand and warehouse location (referring to the collection of units). Hence, although it is a category it is treated as an instance of the entity product.

Suppose, on the other hand, that product number 1242 is defined as "1988 Porsche 944." Again, there may be multiple units (instances) of that thing in inventory; however, a unique vehicle identification number (VIN) identifies each of them. Characteristics such as weight and wheelbase can be ascribed to "1988 Porsche 944;" however, characteristics such as color and mileage must be ascribed to the specific instance. In this case, "1988 Porsche 944" is an instance of Product, however, "the 1988 Porsche 944 having VIN 34FRD88JH99HHY" is an instance of *it*. Although "1988 Porsche 944" is a category and could be defined as an entity, it is more likely that a more general category, such as "Car" would be more appropriate.

## B. Relationship

A *relationship* is a named association among entities. A relationship defines the entities involved and the roles played by each in that association. A relationship may associate instances of the same entity or different entities. For example, *prerequisite* is the named relationship associating two instances of the entity Course; *reporting* is the named relationship associating an instance of Employee with an instance of Department; *sale* is the named relationship associating an instance of Customer, an instance of Salesperson, and an instance of Product.

A relationship can be characterized by its *degree* and its *cardinality*. The degree of a relationship is the number of entities it associates. The cardinality of a relationship describes the number of times an instance of an entity may participate in a relationship. Cardinality has also been termed *connectivity* or *multiplicity*. Each of these characteristics is discussed below.

### 1. Relationship Degree

A relationship associating instances of the same entity, e.g., prerequisite is termed a *unary* or *recursive* relationship. It is said to have a degree of 1. A relationship associating instances of two different entities, e.g., reporting is termed a *binary* relationship (degree 2). A relationship associating instances of three entities, e.g., sale is termed a *ternary* relationship (degree 3). Generally a relationship associating instances of N entities is termed an *N-ary* relationship (degree N). The original ER model supports N-ary relationships. The binary relationship models restrict relationships to at most binary. The implications of this restriction are discussed below.

It is often important to distinguish the "roles" played by the entities in a relationship, particularly when a relationship associates instances of the same entity or when it is not clear from the entities themselves. In the relationship prerequisite, for example, it is crucial to distinguish which instance of Course plays the role "has-prerequisite" and which plays the role "is-prerequisite-for." Specifying that the courses Computer Science 101 and Mathematics 220 participate in the relationship named "prerequisite" is not very useful until the roles are specified. Typically this specification utilizes one role or the other to form a sentence: "Computer Science 101 has-prerequisite Mathematics 220" or "Mathematics 220 is-prerequisite-for Computer Science 101." In the relationship reporting, the roles of Employee and Department are clear, Employee instances "report-to" Department instances or Department instances "are the reporting units for" Employee instances.

### 2. Relationship Cardinality

A relationship is also characterized by its *cardinality*, i.e., the number of times an instance of each entity can participate in the relationship. Hence, a relationship has a cardinality for each participating entity. More precisely, a relationship has a minimum and a maximum cardinality for each participating role. Unary and binary relationships are often characterized by the maximum cardinality of each role as being "one-to-one," "one-to-many," or "many-to-many." The relationship prerequisite, for example, is a many-to-many relationship since a course can *have* many prerequisite courses and a course can *be* a prerequisite for many courses. Continuing with the above example, Computer Science 101 may require not only Mathematics 220, but also Computer Science 100. Furthermore, Mathematics 220 may be a prerequisite not only for Computer Science 101 but also for Accounting 1000 and Marketing 1015.

The cardinality of the relationship reporting, on the other hand, is not so clear. If an employee can report to only one department at a time, a department can have many employees report to it at the same time, and departmental reporting history is not maintained, then it is a one-to-many relationship (one department to many employees). If, on the other hand, employees can report to multiple departments at the same time or the relationship includes reporting history over time then it is a many-to-many relationship. A data model must explicitly represent which of these alternatives is appropriate for the domain being modeled. The "right" representation is clearly domain dependent. Determining it is the task of the data modeler or systems analyst.

While it is important to determine the maximum cardinality of a relationship, it is just as important to determine its minimum cardinality. Typically the minimum cardinality for a relationship is either 0 or 1, although it could be larger. Considering the prerequisite relationship, clearly it is not necessary for a course to have a prerequisite or to be a prerequisite for other courses. Hence the minimum cardinality for each role is 0. On the other hand, considering the reporting relationship, an organization may require, for budgetary reasons, that an employee always report to a single department but that a department need not have any employees report to it. In this case the minimum cardinality is 1 for Employee and 0 for Department—each employee must participate in exactly one reporting relationship; each department may participate in 0 or more reporting relationships.

If the minimum cardinality of a role in a relationship is 1 or greater, then the entity playing that role is said to be *dependent* upon the other entity (or entities) in that relationship. That is, an instance of that entity cannot exist without an associated instance of the other(s). This type of dependency is often termed a *referential integrity* or *existence dependency*. If the minimum cardinality of a role is zero, then the entity is said to be *independent* of the other entity (or entities) in the relationship; it can exist without participating.

## C. Attribute

Attributes name and specify the characteristics or descriptors of entities and relationships that must be maintained within an information system. Each instance of an entity or relationship has a value for each attribute ascribed to that entity or relationship. Chen defined an attribute as a *function* that maps from an entity or relationship instance into a set of values. The implication is that an attribute is single valued—each instance has exactly one value for each attribute. Some data modeling formalisms allow multivalued attributes, however, these are often difficult to conceptualize and implement. They will not be considered in this article.

Returning to the definition of an entity, the "common set of characteristics or descriptors" shared by all instances of an entity is the combination of its attributes and relationships. Hence an entity may be viewed as that collection of instances having the same set of attributes and participating in the same set of relationships. Of course, the context determines the set of attributes and relationships that are "of interest." For example, within one context a Customer entity may be defined as the collection of instances having the attributes customer number, name, street address, city, state, zip code, and credit card number, independent of whether that instance is an individual person, a company, a local government, a federal agency, a charity, or a country. In a different context, where the type of organization determines how the customer is billed or even if it is legal to sell specific product to that instance, these same instances may be organized into different entities and additional attributes may be defined for each.

## D. Identifier

Each entity has at least one combination of attributes and relationships whose values uniquely distinguish its instances. This unique combination is termed an *identifier.* An entity may have multiple, alternate identifiers. Given the value of an entity's identifier, there is at most one corresponding instance of that entity in the domain of interest. Employee number 12314, for example, identifies one specific employee just as customer number 5958 identifies one specific customer. Social Security Number 111–11–1111 may alternately identify the same employee.

Identifiers are often *artificial* in the sense that an identifying attribute is created for the entity and values assigned to its instances. Employee number and customer number are examples of artificial identifiers. The reason is that often there is not a set of "natural" attributes that are guaranteed to uniquely distinguish all instances of an entity, e.g., there may be several employees with the same name, or even the same name and address. Even Social Security Number is an artificial identifier assigned by the federal government.

## III. GRAPHICAL REPRESENTATIONS

There are two major graphical representations for a data model. The first was posed by Chen in 1976 and is most properly called the ER model. It uses boxes to represent entities, embedded boxes to represent dependent entities, diamonds connected to entities to represent relationships, ovals to represent attributes, and the identifier underlined (the original proposal did not include a graphical representation for attributes, this was added later). Minimum and maximum relationship cardinality are written beside the line connecting each entity with the relationship diamond.

An alternate binary relationship graphical representation was popularized, among others, by a system development methodology called information engineering (IE). It too uses boxes to represent entities but uses a line to represent relationships and includes attributes within the entity box. It uses the "chicken feet" notation to represent relationship cardinality. It does not allow relationships to have attributes, does not support many-to-many relationships or relationships with degree higher than 2, i.e., binary relationships. Equivalent ER and IE representations are shown in Figs. 1 and 2, respectively. These representations are compared and explained in the next section.

### A. Interpretation and Comparison

The ER representation (Fig. 1) has three entities, Department, Employee, and Project and three relationships, Managing (one-to-one), Reporting (one-to-many), and Assign (many-to-many). Maximum and minimum cardinality is shown for each entity in each relationship. The Reporting relationship, for example, specifies that each employee must report to exactly one department (1:1 specifying a minimum of 1 and a maximum of 1). A department can have zero or many employees Reporting to it (0:M specifying a minimum of 0 and a maximum of Many). Each entity has attributes shown in ovals connected to the appropriate entity. The Assign relationship also has attributes, again shown in ovals connected to it. Department, Employee, and Project have (artificial) attribute identifiers eno (Employee Number), dno (Department Number), and pno (Project Number), respectively. These are underlined. Relationships without attributes are assumed to be identified by the combination of their related entity instances. Hence, Managing instances are identified by the combination of the department and the managing employee and Reporting instances are identified by the employee and the reporting department. A relationship can include any of its attributes in its

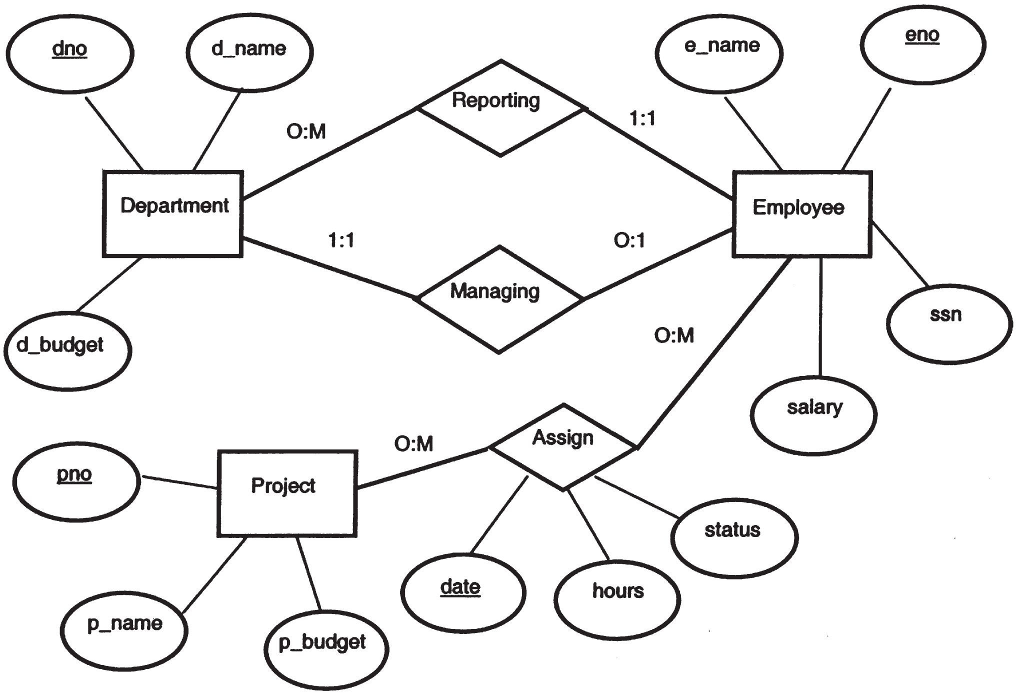


**Figure 1** An entity-relationship graphical representation.

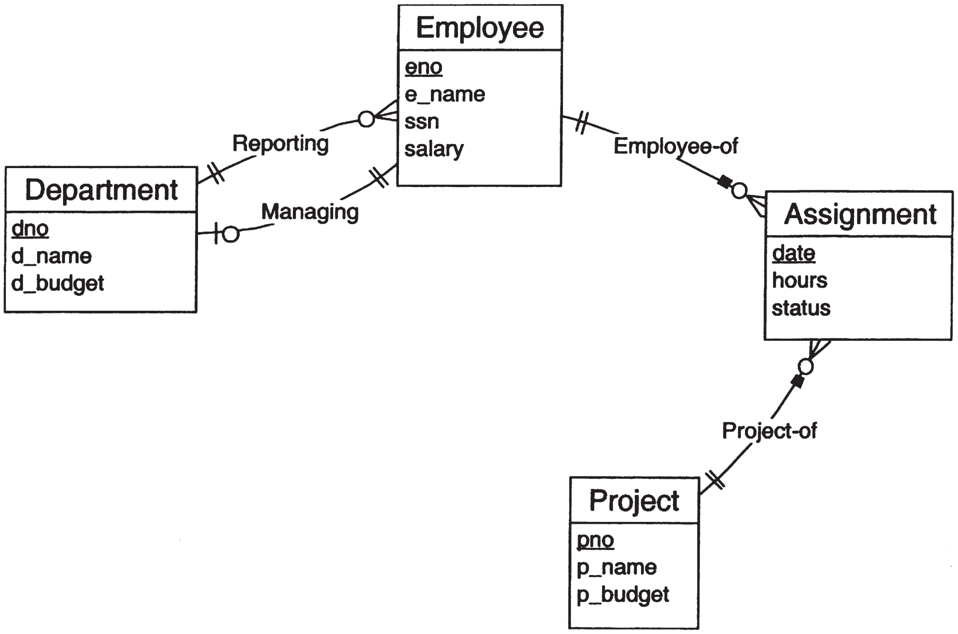


**Figure 2** A binary relationship graphical representation.

identifier. Assign instances are identified by the combination of the employee and the project to which the employee is assigned and the date on which the assignment was made (date).

The binary relationship (IE) representation (Fig. 2) also includes the entities, Employee, Department, and Project. Again, entities are represented by boxes, but the name of the entity is shown at the top of the box and the attributes are listed inside the box. The Managing (one-to-one) and Reporting (one-to-many) relationships are represented as lines with minimum and maximum cardinalities shown symbolically on the line. The symbol for maximum cardinality is located closest to the entity, that for minimum cardinality next to it. The Reporting relationship, for example, has a chicken foot on the Employee entity specifying that a maximum of many employees can report to one department. It has a 0 next to the chicken foot specifying that a minimum of 0 employees can report to 1 department. It has a 1 next to Department specifying that an employee can report to a minimum of 1 department and another 1 next to it specifying that an employee can report to a maximum of 1 department. Again, identifier attributes are underlined.

The major difference between the formalisms is how to represent the fact that employees are assigned to projects on a specific date (many-to-many), and that additional facts must be maintained relevant to that assignment, such as the hours spent and the current status. Since the IE representation does not support many-to-many relationships or relationships with attributes, it includes a fourth entity, Assignment, having one-to-many relationships with Employee and Project and the attributes that the ER representation ascribes to the relationship Assign. Its identifier is the combination of these relationships and the date on which the assignment was made (date). Each of these relationships has a solid square box on it near the Assignment entity to represent the fact that they are part of the identifier. Effectively the IE approach defines Assignment as an entity while the ER approach defines it as a relationship. While there are numerous debates about the benefits of each of these approaches, the IE approach is less cluttered and more intuitive. It will be used in the remainder of this article.

Notice that both representations convey the same semantics. Each thing is specified as having the same descriptors and constraints. Each employee *reports* to exactly one department (hence Employee is existent dependent upon Department within the database). A department may *employ* zero or more employees (departments can exist without employees). A department is managed by at most one employee. An employee manages at most one department. Not all employees participate in this relationship, all departments do. That is, not all employees manage a department, but all departments must have a manager.

As discussed above, the major difference between these formalisms is the manner in which they represent certain types of relationships. These are, many-to-many relationships, relationships having attributes, and relationships having a degree of 3 or higher. Such "relationships" are represented directly in the ER model but require the creation of an *intersection* entity in the IE model. While this may seem somewhat artificial, experience suggests that such "relationships" are often a major focus of the information system and perceived by the users as being "entities" (categories of things). Frequently they represent "events" occurring at a point in time, or having some time duration, about which information must be maintained. Representing them as entities enhances their importance and simplifies the model.

## IV. ADVANCED CONCEPTS

### A. Generalization

A common criticism of early data modeling formalisms was the lack of an abstraction capability. Process modeling was based upon the concept of functional decomposition; that is, breaking a complex process down into simpler ones. Data models, on the other hand, tended to be relatively "flat." Each entity was defined as a set of nonoverlapping instances. This makes it difficult to adequately model certain types of data.

Consider, for example, the way in which managers are represented in Figs. 1 and 2. A manager is an Employee, but a special kind of Employee, one with a special function in the organization and perhaps having different characteristics from other employees. The fact that the Managing relationship between Employee and Department has a minimum degree of 0 indicates that there is some heterogeneity among the instances of Employee—some are managers, some are not. This heterogeneity forces application programs that access the database to treat some employees differently from others. The data model does not adequately represent the domain of interest.

Smith and Smith introduced the notion of *generalization* as a mechanism by which to increase the fidelity of data models to a domain of interest. Generalization allows the instances of different entities to overlap and provides a means for defining the nature of that overlap.

A more accurate representation of the employees, managers, and their association with departments uses generalization as shown in Fig. 3. Employee is a generalization or *supertype* or *superclass* of Manager. Conversely Manager is a specialization or *subtype* or *subclass* of Employee. This is indicated by the specialization arc containing a triangle pointing from Manager to Employee. Generalization is a special case of a one-to-one relationship, termed an ISA (is a) relationship. Each instance of Manager ISA instance of Employee, but only some instances of Employee are also instances of Manager. Using generalization it is clear that the Managing relationship applies to only a subset of employees.

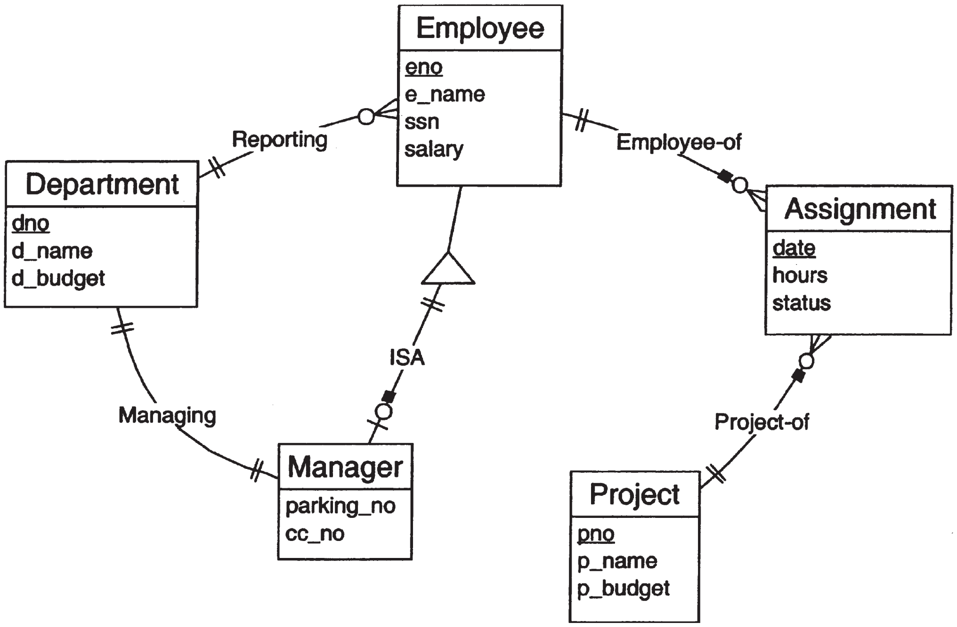


**Figure 3** Manager as a subtype of employee.

Since each Manager ISA Employee, all attributes of Employee apply to all instances of Manager. It is said that Manager *inherits* all properties (descriptors) from Employee. This, so-called *property inheritance* is fundamental to the object-oriented models such as the unified modeling language (UML).

Often a subtype such as Manager will have additional descriptors that do not apply to the supertype. Manager, for example, has the attributes parking space number and credit card number that are not applicable to any other employees. Of course, Manager participates in the relationship Managing that does not apply to other employees.

Whenever there is an entity with a relationship whose minimum degree is 0 or whenever there are attributes that do not apply to all instances of an entity, generalization can be used to decompose that entity into one or more subtypes. By using generalization in this way the fidelity of the data model to the domain of interest is increased. However, using generalization in this way also increases the complexity of the diagram. It is suggested that generalization only be used when entity subsets (such as managers) are routinely distinguished within the application area.

## B. Generalization Constraints

The data model in Fig. 3 illustrates a single subtype that is completely contained in a single supertype, i.e., every Manager ISA Employee. It is also possible for a supertype to have multiple, possibly overlapping, subtypes and for entities to be subtypes of more than one supertype. The last is termed a *generalization lattice* having *multiple inheritance.* These situations may require the specification of *generalization constraints* to clarify the nature of these relationships.

While a complete discussion of such constraints is beyond the scope of this article it is instructive to examine the three most common subtype constraints. These are exclusion, cover, and partition. They refer to a collection of subtypes of a single supertype. Exclusion specifies that each instance of the supertype can be in at most one of the subtypes. Cover specifies that each instance of the supertype must be in at least one of the subtypes. Partition specifies that an instance of the supertype must be in exactly one of the subtypes.

Consider, for example, a supertype Client having a collection of three subtypes, Customer, Vendor, and Employee. If it is not possible for the same Client instance to be categorized more than one of these subclasses, then an Exclusion constraint exists. That is, an Exclusion constraint precludes a client from being a customer and a vendor or a customer and an employee or a vendor and an employee. If a client must be at least one of the subtypes, customer, vendor, or employee, then a Cover constraint exists. That is, there cannot be any other types of clients than customers, vendors, and employees. If a client must be exactly one of customer, vendor, or employee, then a Partition constraint exists.

## C. Time-Dependent Data

It is frequently important to representing time-dependent or *temporal* aspects of an application within a data model. That is, specific applications frequently require not only the current values of attributes and relationships, but also their past values. Data models are frequently developed to represent current values only. Consider the data model represented in Fig. 2. It specifies that each employee has exactly one value of `eno`, `e_name`, `ssn`, and `salary`, i.e., these are attributes and that an employee reports to exactly one department. Clearly, over time, an employee can legally change his name, hopes for salary increases, and is likely to change departments. Hence, over time an employee can have many values of `e_name` and `salary` and can be related to many departments. If it is important to include such temporal aspects of these attributes and relationships within the domain of interest, then they must be represented in the data model.

A number of temporal data models have been proposed. These typically extend the basic data modeling constructs to include time dependencies. Typically attributes and relationships are designated as being either static or temporal. Designating an attribute or relationship as "temporal" explicitly specifies that history must be maintained for that attribute or relationship. If, for example, salary and departmental reporting history were to be maintained, the salary attribute and Reporting relationship would simply be designated as being temporal. It is left to the designer of the physical database to determine how that history is maintained. Frequently it is assumed that a temporal database management system will be used.

A more semantic approach to representing temporal data is to identify the events that cause the values of attributes and relationships to change and represent them as entities. These event entities include the time designation of when the event occurred and the new values or incremental changes enabling the calculation of the new values. In this approach the event causing salary to change would be identified and an entity would be created for it. Suppose this event is called Performance Review. This entity would be one-to-many related to Employee and have attributes such as the date on which the salary review took place and the new salary. Similarly, the event causing the Reporting relationship to change would be identified and related to both Employee and Department.

## V. TECHNIQUES FOR BUILDING A DATA MODEL

Data modeling is a part of an overall process of developing a system representation. Numerous techniques have been proposed; however, the process remains highly subjective. The key tasks are (1) identify the major entities, including supertypes and subtypes; (2) establish relationships; (3) determine attributes and identifiers; and (4) document relationship cardinalities and other constraints. The developed data model must be validated with other representations of the information system such as process and behavior representations.

Three commonly used approaches to modeling the data content of a domain of interest are *sentence* analysis, *document* analysis, and *event* analysis. Each provides a different perspective from which to view the domain. They are most effectively used in combination.

Obtaining sentences describing the tasks and processes required in a domain is relatively straightforward. Knowledgeable users are simply asked to "tell their story." The resultant sentences serve as exemplars of tasks and processes performed in the domain that must be supported by the information system. The data requirements implicit in those descriptions are extracted and formally represented in a data model. Analyzing the documents, including transactions and reports, used in performing those tasks and processes provides insight into current processes and enables the identification of opportunities for process improvements. Event analysis focuses on identifying and describing, what happens (the events), who is involved (the actors and business resources), and what responses are required as a result. Each of these techniques is described below and a data model created to illustrate this process.

### A. Sentence Analysis

Sentence analysis simply identifies subjects, verb phrases, and objects. Often these are in the form of specific instances and must be generalized to the type level. If the subject and object are both entities, then the verb phrase represents a relationship, typically stated in the form of the role of the entity represented by the subject. If the subject represents an entity but the object represents a fact about that entity, then the object is an attribute and the verb phrase explains the meaning of the attribute. This distinction is typically made within the scope of the domain under consideration and must be generalized to identify opportunities for integration with other systems. Consider the following sentences that have already been generalized to the type level.

- Salespeople service Customers.
- Customers place Orders through a Salesperson.
- Freight is determined when an Order is Shipped.
- Salespeople are paid commission based on their commission rate and Invoiced sales.
- Each Salesperson has a number, name, and address.
- Each Customer has a number and a bill-to-address.
- Each Shipped Order results in an Invoice for which the Customer is responsible.

The first two sentences specify relationships, Salesperson-Customer, Customer-Order, and Salesperson-Order. Additional analysis would be required to determine the nature of these relationships and if any are derived. For example, suppose the Salesperson who gets credit for the Order is always the same as the Salesperson who services the Customer who placed the Order. Then the relationship between Salesperson and Order can be derived from the relationship between Salesperson and Customer combined with the relationship between Customer and Order. It would not be necessary to keep a direct relationship between Salesperson and Order. However, if salespeople occasionally go on vacation or if customers are occasionally moved from one salesperson to another and sales history must be maintained with salespeople, then likely, this relationship would need to be maintained explicitly. The next four sentences specify attributes. Salesperson has the attributes, commission rate, number, name, and address. Customer has the attributes number and bill-to-address. The last sentence equates Orders and Invoices.

### B. Document Analysis

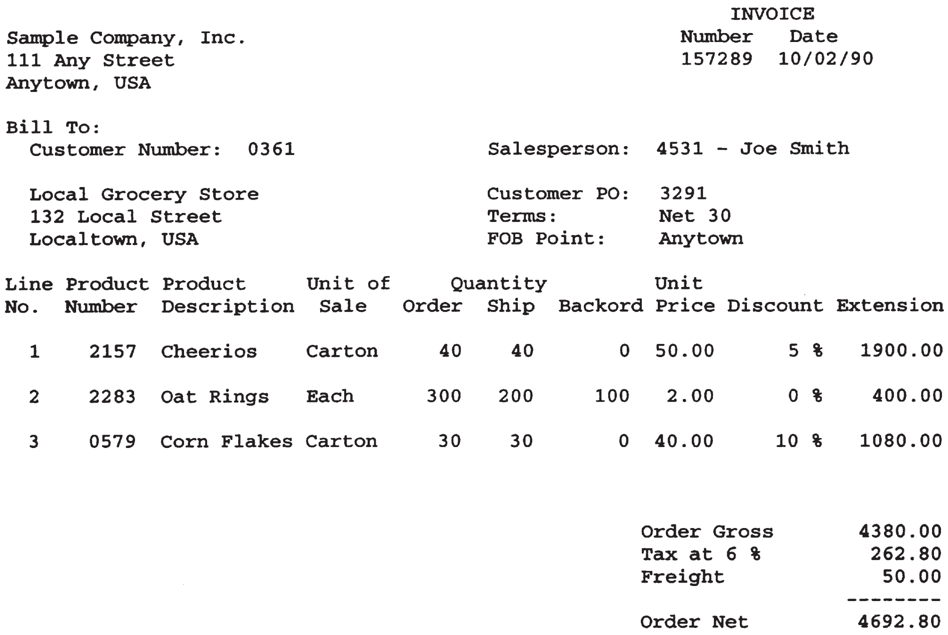
Document analysis similarly looks for facts represented in the documents used in a business process. To analyze a document, each heading on the document is classified as representing: (1) an entity (its identifier); (2) an attribute; or (3) calculated data. This determination is done by an analyst in conjunction with the end users; however, as mentioned above, it is extremely common for an organization to create artificial identifiers for important entities. These identifiers invariably appear on reports or transaction documents as `<entity>` Number (or No. or num) or as some other obvious designator of an entity.

Consider the invoice document illustrated in Fig. 4. A scan of that document reveals 22 headings: Invoice (Order) Number, Date, Customer Number, Salesperson, Bill-to-Address, Customer PO, Terms, FOB Point, Line Number, Product Number, Product Description, Unit of Sale, Quantity Ordered, Quantity Shipped, Quantity Backordered, Unit Price, Discount, Extension, Order Gross, Tax, Freight, and Order Net. The fact that these are on the same document indicates that there is some association among these various items. They represent descriptors of related entities.

To develop a data model from this document, the headings are categorized into one of three classes: entity identifier, attribute, or calculated value. This is somewhat subjective, but recalling the definition of an entity as any thing about which information must be maintained, the following are initially classified as entity identifiers: Invoice Number (equivalent to Order Number), Customer Number, Salesperson (with an implicit Salesperson number), and Product Number. Tax, Order Gross, and Order Net are classified as calculated values. All others are classified as attributes. For each entity identifier, the entity is named: Invoice Number identifies Invoice, Customer Number identifies Customer, and Product Number identifies Product.

Next, relationships are established. It is obvious that Customer relates to Invoice. Given the existence of a relationship the next questions that must be asked are: “How many Customers are responsible for a single Invoice?” (answer: exactly one) and “How many Invoices can be the responsibility of one Customer?” (answer: zero or more). In this way it is determined that there is a one-to-many relationship between Customer and Invoice (see Fig. 5).

Similarly a many-to-many relationship is established between Invoice and Product—one Invoice contains many Products (this is obvious), and that the *same* Product can be on many Invoices. How can the *same* product be on more than one Invoice? This assumes that what is meant by the *same* Product is **not** the same physical instance of the product, but different instances of the same *type* of product, having a single Product Number, where the instances of which are completely interchangeable. That is, the customer is being invoiced for some *quantity* of the *same* product. In Fig. 3, for example, “Cheerios” is Product Number 2157, sold in units of cartons. Local Grocery Store is being invoiced for 40 cartons of this *same* product.



```
                                                           INVOICE
Sample Company, Inc.                                    Number   Date
111 Any Street                                          157289  10/02/90
Anytown, USA

Bill To:
  Customer Number:  0361              Salesperson:  4531 - Joe Smith

  Local Grocery Store                 Customer PO:  3291
  132 Local Street                    Terms:        Net 30
  Localtown, USA                      FOB Point:    Anytown

Line Product Product     Unit of     Quantity         Unit
No.  Number  Description  Sale   Order  Ship  Backord Price Discount Extension

  1    2157  Cheerios    Carton     40    40       0  50.00      5 %   1900.00

  2    2283  Oat Rings   Each      300   200     100   2.00      0 %    400.00

  3    0579  Corn Flakes Carton     30    30       0  40.00     10 %   1080.00


                                                Order Gross       4380.00
                                                Tax at 6 %         262.80
                                                Freight             50.00
                                                                 --------
                                                Order Net         4692.80
```

**Figure 4** An example invoice document.

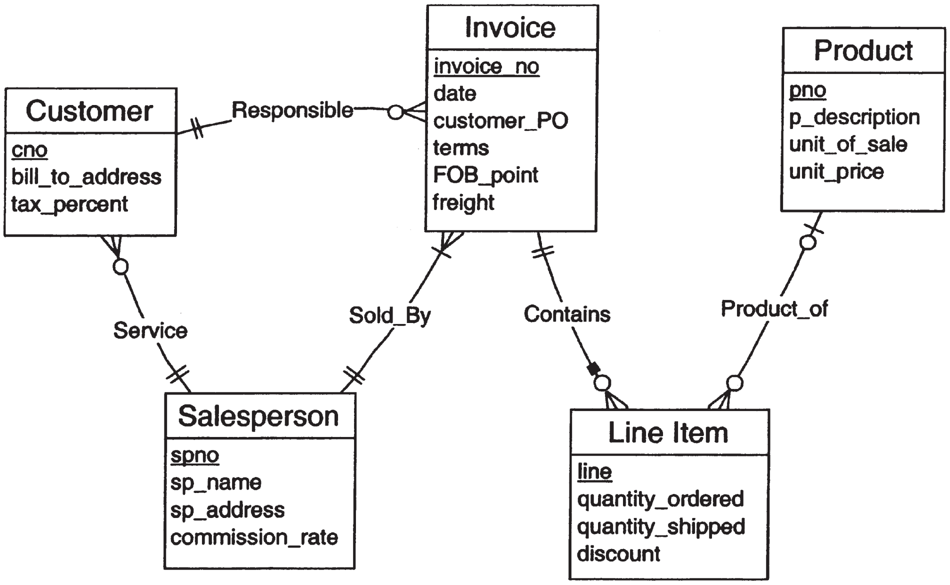


**Figure 5** A data model for an order processing application.

Presumably 40 (or 50 or 100) more cartons of the *same* product could be ordered, shipped, and invoiced to a different Customer. Hence the relationship is many-to-many.

As many-to-many relationships are not permitted in this formalism, an intersection entity must be created between Invoice and Product. The intersection entity is related many-to-one to each of the entities in the many-to-many relationship (chicken feet on the intersection entity). An obvious name for the intersection entity is *Line Item,* since, in common business terminology, a line item on an invoice represents a single product on the invoice. Each Line Item is identified by the combination of the Invoice to which it relates and the Line Number (line) as illustrated in Fig. 5.

Finally, attributes are identified from the values on the document and associated with the appropriate entity. Frequently attributes are named for the heading under which the value appears, e.g., Customer PO, Terms, FOB Point, Product Description. In other cases the value is the result of a calculation, e.g., Extension, Order Gross, and Tax at 6%. When that is the case, the calculated value itself does not correspond to an attribute, however, all data needed to perform the calculation must be represented within the attributes. For example, Extension is defined by the calculation,

$$\text{Order Quantity} * \text{Unit Price} * (1 - \text{Discount}/100).$$

Order Quantity, Unit Price, and Discount are each represented as attributes. Extension is not. Similarly Order Gross is the sum of all Extension calculations for each Line Item on the Invoice. Tax at 6% is also a calculated value, however, it requires an additional attribute, Tax Rate. The appropriate entity for this attribute must be determined. There may be a single Tax Rate that applies to each Customer. Alternately there may be a different Tax rate for each State in which Customers reside. The domain for which the data model in Fig. 5 was developed has a potentially different tax rate for each customer. Hence, `tax_percent` is an attribute of Customer. If there is not an appropriate entity for an attribute, i.e., one that is uniquely described by an attribute, then there is a missing entity. Further analysis is required to identify it.

Finally, the model is reviewed to identify subtypes. If a relationship applies to some, but not all instances of an entity, that is, the minimum degree of one of the relationship descriptors is zero, then subtypes exist in at least one of the entities, i.e., the subtype participates in the relationship while the supertype does not.

Similarly, if an attribute applies to some, but not all instances of an entity, then subtypes exist for the entity, i.e., the subtype has the additional attributes. The value of explicitly recognizing subtypes depends on the degree of heterogeneity among the subtypes. The purpose of the data model is to communicate the meaning of the data. If the introduction of subtypes confuses rather than clarifies, they should not be introduced.

Referring to the data model of Fig. 5, some Customers may not have any outstanding Invoices (minimum cardinality of zero). Thus there are two subtypes of Customer—those with Invoices and those without

Invoices. If this is the only distinction for subtyping, it is probably not worthwhile explicitly recognizing the subtypes. If, on the other hand, Customers with Invoices have additional attributes or are viewed differently from Customers without Invoices, then it may be beneficial to represent this subtype explicitly in the data model.

## C. Event Analysis

Event analysis defines an entity for each event and identifies the associated actors and resources required for that event to occur. The sentences in the Section V.A not only identify things, they also identify events. At least three events can be distinguished, Place Order, Ship Order, and Invoice Order. If additional sentences were obtained, it is likely that a fourth event would be identified, Pay Invoice. Event analysis can often help clarify the nature of attributes and relationships. Invoice, for example, has an attribute named date. The identification of four events related to this entity forces the question: "To which event does this date refer?" Most likely four different date attributes must be maintained, one for each event. Furthermore, the attribute freight does not have a meaningful value for an Order that has not been shipped. Similarly, for the Line Item entity, the attribute `quantity_shipped` is not meaningful until the Ship Order event has occurred.

Event methodologies recommend creating an entity for each event. Hence, Place Order, Ship Order, Invoice Order, and Pay Invoice would be separate entities in the Order processing data model. Such a representation highlights the possibility that a single Order may have many Ship events or that a single Ship event may include multiple Orders. It effectively forces an analyst to investigate such possibilities.

## D. Evaluating the Resultant Data Model

The quality of the data model is assured by the manner in which it was constructed. The construction described above is based on the principle of *normal forms* originally proposed for the relational data model and later applied to semantic data models. Each attribute is associated with an entity *only* if that attribute directly describes that entity with a single value. In the "normal form" terminology, the assignment of attributes to entities in this way assures that each attribute is *fully functionally dependent* upon the identifier of the entity, and not dependent upon any other attribute in the model.

Disallowing many-to-many relationships assures that each relationship corresponds to a functional dependency between entity identifiers. The identifier of the entity on the "one" side is fully functionally dependent upon the identifier of the entity on the "many" side. Thus the resultant data model is "well formed" in the sense that it can be directly transformed into a "third normal form" relational schema.

The structural rules for evaluating a data model are

1. Each entity must be uniquely named and identified
2. Attributes are associated with entities (not relationships), and each entity must have one and only one value for each of its attributes (otherwise an additional entity must be created)
3. Relationships associate a pair of entities or associate an entity with itself (only binary relationships are allowed but relationships can be recursive)
4. Many-to-many relationships are not allowed (an intersection entity with two one-to-many relationships must be created)
5. Subtypes are identified when the minimum degree of a relationship descriptor is zero or when an attribute does not apply to all instances of an entity; these subtypes are explicitly recognized when there is a "significant" difference among the subtypes (e.g., they have multiple attributes or relationships)

# VI. TRANSFORMING DATA MODELS INTO DATABASE DESIGNS

After validation a data model must be transformed into the schema definition of a DBMS for implementation. This section describes how data models can be transformed into an RDBMS schema and presents the basic efficiency issues that must be considered in that transformation.

## A. Relational DBMS

The basic elements of a relational database schema define tables, columns in those tables, and constraints on those columns. Constraints include primary keys (identifiers) and foreign keys (relationships). While there are numerous efficiency issues related to transforming a data model into such a database schema, a "quick and dirty" approach simply maps entities into tables, attributes into columns, identifiers into primary keys, and relationships into columns designated as foreign keys.

Fig. 7 shows such a relational schema in tabular form for the data model of Fig. 5. Figure 6 shows its definition in the standard relational language SQL. The data model has five entities, Customer, Salesperson, Order, Line Item, and Product. Five corresponding tables are defined in the relational schema. Similarly, columns are defined for each attribute in each entity. In a relational schema all interconnections among tables are represented by data values. Hence, columns must be created to represent relationships. For a one-to-many relationship, a column is created in the table representing the entity on the "many" side for each attribute (column) in the identifier (primary key) of the table representing the entity on the "one" side. These columns are termed "foreign keys."

As illustrated in Figs. 6 and 7, the Customer table has an "extra" column, spno (salesperson number) to represent the one-to-many relationship between Salesperson and Customer. It is designated in a Foreign Key constraint to reference the Primary Key column, spno, in the Salesperson table. The spno column in the Customer table is constrained to be `NOT NULL`. This constraint specifies that each row of the Customer table must have a value for that column. The Foreign Key constraint specifies that the value of that column must appear in the spno column of some row in the Salesperson table. Together these implement the specified relationship between Customer and Salesperson in the data model having a minimum 1 and maximum 1 on the Customer side. Removing the `NOT NULL` constraint would implement a minimum 0 and maximum 1 relationship on the Customer side. The minimum 0 and maximum many (unlimited) on the Salesperson side is implicit in the Foreign Key representation. Enforcing a minimum other than 0 or a maximum other than unlimited on the Salesperson (many) side requires a procedurally implemented constraint.

The Line Item table similarly has two "extra" columns: invoice_no (invoice number), representing its many-to-one relationship with Invoice, and pno (product number) representing its many-to-one relationship with Product. Again each is designated in a Foreign Key constraint; however, while invoice_no is designated to be NOT NULL, pno is not. This implementation requires each Line Item to have a related Invoice, but does not require it to have a related Product. Hence this schema implements the specified relationships in the data model of Fig. 5.

A one-to-one relationship may be represented in the table corresponding to either entity in the relationship. When one role in a one-to-one relationship has a minimum of 0 and the other has a minimum of 1, the relationship is most commonly represented by a foreign key in the table corresponding to the entity on the minimum 0 side. Consider, for example, the data model in Fig. 3. The Managing relationship could be represented by a foreign key in either the Employee table or the Department table. Since it has a minimum of 0 on the Department side, it would most likely be represented by a foreign key column constrained to be NOT NULL in the table corresponding to that entity. If this relationship was represented in the table corresponding to the Employee table, the foreign key column would not be constrained to be NOT NULL and, in fact, would contain a NULL value in each row corresponding to an employee who did not manage a department, likely most of them.

## B. Efficiency Issues in Relational Database Design

This type of direct transformation from a data model into a database schema may be inefficient for database processing. Decisions related to efficiency of implementation are part of physical database design. There are numerous physical database design possibilities. The following are illustrative.

### 1. Vertical and Horizontal Fragmentation

For efficiency reasons, an entity may be split vertically or horizontally. This is termed fragmentation. Vertical fragmentation assigns subsets of attributes to different tables. Horizontal fragmentation assigns subsets of instances to different tables. Of course, these approaches can be combined.

**Salesperson (spno, sp_name, sp_address, commission_rate)**

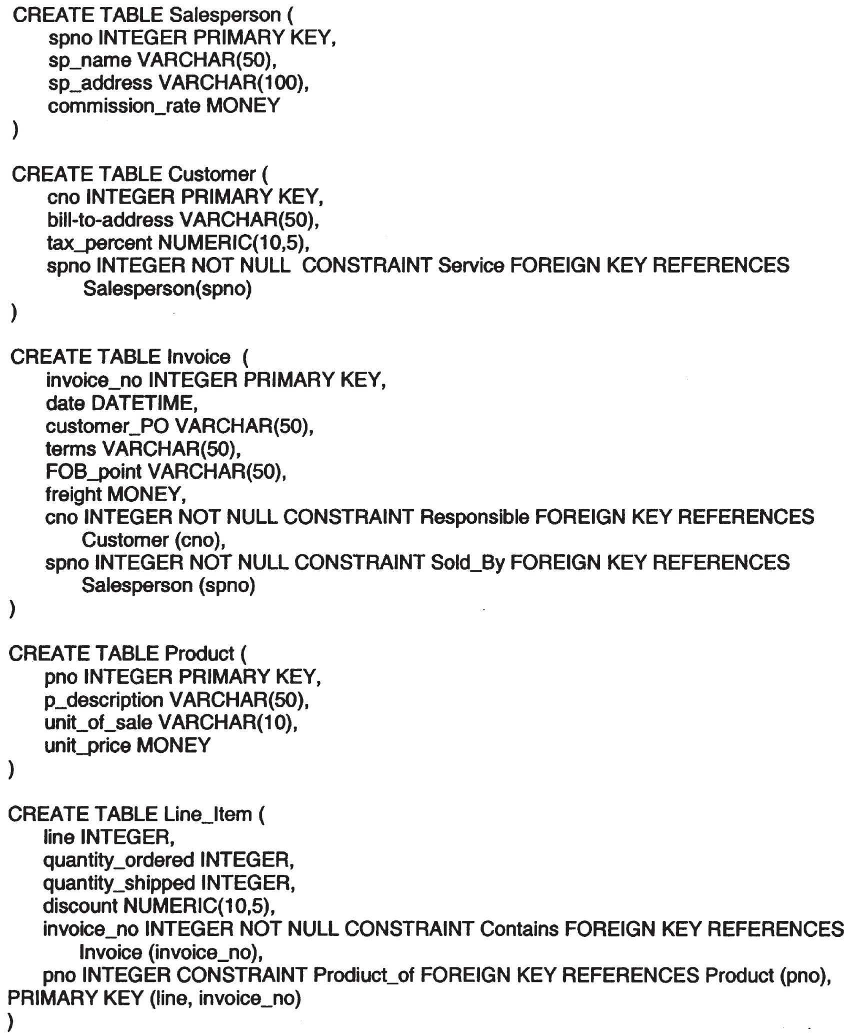
**Customer (cno, bill-to-address, tax_percent, *spno*)**

**Invoice (invoice_no, date, customer_PO, terms, FOB_point, freight, *cno*, *spno*)**

**Product (pno, p_description, unit_of_sale, unit_price)**

**Line_Item (line, *invoice_no*, quantity_ordered, quantity_shipped, discount, *pno*)**

**Figure 6** A relational database schema in tabular form (primary keys are underlined, foreign keys are in italic).



```
CREATE TABLE Salesperson (
     spno INTEGER PRIMARY KEY,
     sp_name VARCHAR(50),
     sp_address VARCHAR(100),
     commission_rate MONEY
)

CREATE TABLE Customer (
     cno INTEGER PRIMARY KEY,
     bill-to-address VARCHAR(50),
     tax_percent NUMERIC(10,5),
     spno INTEGER NOT NULL  CONSTRAINT Service FOREIGN KEY REFERENCES
          Salesperson(spno)
)

CREATE TABLE Invoice  (
     invoice_no INTEGER PRIMARY KEY,
     date DATETIME,
     customer_PO VARCHAR(50),
     terms VARCHAR(50),
     FOB_point VARCHAR(50),
     freight MONEY,
     cno INTEGER NOT NULL CONSTRAINT Responsible FOREIGN KEY REFERENCES
          Customer (cno),
     spno INTEGER NOT NULL CONSTRAINT Sold_By FOREIGN KEY REFERENCES
          Salesperson (spno)
)

CREATE TABLE Product (
     pno INTEGER PRIMARY KEY,
     p_description VARCHAR(50),
     unit_of_sale VARCHAR(10),
     unit_price MONEY
)

CREATE TABLE Line_Item (
     line INTEGER,
     quantity_ordered INTEGER,
     quantity_shipped INTEGER,
     discount NUMERIC(10,5),
     invoice_no INTEGER NOT NULL CONSTRAINT Contains FOREIGN KEY REFERENCES
          Invoice (invoice_no),
     pno INTEGER CONSTRAINT Prodiuct_of FOREIGN KEY REFERENCES Product (pno),
PRIMARY KEY (line, invoice_no)
)
```

**Figure 7** A relational database schema definition in SQL.

Vertical fragmentation can increase efficiency if clusters of "frequently" used and "rarely" used attributes can be identified. Frequently used attributes can be stored in the table corresponding to the entity and rarely used attributes can be stored in a separate table. This reduction in the size of the table corresponding to the entity can significantly reduce processing requirements. Consider, for example, an Employee entity used in a payroll application. The Employee entity may contain attributes such as emergency contact, emergency contact address, and emergency contact telephone number that are rarely, if ever, used in payroll processing. Segmenting them to a separate table, related to the Employee table by a foreign key reduces the size of the Employee table, which is likely scanned for payroll processing.

Horizontal fragmentation can similarly increase efficiency by reducing the size of the table corresponding to an entity by identifying "rarely" used instances. This technique is similar to "archiving" rarely used data. Again considering an Employee entity used in a payroll application, terminated employees must be retained for end of year processing, but are no longer used in normal payroll processing. Segmenting them to a separate table can reduce normal payroll processing time.

## 2. Attribute Replication

Efficiency can be increased by redundantly storing specific attributes with related entities. Consider, for example, the processing required to produce invoices from the set of tables shown in Fig. 5. Assuming that invoices require Customer, Order, Salesperson, Line Item, and Product data, the production of this report requires all five tables to be joined together. Redundantly storing the required Salesperson attributes in the Order table and the required Product attributes in the Line Item table would eliminate two joins, significantly reducing the computing effort to produce it. Of course this replication increases the size of those tables and can result in update anomalies since the third normal form is violated. This approach is most effective when the replicated data is small and is not subject to frequent modification.

### 3. Entity Merging

Merging related entities that are frequently retrieved together into a single table may also increase retrieval efficiency. Merging the entity on the one side of a one-to-many relationship into the entity on the many side is similar to attribute replication, but replicates the entire entity. This approach can be effective when combined with judicious vertical fragmentation. Merging the entity on the many side of a one-to-many relationship into the entity on the one side violates first normal form. It is not directly supported by RDBMSs although it is directly supported by object DBMSs. To implement this strategy in an RDBMS a set of columns must be defined for the maximum possible number of related instances. It is most effective when the actual number of related instances is fixed and relatively "small." Consider, for example, an inventory system that tracks the ending inventory level for each product for each of the past 12 months. The data model for such a domain would have an entity for Product related one to many to an entity for Ending Inventory. The Ending Inventory entity would need two attributes, Month and Ending Quantity. The minimum and maximum cardinality on the Ending Inventory side is 12. Merging that entity into the Product entity would require 12 columns, one for the ending inventory in each of the past 12 months. The month would be built into the column name.

## VII. SUMMARY AND DIRECTIONS FOR FURTHER RESEARCH

Data modeling is a process by which the logical or "natural" data structure of a domain of interest is represented using a predefined set of constructs. The set of constructs is termed a data modeling formalism. The product of data modeling is a logical data model.

This article has discussed the major constructs of data modeling formalisms. It used a simple graphical notation to illustrate these constructs. Approaches to developing a data model were presented and illustrated. Finally, a simple way to transform a logical data model into a database schema was presented and efficiency issues discussed.

Current research in data modeling is progressing in several directions. These include the development of modeling and design tools and techniques, user interfaces based on data modeling constructs, usability, semantics and constraints, quality and reliability metrics, and the interface with object technologies and languages. A number of these are addressed in the articles listed in the bibliography.

## SEE ALSO THE FOLLOWING ARTICLES

Database Administration • Database Development Process • Database Systems • Data Modeling: Object-Oriented Model • Relational Database Systems

## BIBLIOGRAPHY

Carlis, J. V., and Maguire, J. (2000). *Mastering data modeling: A user-driven approach.* Reading, MA: Addison-Wesley.

Chen, P. P.-S. (1976). The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems,* Vol. 1, No. 1, 9–36.

Denna, E., Cherrington, J. O., Andros, D., and Hollander, A. (1993). *Event-driven database solutions.* Irwin, Homewood, IL: Business One.

Fowler, M., Kendall, S., and Booch, G. (1999). *UML distilled,* Second Edition. Reading, MA: Addison-Wesley.

Hull, R., and King, R. (1987). Semantic database modelling: Survey, applications, and research issues. *ACM Computing Surveys,* Vol. 19, No. 3, 201–260.

Ling, T. W., and Ram, S. (1998). *Conceptual modeling—ER '98.* Berlin: Springer.

McFadden, F. R., Hoffer, J., and Prescott, M. (1999). *Modern database management,* Oracle 7.3.4 Edition. Reading, MA: Addison-Wesley.

Peckham, J., and Maryanski, F. (1988). Semantic data models. *ACM Computing Surveys,* Vol. 20, No. 3, 153–189.

Snodgrass, R. T. (2000). *Developing time-oriented database applications in SQL.* San Francisco, CA: Morgan Kaufmann.

Teorey, T. (1994). *Database modeling and design, the fundamental principles,* Second Edition. San Francisco, CA: Morgan Kauffmann.

# Data Modeling: Object-Oriented Data Model

**Michalis Vazirgiannis**
*Athens University of Economics and Business*

## GLOSSARY

**data model** In data modeling we try to organize data so that they represent as closely as possible a real world situation, yet representation by computers is still feasible. A data model encapsulates three elements: objects' structure, behavior and integrity constraints.

**encapsulation** A property of the object-oriented model, promoting data and operation independence. This is achieved by hiding the internal structure and implementation details from the external world simplifying thus the maintenance and usage of a multitude of object classes in an application.

**inheritance** The ability of one class to inherit the structure and behavior of its ancestor. Inheritance allows an object to inherit a certain set of attributes from another object while allowing the addition of specific features.

**object-oriented modeling** It is an abstraction of the real world that represents objects' structural content and behavior in terms of classes' hierarchies. The structural content is defined as a set of attributes and attached values and the behavior as a set of methods (functions that implement object's behavior).

**polymorphism** The ability of different objects in a class hierarchy to have different behaviors in response to the same message. Polymorphism derives its meaning from the Greek for "many forms." A single behavior can generate entirely different responses from objects in the same group. Within the framework of the program, the internal mechanism determines what specific name of different purposes is known as function overloading.

**relational model** A data modeling approach having found very successful industrial implementations in DBMS. The fundamental modeling constructs are the relations consisting of tuples of values each one taking its semantics from an appropriate attribute. The relations represent entities of the real world and relationships among entities.

## I. INTRODUCTION

### A. Need for Data Modeling

The word "datum" comes from Latin and, literally interpreted, means a fact. However, data do not always correspond to concrete or actual facts. They may be imprecise or may describe things that have never happened (e.g., an idea). Data will be of interest to us if they are worth not only thinking about, but also worth recording in a precise manner.

Many different ways of organizing data exist. For data to be useful in providing information, they need to be organized so that they can be processed effectively. In data modeling we try to organize data so that they represent as closely as possible a real world situation, yet representation by computers is still feasible. These two requirements are frequently conflicting. The optimal way to organize data for a given application can be determined by understanding the characteristics of data that are important for capturing their meaning. These characteristics allow us to make general statements about how data are organized and processed.

It is evident that an interpretation of the world is needed, sufficiently abstract to allow minor perturbations, yet sufficiently powerful to give some understanding concerning how data about the world are related. An intellectual tool that provides such an interpretation will be referred to as a data model. It is a model about data by which a reasonable interpretation of the data can be obtained. A data model is an abstraction device that allows us to focus on the information content of the data as opposed to individual values of data.

## B. Historical Overview: First and Second Database Model Generations

Information systems demand more and more services from information stored in computing systems. Gradually, the focus of computing shifted from process-oriented to data-oriented systems, where data play an important role for software engineers. Today, many design problems center around data modeling and structuring.

After the initial file systems in the 1960s and early 1970s, the first generation of database products was born. Database systems can be considered intermediaries between the physical devices where data are stored and the users (humans) of the data. Database management systems (DBMS) are the software tools that enable the management (definition, creation, maintenance, and use) of large amounts of interrelated data stored in computer-accessible media. The early DBMSs, which were based on hierarchical and network (Codasyl) models, provided logical organization of data in trees and graphs. IBM's IMS, General Electric's IDS, Univac's DMS 110, Cincom's Total, MRI's System 200, and Cullinet's (now Computer Associates) IDMS are some of the well-known representatives of this generation. Although efficient, these systems used procedural languages, did not offer *physical* or *logical independence,* thus limiting its flexibility. In spite of that, DBMSs were an important advance compared to the files systems.

IBM's addition of data communication facilities to its IMS software gave rise to the first large-scale database/data communication (DB/DC) system, in which many users access the DB through a communication network. Since then, access to DBs through communication networks has been offered by commercially available DBMSs.

C. W. Bachman played a pioneering role in the development of network DB systems (IDS product and Codasyl DataBase Task Group, or DBTG, proposals). The DBTG model is based on the data structure diagrams, which are also known as Bachman's diagrams. In the model, the links between record types, called Codasyl sets, are always one occurrence of one record type. To many, that is, a functional link. In its 1978 specifications, Codasyl also proposed a data definition language (DDL) at three levels (schema DDL, subschema DDL, and internal DDL) and a procedural (prescriptive) data manipulation language (DML).

In 1969–1970, Dr. E. F. Codd proposed the *relational model,* which was considered an "elegant mathematical theory" without many possibilities of efficient implementation in commercial products. In 1970, few people imagined that, in the 1980s, the relational model would become mandatory (a "decoy") for the promotion of DBMSs. Relational products like Oracle, DB2, Ingres, Informix, Sybase, etc., are considered the second generation of DBs. These products have more physical and logical independence, greater flexibility, and declarative query languages (users indicate what they want without describing how to get it) that deal with sets of records, and they can be automatically optimized, although their DML and host language are not integrated. With relational DBMSs (RDBMSs), organizations have more facilities for data distribution. RDBMSs provide not only better usability but also a more solid theoretical foundation.

Unlike network models, the relational model is value-oriented and does not support object identity. Needless to mention, there is an important trade-off between object identity and declarative features. As a result of Codasyl DBTG and IMS support object identity, some authors introduced them in the object-oriented DB class.

The initial relational systems suffered from performance problems. While nowadays these products have achieved wide acceptance, it must be recognized that they are not exempt from difficulties. Perhaps one of the greatest demands on RDBMSs is the support of increasingly complex data types; also, null values, recursive queries, and scarce support for integrity rules and for domains (or abstract data types) are now other weaknesses of relational systems. Some of those problems are solved in the current version of Structured Query Language (SQL), SQL: 1999 (previously SQL3).

In the 1970s, the great debate on the relative merits of Codasyl and relational models served to compare both classes of models and to obtain a better understanding of their strengths and weaknesses.

During the late 1970s and in the 1980s, research work (and, later, industrial applications) focused on query optimization, high-level languages, the normalization theory, physical structures for stored relations, bugger and memory management algorithms, indexing techniques (variations of B-tress), distributed systems, data dictionaries, transaction management, and so on. That work allowed efficient and secure on-line transactional processing (OLTP) environments (in the first DB generation, DBMSs were oriented toward batch processing). In the 1980s, the SQL language was also standardized (SQL/ANS 86 was approved by the American National Standard Institute, ANSI and the International Standard Organization, ISO in 1986), and today, every RDBMS offers SQL.

Many of the DB technology advances at that time were founded on two elements: reference models and data models. ISO and ANSI proposals on reference models have positively influenced not only theoretical researches but also practical applications, especially in DB development methodologies. In most of those reference models, two main concepts can be found; the well-known three-level architecture (external, logical, and internal layers), also proposed by Codasyl in 1978, and the recursive data description. The separation between logical description of data and physical implementation (data application independence) devices was always an important objective in DB evolution, and the three-level architecture, together with the relational data model, was a major step in that direction.

In terms of data models, the relational model has influenced research agendas for many years and is supported by most of the current products. Recently, other DBMSs have appeared that implement other models, most of which are based on object-oriented principles.

Three key factors can be identified in the evolution of DBs: theoretical basis (resulting from researcher's work), products (developed by vendors), and practical applications (requested by users). These three factors have been present throughout the history of DB, but the equilibrium among them has changed. What began as a product technology demanded by users' needs have always influenced the evolution of DB technology, but especially so in the last decade.

Today, we are witnessing an extraordinary development of DB technology. Areas that were exclusive of research laboratories and centers are appearing in DBMSs' latest releases: World Wide Web, multimedia, active, object-oriented, secure, temporal, parallel, and multidimensional DBs. The need for exploiting the Object-Oriented Model for such complex systems is apparent.

## II. MOTIVATION AND THEORY

### A. Motivation

Although one might think that DB technology has reached its maturity, the new DB generation has demonstrated that we still ignore the solutions to some of the problems of the new millennium. In spite of the success of this technology, different "preoccupation signals" must be taken into account. We identify the following *architectural* issues that need to be solved in the light of new application domains:

- Current DBMSs are monolithic; they offer all kinds of services and functionalities in a single "package," regardless of the users' needs, at a very high cost, and with a loss of efficiency
- About half of the production data are in legacy systems
- Workflow management (WFM) systems are not based on DB technology; they simply access DBs though application programming interfaces (APIs)
- Replication services do not scale well over 10,000 nodes
- Integration of strictly structured data with loosely structured data (e.g., data from a relational DB with data from electronic mail)

On the other hand there is wealth of new application domains that produce huge amounts of data and therefore call for database support. Such domains are computer-aided design (CAD), computer-aided software engineering (CASE), office-automation, multimedia databases, geographic information systems (GIS), scientific experiments, telecommunications, etc.

These application domains present some important common characteristics that make their database support by traditional relational systems problematic. Such features include:

- Hierarchical data structures (complex objects)
- New data types for storing images or large textual items
- No general-purpose data structure available
- Nonstandard application-specific operations
- Dynamic changes
- Cooperative design process among designers
- Large number of types
- Small number of instances
- Longer duration transactions

The database technology has to respond to these challenges in a way that the above requirements are addressed as database technology design features. In the sequel we identify the shortcomings of current database technology in the context of the new applications:

- Poor representation of "real-world" entities, need to decompose objects over relations

- Fixed build-in types; no set-valued attributes are supported, thus complex and highly nested objects cannot be represented efficiently
- Semantic overloading
- Poor support for integrity and enterprise constraints
- No data abstraction such as aggregation and generalization, thus inheritance and specialization cannot be addressed
- Limited operations
- Difficulty handling recursive queries
- No adequate version control is supported

## B. Object-Oriented Model

### 1. Historical Preview of Object-Oriented Databases

Before we proceed with our discussion of data modeling, it is necessary to define, even if only approximately, the elementary objects that will be modeled (i.e., what a datum is). Suppose that we accept as a working definition of an atomic piece of data the tuple <*object name, object property value, time*>. After all, a phenomenon or idea usually refers to an object *(object name)* and to some aspect of the object *(object property),* which is captured by a value *(property value)* at a certain time *(time).*

Of these four aspects of data, time is perhaps the most cumbersome aspect of data modeling. Therefore, many data models completely drop the notion of time and replace it either with other kinds of explicit properties or with orderings among objects. The issues of Object-Oriented (OO) Models, object structure–object Classes will be treated as the basis of an object-oriented database management system (OODBMS). Furthermore such a system must support Inheritance (single–multiple) and handle object identity issues. Then OO languages providing persistency (persistence by class, creation, marking, reference) are necessary so that users of an OODBMS are able to define and manipulate database objects.

### 2. Object-Oriented Modeling and Programming Concepts

Hereafter an overview of object-oriented concepts will be presented. Object orientation has its origins in object-oriented programming languages (OOPLS). The "class" concept is introduced by SIMULA, where as abstract data types encapsulation, message passing, and inheritance features are further introduced by the pioneering SMALLTALK. Another language of this family is C++ that integrates the strengths of C with object-oriented concepts. The newest OO language is Java, inherently object-oriented providing a wide selection of classes for different tasks (i.e., visualization, network and task management, persistent features). Its portability across platforms and operating systems made it a very attractive development environment, widely used and with important impact on programming large-scale applications.

An object has an inherent state followed by its behavior, which defines the way the object treats its state as well the communication protocol between the object and the xternal world. We have to differentiate between the transient objects in OOPLs and the persistent objects—in object-oriented databases (OODBs). In the first case the objects are eliminated from the main memory as soon as they are not needed whereas in the case of OODBMSs objects are persistently stored and other mechanisms such as indexing, concurrency, control, and recovery are available. OODBMSs usually offer interfaces with one or more OOPLs.

The three features that differentiate an object from a relational tuple are

1. Object identity, a unique identifier generated with the object creation and follows it throughout its life cycle
2. Encapsulation features (promoting data and operation independence), since the internal structure and implementation details are not accessible from the external world simplifying thus the maintenance and usage of a multitude of object classes in an application
3. Operator polymorphism and overloading, allowing different behaviors to be grouped under the same method and operator names; this facilitates design and evolution of large sets of classes

Here after these features are further analyzed.

#### a. Object Identity, Object Structure and Type Constructors

The object identity (OID) is generated by the system when a new object is created and is unique and immutable. Each OID is used only once and is invisible to the users.

An OODBMS offers a set of type constructors that are used to define the data structures for an OO database schema. The basic constructors offered created atomic values (atom (integer, string, float, etc.), tuples, and sets of objects. Other constructors create lists, bags, and arrays of objects. It is also feasible to have attributes that refer to other objects called references—OID.

An object has an internal structure defined by a triple (OID, type constructor, state). For instance, assume an object o = ($i_1$, tuple, <manager: $i_3$, start-date: $i_5$>). An object can be represented as a graph structure that can be constructed by recursively applying the type constructors. Assume the following example:

```
define type Employee:
   tuple    ( fname:     string;
              minit:     char;
              lname:     string
              ssn:       string;
              birthdate: date;
              address:   string;
              sex:       char;
              salary:    float;
              supervisor: Employee;
              dept:      Department;  );
Define type Date
    tuple   (  year;     integer;
              month:     integer;
              day:       integer;  );
Define type Department:
   tuple    ( dname:     string;
              dnumber: integer;
               mgr:      tuple (
                         manager:
                          Employee;
                          startdate:
                           Date;);
               location: set(string);
              employees:
                     set(Employee);
              projects:
                    set(Project); );
```

**Example 1** Object identities (OIDs).

In this example the types Employee, Date, and Department are defined. As the object structures are potentially complex the issue of object equality becomes interesting. Object equality is a concept that can be viewed from two aspects: deep and shallow.

- Two objects are said to have identical states (deep equality) if the graphs representing their states are identical in every respect, including the OIDs at every level.
- Two objects have equal states (shallow equality) if the graph structures are the same and all the corresponding atomic values in the graphs are also the same. However, some corresponding internal nodes in the two graphs may have objects with *different OIDs.*

### b. Encapsulation

An important feature in the domain of OODBs is the encapsulation that offers an abstraction mechanism and contributes in information hiding since an object encapsulates both data and functions into a self-contained package. Thus the external aspects of an object are separated from its internal details. The external users of the object are only made aware of the *interface* of the object type, i.e., name and parameters of each operation, also called signature.

The functionality of an object is implemented by a set of *methods* and *messages.* A method consists of a name and a body that perform the behavior associated with the method name, whereas a message is simply a request from one object to another object asking the second object to execute one of its methods.

In the context of database applications dealing with objects we distinguish between visible attributes, implemented by external operators or query language predicates or hidden attributes which are referenced only through predefined operations. Assume the example:

```
Define class Employee;
type tuple ( ...)
  operations    age:          integer;
                create_emp:   Employee
                destroy_emp:  boolean;
end Employee;
```

If d is a reference to an Employee object, we can invoke an operation such as age by writing d.age. Of course the main issue in OODBs is to have a mechanism that makes the objects persistent, i.e., stored safely in the secondary storage at appropriate times. The mechanisms for making an object persistent are called naming and reachability. The naming mechanism involves giving an object a unique persistent name through which it can be retrieved by this and other programs. The following example illustrates this concept:

```
 define class DepartmentSet:
 type ...
 operations ...
...
 end DepartmentSet;
 ...
persistent name AllDepartments:
    DepartmentSet;
 ...
 d := create_dept;
```

### c. Type Hierarchies and Inheritance

A type can be defined by giving it a type name and then listing the names of its public functions. Then it is possible to define subtypes emanating from the basic types. For example:

```
PERSON: Name, Address, Birthdate,
  Age, SSN
EMPLOYEE subtype-of PERSON: Salary,
  HireDate, Seniority
STUDENT subtype-of PERSON: Major,
  GPA
```

Each subtype inherits the structure and behavior of its ancestor. It is also possible to redefine (override) an inherited property or method.

In some cases it is necessary to create a class that inherits from more than one superclass. This is the case of multiple inheritance which occurs when a certain subtype T is a subtype of two (or more) types and hence inherits the functions of both supertypes. This leads to the creation of a type lattice. In several cases if the lattice grows integrity and ambiguity problems may arise.

In other cases it is desirable that a type inherits only part of structure and/or behavior of the a supertype. This is the case of selective inheritance. The EXCEPT clause may be used to list the functions in a supertype that are not to be inherited by the subtype.

#### d. Polymorphism

The term polymorphism integrates the various cases where, during inheritance, redefinition of structure or behavior is necessary. Polymorphism implies operator overloading where the same operator name or symbol is bound to two or more different implementations of the operator, depending on the type of objects to which the operator is applied. For instance the function Area should be overloaded by different implementations in the `Geometry_Object` example.

The process of selecting the appropriate method based on an object's type is called *binding*. In strongly typed systems, this can be done at compile time. This is termed early (static) binding.

On the other hand binding can take place at runtime, in this case called dynamic binding. An example follows:

```
template <type T>
  T max(x:T, y:T) {
    if (x > y) return x;
    else  return y;
  }
```

The actual methods are instantiated as follows:

```
int max(int, int);
real max(real, real);
```

If the determination of an object's type can be deferred until runtime (rather than compile time), the selection is called dynamic binding.

#### e. Complex Objects

A complex object, an important concept in the object-oriented approach, is an item that is perceived as a single object in the real world, but combines with other objects in a set of complex a-part-of relationships.

We distinguish two categories of complex objects: unstructured and structured ones. The *unstructured complex objects* contain bitmap images, long text strings; they are known as binary large objects (BLOBs), OODBMSs provide the capability to directly process selection conditions and other operations based on values of these objects.

Such objects are defined by new abstract data types and the user provides the pattern recognition program to map the object attributes to the raw data.

The second category, *structured complex objects,* implies that the object's structure is defined by repeated application of the type constructors provided by the OODBMS. Here we distinguish two types of reference semantics: (1) ownership semantics (is-part-of; is-component-of), and (2) is-associated-with relationship. The idea is that OODBMSs constitute a "marriage" between the concepts of object-oriented programming (OOP), such as inheritance, encapsulation, and polymorphism, and well-founded and industry level supported database capabilities (see Fig. 1).

Conclusively, the ideas that were worked out in the OODBMS area have been concentrated and codified by the Object-Oriented Database System Manifesto. This is an effort toward standardization and promoting ideas that should be part of an object-oriented database system. The most important concepts from this manifesto follow:

- An OODBMS should support complex objects.
- Object identity is an important part of an object and must be supported.
- Encapsulation must be supported as an integral part of the system.
- Types or classes must be supported along with inheritance features, so that types or classes must be able to inherit from their ancestors.
- Dynamic binding must be supported.
- The DML must be computationally complete and the set of data types must be extensible. Moreover

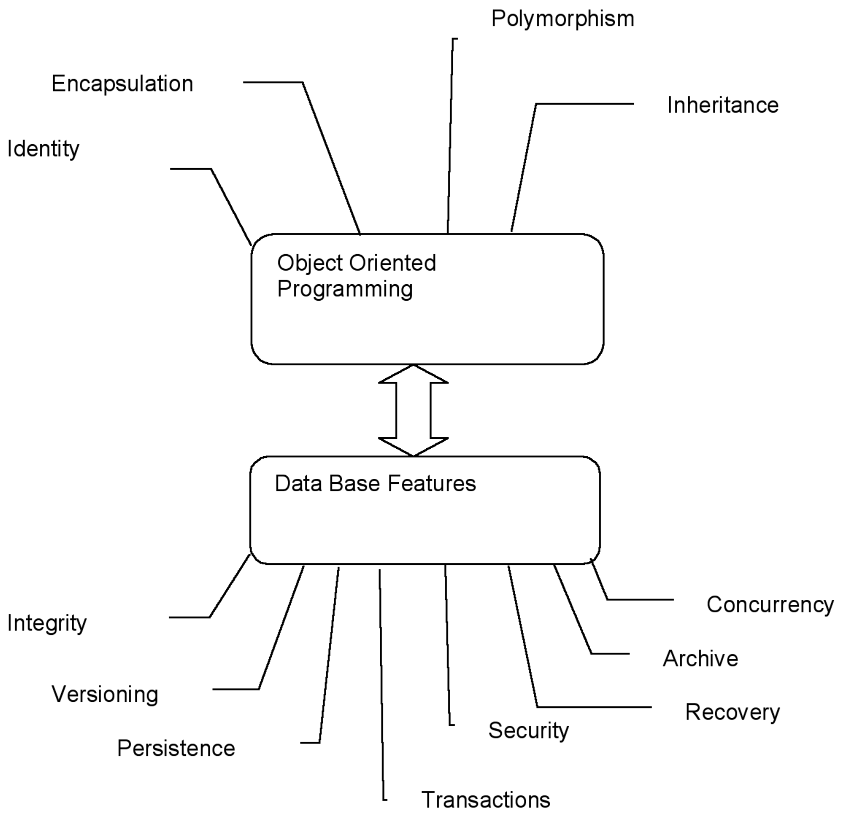


**Figure 1** OODBMS is an amalgamation between OOP and traditional database concepts.

The DBMS must provide a simple way of querying data.

- Data persistence must be provided as a natural feature of a database system.
- The DBMS must be capable of managing very large databases.
- The DBMS must support concurrent users.
- The DBMS must be capable of recovery from hardware and software failures.

Though these features are quite interesting and promising, the impact on the database industry was not as important as hoped. Instead the big database industry tried to extend the well-established relational technology by integrating some of the object-oriented database concepts. This hybrid technology is known as object relational. In the next section there is a review of the main features of this technology.

## C. The Object Relational Approach

Here we contrast and compare the relative strengths and weaknesses of the relational and object-oriented systems in order to motivate the need for object-oriented modeling. The discussion will include user-defined data types and set-based versus navigational access to data. We also examine some simple modeling examples to illustrate the discussions.

### 1. A Quick Look at Relational and Object-Oriented Databases

It is evident that the strengths of the relational paradigm have revolutionized information technology. Relational database technology was originally described by E. F. Codd. Not long afterward, companies like IBM and Oracle created very successful products. The relational DB standard is published by ANSI, with the current specification being X3H2 (SQL'92). The new specification dealing with object extensibility has been labeled X3H7. A relational DB stores data in one or more tables or rows and columns. The rows correspond to a record (tuple); the columns correspond to attributes (fields in the record), with each column having a data type like date, character, or number. Commercial implementations currently support very few data types. For example, character, string, time, date, numbers (fixed and floating point), and currency describe the various options. Any attribute (field) of a record can store only a single value.

Relational DBs enforce data integrity via relational operations, and the data themselves are structures to a simple model based on mathematical set theory. Relationships are not explicit but rather implied by values in specific fields, for example, foreign keys in one table that match those of records in a second table. Many-to-many relationships typically require an intermediate table that contains just the relationships. Relational DBs offer simplicity in modifying table structure. For example, adding data columns to existing tables or introducing entire tables remains an extremely simple operation. The beauty of relational DBs continues to be in its simplicity. The process of normalization establishes a succinct clarity to the management and organization of data in the DB. Redundancies are eliminated and information retrieval its governed by the associations created between primary and foreign keys. Why store the same piece of information in two or more places when a logical connection can be established to it in one place? Referential integrity (RI) has also made an important contribution because it enables business rules to be controlled through the use of constraints. The role of constraints is to prevent the violation of data integrity and, thereby, its normalization.

The origins of OODBs trace their beginnings to the emergence of OOP in the 1970s. Technically, there is no official standard for object DBs. The book *The Object Database Standard: ODMG-V2.0,* under the sponsorship of the Object Database Management Group (ODMG) (http://www.odmg.com), describes an industry-accepted de facto standard. Object DBMSs emphasize objects, their relationships, and the storage of those objects in the DB.

Designers of complex systems realized the limitations of the relational paradigm when trying to model complex systems. Characteristics of object DBs include a data model that has object-oriented aspects like class, with attributes, methods, and integrity constraints; they also have OIDs for any persistent instantiation of classes; they support encapsulation (data and methods), multiple inheritance, and abstract data types.

Object-oriented data types can be extended to support complex data such as multimedia by defining new object classes that have operations to support the new kinds of information.

The object-oriented modeling paradigm also supports inheritance, which allows incremental development of solutions to complex problems by defining new objects in terms of previously defined objects. Polymorphism allows developers to define operations for one object and then share the specification of the operation with other objects. Objects incorporating polymorphism also have the capability of extending behaviors or operations to include specialized actions or behaviors unique to a particular object. Dynamic binding is used to determine at runtime which operations are actually executed and which are not. Object DBs extend the functionality of object programming languages like C++ or Java to provide full-featured DB programming capability. The result is a high level of congruence between the data model for the application and the data model of the DB, resulting in less code, more natural data structures, and better maintainability and greater reusability of code. All of those capabilities deliver significant productivity advantages to DB application developers that differ significantly from what is possible in the relational model.

## 2. Contrasting the Major Features of Pure Relational and Object-Oriented Databases

In the relational DB, the query language is the means to create, access, and update objects. In an object DB, the primary interface for creating and modifying objects is directly via the object language (C++, Java, SMALLTALK) using the native language syntax even though declarative queries are still possible. Additionally, every object in the system is automatically given an OID that is unique and immutable during the object's life. One object can contain an OID that logically references, or points to, another object. Those references prove valuable when in the association of objects with real-world entities, such as products, customers, or business processes; they also form the basis of features such as bidirectional relationships, versioning, composite objects, and distribution. In most ODBMSs, the OIDs become physical (the logical identifier is converted to pointers to specific memory addresses) once the data are loaded into memory (cached) for use by the object-oriented application. No such construct exists in the relational DB. In fact, the addition of navigational access violates the very principles of normalization because OIDs make no reliance on keys.

To further explore the divergent nature of relational and object-oriented DBs, let us look more closely at the drawbacks of each. Our discussion eventually leads us to the justification behind the object-relational paradigm.

In the following we explore the specifics of what it takes to define the object-relational paradigm. A first issue is to enable object functionality in the relational world. Two important aspects must be considered in any definition of the object-relational paradigm. The first is the *logical design aspects* of the architecture. What data types will be supported? How will data be accessed? The other aspect is the mapping of the logical architecture to a physical implementation.

From a technological standpoint, certain capabilities must be included in the list of logical capabilities, or the DB will not qualify to the minimal requirements for being object-relational. Such capabilities are behavior, collection types, encapsulation, inheritance, and polymorphism.

#### a. Behavior

A method, in the purely object-oriented paradigm, is the incorporation of a specific behavior assigned to an object or element. A method is a function of a particular class.

#### b. Collection Types

An aggregate object is essentially a data-type definition that can be composed of many subtypes coupled with behavior. In Oracle 8, for example, there are two collection types: VARRAYs and nested tables. VARRAYs are suitable when the subset of information is static and the subset is small. A suitable implementation of a VARRAY might be in the same context where a reference entity might be used. The contents of reference entities remain relatively static and serve to validate entries in the referencing table. For example, a reference entity called MARKETS can be created to store the valid set of areas where a company does business. In the same way, a VARRAY might be substituted to perform the same reference and validation. VARRAY constructs are stored inline. That means the VARRAY structure and data are stores in the same data block as the rest of the row as a RAW data type. Although they bear some similarity to PL/SQL tables, VARRAYSs are fixed size.

#### c. Encapsulation

Encapsulation is the definition of a class with data members and functions. In other words, it is the mechanism that binds code and data together while protecting or hiding the encapsulation from outside of the class. The actual implementation is hidden from the user, who only sees the interface.

As an illustrative example, think of a plane engine. You can open it and see that it is there, and the plane pilot can start the ignition. The engine causes the plane to move. Although you can see the motor, the inner functions are hidden from your view. You can appreciate the function that the motor performs without ever knowing all the details of what occurs inside or even how.

#### d. Inheritance

Inheritance is the ability of one class to inherit the structure and behavior of its ancestor. Inheritance allows an object to inherit a certain set of attributes from another object while allowing the addition of specific features.

#### e. Polymorphism

Polymorphism is the ability of different objects in a class hierarchy to have different behaviors in response to the same message. Polymorphism derives its meaning from the Greek for "many forms." A single behavior can generate entirely different responses from objects in the same group. Within the framework of the program, the internal mechanism determines specific names for different purposes and is known as function overloading.

If we consider the perspective of moving OODBs closer to the middle, we discover the following points.

- Object-relational DBs require a generalized object-oriented programming language interface versus a specific, hard-coded one. Normally, object-oriented DBs are geared for a specific programming language.
- Object-oriented DB architectures have been known historically for their slow performance.
- Object-oriented DBs are, by design, limited in terms of scalability.
- Object-oriented DBs are not designed for high concurrency.

## III. INDUSTRIAL SYSTEMS—STANDARDS

### A. Standards—The Object Model of the ODMG

ODMG was created at the initiative of a set of object database vendors with the goal of defining and promoting a portability standard for OODBs. It had eight voting members (O2 Technology, Versant, Poet, Gemstone, Objectivity, Object Design, Uni SQL, IBEX), reviewing members (among which, CERN, Hewlett Packard, Microsoft, Mitre, etc.), and academic members (D. DeWitt, D. Maier, S. Zdonic, M. Carey, E. Moss, and M. Solomon).

It provides the potential advantages of having standards: (1) portability, (2) interoperability, (3) compares commercial products easily and consists of the modules:

1. *Object model* It describes the specific object model supported by the ODMG system, being an extension of the OMG (Object Management Group) model.
2. *Object Definition Language (ODL)* It is used to specify the schema of an object database having a specific schema with a C++ binding.
3. *Object query Language (OQL)* An SQL like language that can be used as a stand-alone language for interactive queries or embedded in a programming language.

In the following we provide an overview of the ODMG model. ODMG addresses objects and literals. An object has both an OID and a state where a literal has only a value but no OID. An object is described by four characteristics: identifier, name, lifetime, and structure. On the other hand, three types of literals are recognized: atomic, collection, and structured. The notation of ODMG uses the keyword "interface" in the place of the keywords "type" and "class." Below is an example:

```
interface Object {
  ...
  boolean    same_as (...)
  Object     copy ( );
  void       deleted ( );
}
Interface Collection: Object {
  ...
  boolean is_empty( );
  ...
};

o.same_as (p)
q = o.copy()
```

## 1. Built-In Interfaces for Collection Objects

Any collection object inherits the basic Collection interface. Given a collection object o, the `o.cardinality()` operation returns the elements in the collection. `O.insert_element(e)` and `o.remove_element(e)` insert or remove an element from the collection O. The ODMG object model uses exceptions for reporting errors or particular conditions. For example, the `ElementNotfound` exception in the Collection interface would be raised by the `o.remove_element(e)` operation if e is not an element in the collection o. Collection objects are further specialized into Set, List, Array, and Dictionary.

## 2. Atomic (User-Defined) Objects

In the ODMG model, any user-defined object that is not a collection object is called an atomic object. They are specified using the keyword class in ODL. The keyword *Struct* corresponds to the tuple constructor. A relationship is a property that specifies that two objects in the database are related together. For example, `work_for` relationship of Employee and `has_emps relationship` of Department, Interfaces, Classes, and Inheritance.

An interface is a specification of the abstract behavior of an object type, which specifies the operation signatures. An interface is noninstantiable—that is, one cannot create objects that correspond to an interface definition.

A class is a specification of both the abstract behavior and abstract state of an object type, and is instantiable.

Another object-oriented feature supported by ODMG is inheritance, which is implemented by *extends* keyword.

The database designer can declare an extent for any object type that is defined via a class declaration and it contains all persistent objects of that class. Extents are also used to automatically enforce the set/subset relationship between the extents of a supertype and its subtype. A key consists of one or more properties (attributes or relationships) whose values are constrained to be unique for each object in the extent.

## 3. The ODL

The ODL's main use is to create object specifications—that is, *classes* and *interfaces*. The user can specify a database schema in ODL independently of any programming language. Moreover it is feasible to use the specific language bindings to specify how ODL constructs that can be mapped to constructs in specific programming languages, such as C++, Java, SMALLTALK. There may be several possible mappings from an object schema diagram (or extended entity relational schema diagram) into ODL classes.

The mapping method follows the following steps: (1) the entity types are mapped into ODL classes, and (2) inheritance is done using extends. Here we have to stress that there is no direct way to support multiple inheritance. Below is an example:

```
class Person extent persons
{
  attribute struct Pname
  ...
};

Class Faculty extends Person ( extent
  faculty )
{
  attribute string rank;
  ....
};
```

The class Faculty extends the class Person which is in turn an extension of class person. Therefore the leaf class Faculty contains all the attributes of all the ancestor classes (person and Person).

In the following example we have two classes (Rectangle, Circle) emanating from the same class `GeometryObject`.

```
Interface GeometryObject
{
  attribute ...
  ...
};

Class Rectangle : GeometryObject
{
attribute ...
...
};
Class Circle : GeometryObject
{
attribute ...
...
};
```

## 4. The OQL

The OQL is the query language proposed for the ODMG object model in order to be able to query OODBs. It is designed to work closely with the programming languages for which an ODMG binding is defined, such as C++, SMALLTALK, and Java. It is similar to SQL (the standard query language for relational databases) and the main difference is the handling of path expressions related to the class framework used. For example:

```
SELECT d.name
FROM d in departments
WHERE d.college = 'Engineering'
```

This query results in the set of the names of the departments on the Engineering college. The type of the result is a bag of strings (i.e., duplicates allowed). The query results have to involve in many cases path expressions (i.e., the path from the root parent class to the class required via inheritance links). The following example illustrates the use of path expressions in OQL queries:

```
departments;
csdepartment;
csdepartment.chair;
csdepartment.has_faculty;
Csdepartment.has_faculty.rank ( x )

SELECT f.rank
FROM f in csdepartment.has_faculty;
```

Then we can write the following query to retrieve the grade point average of all senior students majoring in computer science, with the result ordered by gpa, and within that by last and first name:

```
SELECT     struct (last_name:
             s.name.lname, first_name:
           s.name.fname, gpa: s.gpa)
FROM       s in csdepartment.
             has_majors
WHERE      s.class = 'senior'
ORDER BY   gpa DESC, last_name ASC,
             first_name ASC;
```

Another facility provided is the specification of views as named queries. For instance we can define the following view:

```
DEFINE has_minors(deptname) AS
SELECT s
FROM s in students
WHERE s.minors_in.dname = deptname;
```

Then a potential query searching for the students that took as minor Computer Science the following query can be formed:

```
Has_minors('Computer Science');
```

QOL also provides *aggregate functions,* and *quantifiers.* The following example query returns the number of students that took as minor Computer Science:

```
count (s in has_minors(‘Computer
  Science’));
avg (SELECT s.gpa
  FROM s in students
  WHERE s.major_in.dname = ‘Computer
    Science’ and s.class = ‘senior’);
```

The following query illustrates the use of the “forall” predicate. Are all computer science graduate students advised by computer science faculty?

```
For all g in (
  SELECT s
  FROM s in grad_students
  WHERE s.majors_in.dname =
    ‘Computer Science’)
: g.advisor in csdepartment.
    has_faculty;
```

The following query illustrates the use of the “exists” predicate. The query searches for any graduate computer science major having a 4.0 gpa

```
exists g in (
  SELECT s
  FROM s in grad_students
  WHERE s. majors_in.dname =
    ‘Computer Science’
  AND g.gpa = 4;
```

## 5. Object Database Conceptual Design

An important issue that arises here is the conceptual design of an object-oriented schema. Can we benefit from the traditional design techniques applied in relational databases? There are important differences between conceptual design of object databases and relational databases.

As regards relationships, object databases (ODB) exploit object identifiers resulting in OID references while a relational database system (RDB) references to tuples by values or by externally specified/generated foreign keys.

Regarding inheritance, the ODB approach exploits inheritance constructs such as derived (:) and EXTENDS, and this is a fundamental feature of the approach, whereas RDBs do not offer any built in support for inheritance. Of course object relational systems and extended RDB systems are adding inheritance constructs.

An important issue for compatibility and reusability of data is the mapping of an extended entity relationship (EER) schema to an ODB schema. The following steps have to be followed:

*Step 1:* Create an ODL *class* for each EER entity type or subclass

*Step 2:* Add relationship properties or reference attributes for each binary relationship into the ODL classes that participate in the relationship

*Step 3:* Include appropriate operations for each class

*Step 4:* An ODL class that corresponds to a subclass in the EER schema inherits the type and methods of its superclass in the ODL schema

*Step 5:* Weak entity types can be mapped in the same way as regular types

*Step 6:* Declare a class to represent the category and define 1:1 relationships between the category and each of its superclasses

*Step 7:* An n-ary relationship with degree n > 2 can be mapped into a separate class, with appropriate references to each participating class.

## B. OO Systems: O2, Object Store, etc.

In this section we refer briefly to existing object-oriented and object-relational database industrial approaches.

### 1. Example of ODBMS—O2 System

This system has historical importance as it was the only European DBMS and one of the few OODBMSs. Its architecture is based on a kernel, called O2Engine, and is responsible for much of the ODBMS functionality. The implementation of O2Engine at the system level is based on a client/server architecture.

At the functional level, three modules (storage component, object manager, schema manager) implement the functionality of the DBMS.

Data definition in O2 is carried out using programming languages such as C++ or Java. Data manipulation in O2 is carried out in several ways. O2 supports OQL as both an ad hoc interactive query language and as an embedded function in a language. There are two alternative ways for using OQL queries. For example:

```
Q1:
d_Bag<d_Ref<Department>>
  engineering_depts;
departments ->query(engineering_depts,
  ‘’this.college = “Engineering’’ ‘’);

Q2:
d_Bag<d_String>
  engineering_dept_names;
```

```
d_oql_Query q0(‘’select d.dname from
  d in departments where
  d.college = “Engineering” “);
d_oql_execute
  (q0, engineering_dept_names);
```

Both queries (Q1and Q2) search for the names of the departments of the college of Engineering.

## 2. Object Relational Systems—Complex Types and Object Orientation

Object-relational DBs are the evolution of pure object-oriented and relational DBs. The convergence of those two disparate approaches came about as a realization that there were inherent shortcomings in the existing paradigms when considered individually. Observers of information technology can still see the debate that rages in the software community over the ultimate character of object-relational DBs or object-relational DBMSs (ORDBMS).

Industry experts are frequently critical of object-relational DBs because they usually demonstrate a limited or nonexistent ability to perform certain relational or object-oriented tasks in comparison to their pure counterparts. A case in point would be the limited support for inheritance in object-relational DBs, a feature fully supported in the object-oriented paradigm. Others argue that inheritance is of such limited consequence when employed in the storage of data and datacentric objects that its pursuit is a waste of effort. Each point of view is almost always driven by the particular background of the individual presenting the criticism. Because the object-relational paradigm is a compromise of two very different architectures, the most effective definition of its ultimate character will be devised by those who have an unbiased appreciation for relational and object-oriented systems alike.

An issue to be tackled can be the question whether ORDBMSs should begin with a relational foundation with added object-orientation of the reverse. From a conceptual level relational DBs have been far more successful than their OODBMS counterparts. It should not be any surprise then, that virtually all major vendors are approaching the object-relational arena by extending the functionality of existing relational DB engines. A case in point would be IBM, Oracle, and Informix, whose efforts to create a “universal server” began by extending their core relational engines.

Anything that causes such controversy should be worth the effort, which begs two important questions: First, what factors have led to the development of object-relational DBs? And second, what characterizes an accurate definition of an object-relational DB? The general answer to the first question is that developers need a more robust means of dealing with complex data elements without sacrificing the access speed for which relational DBs have become known. To answer the second question, there are several characteristics that, as a minimum, must be included to achieve a true object-relational structure. Those characteristics are the following:

- Retrieval mechanism, that is, a query language like SQL but one adapted to the extended features of the ORDBMS, the retrieval mechanism must include not only relational navigation but also object-oriented navigational support
- Support for relational features like keys, constraints, indexes, and so on
- Support for referential integrity as it is currently supported by the relational paradigm
- Support for the object metamodel (classes, types, methods, encapsulation, etc)
- The ability to support user-defined data types
- Support for the SQL3 ANSI standard

A well known extension requirement regards spatial information. Oracle, Informix, and IBM DB2 offer capabilities for user-defined data types and functions that are applied to them. All of them propose the nested storage model, where a complex attribute is stored in an attribute (they call it in-line). They claim it is more efficient when queries regarding an object reposed since self-join is avoided. This is a good argument for choosing this approach.

DB2 Spatial extender provides a comprehensive set of spatial predicates, including comparison functions (contains, cross, disjoint, equals, intersects etc.), relationship functions (common point, embedded point, line cross, area intersect, interior intersect, etc.), combination functions (difference, symmetric difference, intersection, overlay, union, etc.), calculation functions (area, boundary, centroid, distance, end point, length, minimum distance, etc.) and transformation functions (buffer, locatealong, locate between, convexhull, etc.).

The Informix Dynamic Server with Universal Data Option offers type extensibility. So-called DataBlade modules may be used with the system, thus offering new types and associated functions that may be used in columns of database tables. The Informix Geodetic DataBlade Module offers types for time instants and intervals as well as spatial types for points, line segments, strings, rings, polygons, boxes, circles, ellipses, and coordinate pairs.

Since 1996, the Oracle DBMS has offered a so-called spatial data option, also termed "spatial cartridge," that allows the user to better manage spatial data. Current support encompasses geometric forms such as points and point clusters, lines and line strings, and polygons and complex polygons with holes.

## IV. CONCLUSION—RESEARCH ISSUES AND PERSPECTIVES

The presence of rich voluminous and complex data sets and related application domains rises requirements for object oriented features in database support. Significant research has been devoted in this area of integrating such object-oriented features in database technology. Important effort has been committed to either as pure OODBMS or as extensions of the relational approach (ORDBMSs).

The object-oriented model offers the advantageous features regarding:

- Flexibility to handle requirements of new database applications
- Complex objects and operations specification capabilities
- OODBs are designed so they can be directly—or seamlessly—integrated with software that is developed using OOPLs

Nevertheless, there are several interesting issues for further research in the context of object-oriented and object relations database systems. Some of them are:

- View definition and management on the ODB/ORDB schema
- Querying the schema, in terms of class names, attributes, and behavior features
- Optimization in several levels such as path expression queries, storage, and retrieval issues
- Access mechanisms, indexing, and hashing techniques specialized for object-oriented and object relations databases
- Dynamic issues such as schema evolution (class removal or moving in the inheritance tree and related instances management)

Last but not least one should take into account the requirements and the tremendous potential of the World Wide Web content viewed as a loosely structured database of complex objects. There the applicability of the object-oriented approach both for modeling and storage/retrieval is very promising.

## V. CASE STUDY: OBJECT RELATIONAL SOLUTIONS FOR MULTIMEDIA DATABASES

Hereafter we will present the soulutions provided by different ORDMS for video storage and retrieval. IBM's DB2 system supports video retrieval via "video extenders." Video extenders allow for the import of video clips and querying these clips based on attributes such as the format, name/number, or description of the video as well as last modification time.

Oracle (v.8) introduced integrated support for a variety of multimedia content (Oracle Integrated Multimedia Support). This set of services includes text, image, audio, video, and spatial information as native data types, together with a suite of data cartridges that provide functionality to store, manage, search, and efficiently retrieve multimedia content from the server. Oracle 8i has extended this support with significant innovations, including its ability to support cross-domain applications that combine searches of a number of kinds of multimedia forms and native support for data in a variety of standard Internet formats, including JPEG, MPEG, GIF, and the like. Oracle has packaged its complex datatype support features together with management and access facilities into a product called Oracle 8i interMedia. This product enables Oracle 8i to manage complex data in an integrated fashion with other enterprise data, and permits transparent access to such data through standard SQL using appropriate operators. It also includes Internet support for popular web-authoring tools and web servers. It offers online Internet-based geocoding services for locator applications, and powerful text search features.

Informix's multimedia asset management technology offers a range of solutions for media or publishing organizations. In fact, Informix's database technology is already running at the core of innovative multimedia solutions in use. Informix Dynamic Server with Universal Data Option enables effective, efficient management of all types of multimedia content—including images, sound, video, electronic documents, web pages, and more. The Universal Data Option enables query, access, search, and archive digital assets based on the content itself. Informix's database technology provides: cataloging, retrieval, and reuse of rich and complex media types like video, audio, images, time series, text, and more—enabling viewer ac-

**Table I** Comparative Presentation of Multimedia Retrieval Capabilities of Commercial Object Relational Systems

| | QBIC | Oracle | Informix | DB2 |
|---|---|---|---|---|
| Color | Percentage, layout (hist) | Visual Info Retrieval<br>Global, Local color | Excalibur (Image Dblade) | |
| Texture | Similar | Graininess, smoothness | Excalibur (Image Dblade) | |
| Shape | | | Excalibur (Image Dblade) | |
| Spatial relationships | Show position | | | |
| Scene detection | | | MEDIAstra (Video Dblade) | |
| Object detection | | | MEDIAstra (Video Dblade)<br>MPEG-4 approach | |
| Captions & Annotations | Manual annotation | | DbFlix—meta-data storage<br>Time, frame, content based approach. | Description (img)<br>Format, frame rate, tracks (video)<br>Format, last update (audio) |
| Extend functionality | | | Datablades | DB2 Extenders |
| Sound | | | Muscle Fish Audio (content based queries) | Limited |
| Other | Feature layout | Ideal for Video on Demand | Feature vector (Exc)<br>Video reproduction (Media) | Feature layout<br>Voice to text |

cess to audio, video, and print news sources; high-performance connectivity between your database and Web servers providing on-line users with access to up-to-the-minute information; tight integration between your database and web development environments for rapid application development and deployment; and extensibility for adding features like custom news and information profiles for viewers (Table I).

## SEE ALSO THE FOLLOWING ARTICLES

Cohesion, Coupling, and Abstraction • Database Administration • Data Modeling: Entity-Relationship Data Model • Multimedia • Object-Oriented Programming • Relational Database Systems

## BIBLIOGRAPHY

Atkinson, M., Bancillon, F., DeWitt, D., Dittrich, K., Maier, D., and Zdonik, S. (1992). The Object-Oriented Database System Manifesto. In *Building an Object-Oriented Database System: The Story of O2,* Bancillon et al. (eds), San Francisco: Morgan Kaufmann. Also in *Proc. Int. Conf. Deductive Object-Oriented Databases,* Kyoto, Japan, Dec. 1989.

Bancillon, F., Briggs, T., Khoshafian, S., Valduriez, P. (1987). FAD-a Simple and Powerful Database Language, *Proceedings of VLDB 1987.*

Chou, Ç., *et al.* (1985). Design and implementation of the Wisconsin storage system, *Software-Practice and Experience,* 15(10), Oct. 1985.

Codd, E. F. (1970). A Relational Model for Large Shared Data Banks, *Communications of the ACM,* 13, 377–387.

Copeland, G., and Khoshafian, S. (1986). Identity and Versions for Complex Objects, *Proceedings of the International Workshop on Object-Oriented Database Systems,* September, Pacific Grove, CA.

Dahl, O. J., and Nygaard, K. (1966). SIMULA—An ALGOL-based simulation language. *Communications of the ACM,* 9(9):671–678.

Date, C. J., and Darwen, J. (1996). *Foundation for object/relational databases, the third manifesto.* Reading, MA: Addison-Wesley.

Davis, J. R. (1998). IBM's DB2 Spatial Extender: Managing Geo-Spatial Information Within the DBMS. Technical report. IBM Corporation.

Deux, Ï., *et al.* (1990). The story of O2, *IEEE Transactions on Knowledge and Data Engineering,* 2(1), March 1990.

Dittrich, K. (1986). Object-Oriented Database Systems: The Notion and the Issues. *Proceedings of the International Workshop on Object-Oriented Database Systems,* September, Pacific Grove, CA.

Dittrich, K., and Dayal, U. (eds.). (1986). *Proceedings of the International Workshop on Object-Oriented Database Systems,* September, Pacific Grove, CA.

Ehoshafian, S., (1990). Insight into object-oriented databases. *Information and Software Technology,* 32(4).

Goldberg, A., and Robson, D. (1982). *Smalltalk 80: The language and its implementation.* Reading, MA: Addison-Wesley. http://www.software.ibm.com/data/dbs/extenders.

Ìaier, D. (1986). Why Object-Oriented Databases Can Succeed Where Others Have Failed, *Proceedings of the International Workshop on Object-Oriented Database Systems,* September, Pacific Grove, CA.

Informix DataBlade Technology (1999). Transforming Data into Smart Data.

Khoshafian, S. (1993). *Object Oriented Databases.* New York: John Wiley.

Kim, W. (1991). *Introduction to Object-Oriented Database.* Cambridge. MA: The MIT Press.

Lynbaek, P., and Kent, W. (1986). A Data Modeling Methodology for the Design and Implementation of Information Systems, *Proceedings of the International Workshop on Object-Oriented Database Systems,* September, Pacific Grove, CA.

Maier, D., Stein, J. (1986). Indexing in an Object-Oriented DBMS, *Proceedings of the International Workshop on Object-Oriented Database Systems,* September, Pacific Grove, CA.

Maiers, R. (1990). Making Database Systems Fast Enough for CAD, In *Object-Oriented Concepts, Databases and Applications,* (Kim, W., and Lochovsky, F., eds.). Reading, MA: Addison-Wesley Publishing.

Manola, F. (1989). *An Evaluation of Object-Oriented DBMS Developments,* GTE Laboratories Incorporated, TR-0066-10-89-165.

Oracle 8 SQL Type Data Definition Language. (1997). An Oracle Technical White Paper.

Piatini, M., and Diaz, O., eds. (2000). *Advanced database technology and design.* Norwood, MA: Artech House.

Stonebraker, M., Rowe, L., Lindsay, B., Gray, J., and Carey, M. (1990). Third-Generation Data Base System Manifesto. Memorandum No. UCB/ELR. M90/23, April. The Committee for Advanced DBMS Function, University of California, Berkeley, CA.

Tsichritzis, D., and Lochovsky, F. (1982). *Data models.* New York: Prentice Hall.

Wilkinson, K., Lyngbaek, P., Hasan, W. (1990). The Iris Architecture and Implementation, *IEEE Transactions on Knowledge and Data Engineering,* 2(1), March 1990.

# Data Warehousing and Data Marts

**Zhengxin Chen**
*University of Nebraska, Omaha*

## GLOSSARY

**data mart** A departmental subset of the warehouse data focusing on selected subjects. A data warehouse could be a union of all the constituent data marts.

**data mining** Also referred to as knowledge discovery in databases, it is the nontrivial extraction of implicit, previously unknown, interesting, and potentially useful information (usually in the form of knowledge patterns or models) from data.

**data warehouse** A subject-oriented, integrated, time-varying, nonvolatile collection of data that is used primarily in organizational decision making. It is a collection of materialized views derived from base relations that may not reside at the warehouse

**metadata** Data about data, or what the warehouse data look like (rather than warehouse data themselves). In a warehousing environment, metadata include semantic metadata, technical metadata, and core warehouse metadata.

**on-line analytical processing (OLAP)** Applications dominated by stylized queries that typically involve group-by and aggregation operators for analysis purpose.

**Web** The World Wide Web is a hypermedia-based system that provides a simple "point and click" method of browsing information on the Internet using hyperlinks.

## I. OVERVIEW

The complexity involved in traditional distributed database systems (DDBS) has stimulated organizations to find alternative ways to achieve decision support. Data warehousing is an emerging approach for effective decision support. According to the popular definition given by the "godfather" of data warehousing technique, William Inmon (1996), a *data warehouse* is a "subject-oriented, integrated, time-varying, nonvolatile collection of data that is used primarily in organizational decision making." Although considered by some business people as a low-key answer for the "failed" DDBS, data warehousing does take advantage of various techniques related to distributed and parallel computing.

Data warehousing provides an effective approach to deal with complex decision support queries over data from multiple sites. The key to this approach is to create a copy (or derivation) of all the data at one location, and to use the copy rather than going to the individual sources. Note that the original data may be on different software platforms or belong to different organizations.

There are several reasons for the massive growth of volumes of data in the data warehouse environment:

1. Data warehouses collect historical data
2. Data warehouses involve the collection of data to satisfy unknown requirements

3. Data warehouses include data at the summary level, but may also include data at the very detailed atomic or granule level
4. Data warehouses contain external data as well (e.g., demographic, psychographics, etc.).

Significant amounts of external data are collected to support a variety of data mining activities for prediction or knowledge discovery. For example, data mining tools would use this external data to predict who is likely to be a good customer or how certain companies are likely to perform in the marketplace.

Data warehouses contain consolidated data from many sources (different business units), spanning long time periods and augmented with summary information. Warehouses are much larger than other kinds of databases, typical workloads involve ad hoc, fairly complex queries, and fast response times are important. Data warehousing encompasses frameworks, architectures, algorithms, tools, and techniques for bringing together selected data from multiple databases or other information sources into a single repository suitable for direct querying or analysis. Data warehousing is especially important in industry today because of a need for enterprises to gather all of their information into a single place for in-depth analysis, and the desire to decouple such analysis from their on-line transaction processing (OLTP) systems. Since decision support often is the goal of data warehousing, clearly warehouses may be tuned for decision support, and perhaps vice versa.

In its simplest form, data warehousing can be considered as an example of asynchronous replication, in which copies are updated relatively infrequently. However, a more advanced implementation of data warehousing would store summary data or other kinds of information derived from the source data. In other words, a data warehouse stores materialized views (plus some local relations if needed).

It is common in a data warehousing environment for source changes to be deferred and applied to the warehouse views in large batches for efficiency. Source changes received during the day are applied to the views in a nightly batch window (the warehouse is not available to the users during this period). Most current commercial warehousing systems focus on storing the data for efficient access, and on providing extensive querying facilities at the warehouse. Maintenance of warehousing data (in a large degree, maintenance of materialized views) is thus an important problem.

The need for data warehousing techniques is justified largely due to *decision support queries,* which are ad hoc user queries in various business applications. In these applications, current and historical data are comprehensively analyzed and explored, identifying useful trends and creating summaries of the data, in order to support high-level decision making in data warehousing environment. A class of stylized queries typically involve group-by and aggregation operators. Applications dominated by such queries are referred to as *on-line analysis processing* (OLAP), which refers to applications dominated by stylized queries that typically involve group-by and aggregation operators for analysis purpose. Such queries are extremely important to organizations to analyze important trends so that better decisions can be made in the future. In addition, most vendors of OLAP engines have focused on Internet-enabling their offerings. The true promise of the Internet is in making OLAP a mainstream technology, that is, moving OLAP from the domain of analysts to consumers. E-commerce has emerged as one of the largest applications of the Internet in decision support. We will revisit issues related to OLAP after we examine basic features of data warehousing.

## II. OPERATIONAL SYSTEMS AND WAREHOUSE DATA

In order to understand the nature of data warehousing, we should first pay attention to the relationship between the parallel universes of operational systems and the warehouse data. *Operational data stores* (ODSs), also called *operational systems,* feature standard, repetitive transactions that use a small amount of data as in traditional database systems, while *warehouse systems* feature ad hoc queries and reports that access a much larger amount of data. High availability and real-time updates are critical to the success of the operational system, but not to the data warehouse. Operational systems are typically developed first by working with users to determine and define business processes. Then application code is written and databases are designed. These systems are defined by how the business operates. The primary goal of an organization's operational systems is efficient performance. The design of both applications and databases is driven by OLTP performance. These systems need the capacity to handle thousands of transactions and return information in acceptable user-response time frames, often measured in fractions of a second. By contrast, data warehouses start with predefined data and vague or unknown usage requirements. Operational systems have different structures, needs, requirements and objectives from data warehouses. These variations in data, access, and usage prevent organizations from simply using existing operational systems as data warehouse resources. For example, operational data is short-lived and changes rapidly, while warehouse data has a long life and is static. There is a need for transformation of operational data to warehouse data. The architecture of a data warehouse differs considerably from the design and structure of an operational system. Designing a data warehouse can be more difficult than building an operational system because the requirements of the warehouse are often ambiguous. In designing a data warehouse, an organization must understand current information needs as well as likely future needs. This requires a flexible design that ensures the data warehouse can adapt to a changing environment.

Both operational and warehouse systems play an important role in an organization's success and need to coexist as valuable assets. This can be done by carefully designing and architecting the data warehouse so that it takes advantage of the power of operational data while also meeting the unique needs of the organization's knowledge workers.

## III. ARCHITECTURE AND DESIGN OF DATA WAREHOUSES

### A. Data Warehouse Components

The data warehouse is an integrated environment, containing integrated data, detailed and summarized data, historical data, and metadata. An important advantage of performing data mining in such an environment is that the data miner can concentrate on mining data, rather than cleansing and integrating data.

Data warehousing provides an effective approach to deal with complex decision support queries over data from multiple sites. According to a popular definition, a data warehouse is a subject-oriented, integrated, time-varying, nonvolatile collection of data that is used primarily in organizational decision making. The key to the data warehousing approach is to create a copy of all the data at one location, and to use the copy rather than going to the individual sources. Data warehouses contain consolidated data from many sources (different business units), spanning long time periods, and augmented with summary information. Warehouses are much larger than other kinds of databases, typical workloads involve ad hoc, fairly complex queries, and fast response times are important. Since decision support often is the goal of data warehousing, clearly warehouses may be tuned for decision support, and perhaps vice versa.

A typical data warehousing architecture consists of the following components (as in Fig. 1):

- A *relational database* for data storage. As the data warehouse proper, it stores the corporate data. Here data volumes are very high as multi-terabyte data warehouses are beginning to appear more frequently.
- *Data marts.* Departmental subsets of the warehouse data focusing on selected subjects. The data mart is where departmental data is stored, and often various external data items are added. The data volumes are usually 15–30% of warehouse sizes, and the envelope is being pushed toward the terabyte limit. These databases are also usually either based on star schemas or are in a normalized form. They mostly deal with the data space, but at times some multidimensional analysis is performed.
- *Back end.* System components providing functionality such as extract, transform, load and refresh data and *front end* such as OLAP and data mining tools and utilities.
- *Metadata.* The system catalogs associated with a warehouse are very large, and are often stored and managed in a separate database called a metadata repository.
- *Other components.* These depend on the design methods and the specific needs of the organizations.

### B. Data Warehouse Design

There are four different views regarding the design of a data warehouse. The top-down view allows the selection of the relevant information necessary for the data warehouse. The data source view exposes the information being captured, stored, and managed by operational systems. The data warehouse view includes fact tables and dimension tables. The business query view is the perspective of data in the data warehouse from the viewpoint of the end user. The process of data warehouse design can take several approaches. The top-down approach starts with the overall design and planning, and is useful in cases when the business problems are clear and well understood. The bottom-up approach starts with experiments and prototypes, and is useful in the early stages of business modeling and technology development. In addition, a combination of both can be used.

In general, data warehouse design process consists of the following steps:

1. Choose a business process to model, such as sales, shipments, etc.
2. Choose the grain of the business process. The grain is the granularity (namely, fundamental, atomic) level of the data used in the fact table.

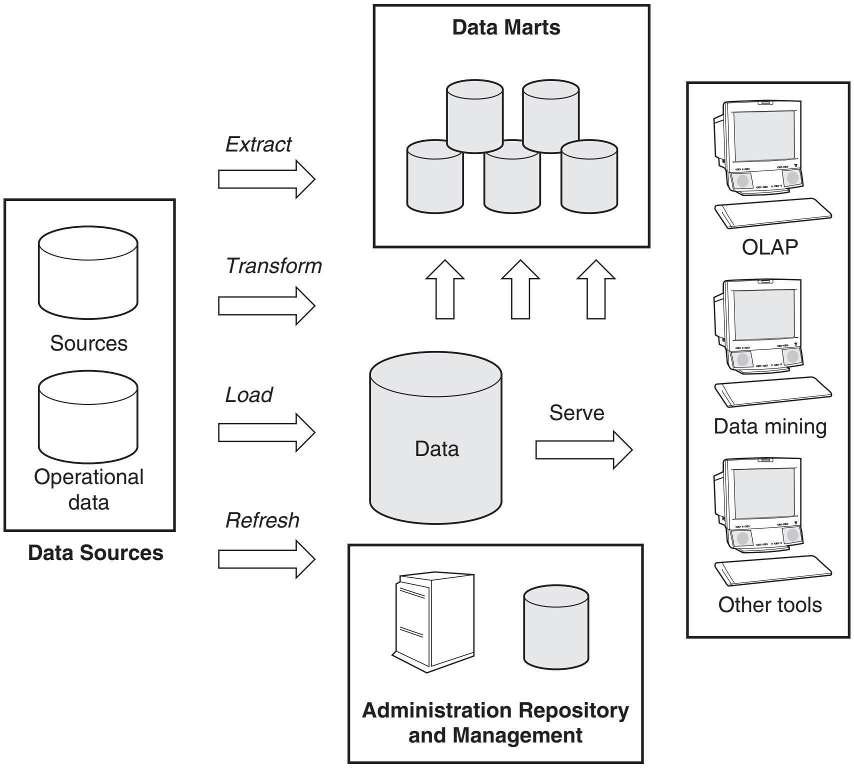


**Figure 1** Data warehouse architecture.

The data stored there are the primary data based on which OLAP operations can be performed.

3. Choose the dimensions that will apply to records in the fact table(s). For example, time is a typical dimension. Dimensions are important and based on which various OLAP operations can be performed.
4. Choose the measures that will populate each fact table record. Measures are numeric additive quantities such as sales amount or profit.

The architecture depicted in Fig. 1 is basically two-tier, namely, warehouse and its front ends. A variation of data warehouse architecture consists of three tiers: the bottom tier is a warehouse database server, which is typically a relational database system; the middle tier is an OLAP server; and the top tier is a client, containing query and reporting tools.

## IV. DATA WAREHOUSES AND MATERIALIZED VIEWS

### A. Materialized Views

At the beginning of this article we provided a brief discussion on data warehouses based on a business perspective. However, this discussion requires some further technical clarification. For example, we said that a data warehouse consists of a copy of data acquired from the source data. What does this copy look like? In fact, we may need to distinguish between a "true" copy (duplicate), a derived copy, approximate duplicate, or something else. For this reason, we need to examine the concept of data warehouse in more depth. In fact, a data warehouse can be characterized using materialized views and indexing. In the following, we will examine these two issues.

According to fundamentals of database management systems (DBMS), relational views are the most important asset of the relational model. Recall that we have the following basic concepts in relational databases:

- *Relation* (base table): It is a stored table.
- *External view* (virtual view, or simply view): It is a virtual table (derived relation defined in terms of base relations).
- *Materialized view:* A view is materialized when it is stored in the database, rather than computed from the base relations in response to queries.

The general idea of the approach is to materialize certain expensive computations that are frequently inquired, especially those involving aggregate functions, such as count, sum, average, max, etc., and to store such materialized views in a multidimensional database (called a data cube) for decision support, knowledge discovery, and many other applications.

Commercial relational database products are used to discard views immediately after they are delivered to the user or to a subsequent execution phase. The cost for generating the views is for one-time-use only instead of being amortized over multiple and/or shared results. Caching query results or intermediate results for speeding up intra- and interquery processing has been studied widely. All these techniques share one basic idea: the reuse of views to save cost.

The benefit of using materialized views is significant. Index structures can be built on the materialized view. Consequently, database access to the materialized view can be much faster than recomputing the view. A materialized view is just like a cache, which is a copy of the data that can be accessed quickly. Materialized views are useful in new applications such as data warehousing, replication servers, chronicle or data recording systems, data visualization, and mobile systems. Integrity constraint checking and query optimization can also benefit from materialized views, but will not be emphasized in our current context.

We now discuss the issue of what materialized views look like. The traditional relational database design has put emphasis on normalization. However, data warehouse design cannot be simply reduced to relational database design. In fact, frequently materialized views involve join operation, and they are no longer in high normal forms as discussed in DBMS. Although normalized data guarantees integrity constraints and avoiding anomalies, in the business community, it is not uncommon for people to feel that normalized designs are hard to comprehend; denormalized designs tend to be more self-explanatory, even though denormalized tables have longer records. Typical multi-attribute search-and-scan performance is better on denormalized data because fewer tables are involved than in normalized designs. Denormalization data provide an intuitive productive environment for users who need to be trained or retrained. On the other hand, denormalization is the greatest cultural hurdle for most incremental data mart design teams, because they are used to deal with OLTP. Redundancy of data is the result of denormalization. For example, two relations along with the joined result coexist.

Another remark we want to make here is on the impact of ER modeling to data warehouse design. There are two schools of thought in enterprise data warehouse design. The ER normalized school still starts from the fundamentally normalized tables and then spawn off subset data marts that are denormalized. In contrast, Ralph Kimball and his school endorse a consistent, denormalized star schema environment across the entire enterprise data warehouses.

## B. Indexing Techniques

Due to the close relationship between materialized views and indexing, it is worthwhile to take a look at the issue of indexing. Traditional indexing techniques can be used, but there are also additional issues which are unique in a data warehousing environment.

The mostly read environment of OLAP systems makes the CPU overhead of maintaining indices negligible, and the requirement of interactive response times for queries over very large datasets makes the availability of suitable indices very important.

- *Bitmap index.* The idea is to record values for sparse columns as a sequence of bits, one for each possible value. For example, the biological gender of a customer (male or female) can be represented using bitmap index. This method supports efficient index operations such as union and intersection; more efficient than hash index and tree index.
- *Join index.* This method is used to speed up specific join queries. A join index maintains the relationships between a foreign key with its matching primary keys. The specialized nature of star schemas makes join indices especially attractive for decision support.

Indexing is important to materialized views for two reasons: indexes for a materialized view reduce the cost of computation to execute an operation (analogous to the use of an index on the key of a relation to decrease the time needed to locate a specified tuple); and indexing reduces the cost of maintenance of the materialized views. One important problem in data warehousing is the maintenance of materialized views due to changes made in the source data. Maintenance of materialized views can be a very time-consuming process. There need to be some methods developed to reduce this time (one method is use of supporting views and/or the materializing of indexes).

# V. DATA MARTS

## A. Why Data Marts

As an important component of data warehousing architecture, a data mart is a departmental subset on selected subjects. Therefore, a data mart is an application-focused data warehouse, built rapidly to support a single line-of-business application. Data marts still have all of the other characteristics of data warehouses, which are subject-oriented data that is nonvolatile, time-variant, and integrated. However, rather than representing a picture of the enterprise data, it contains a subset of that data which is specifically of interest to one department or division of that enterprise.

The data that resides in the data warehouse is at a very granular level and the data in the data mart is at a refined level. The different data marts contain different combinations and selections of the same detailed data found at the data warehouse. In some cases data warehouse detailed data is added differently across the different data marts, while in other cases a specific data mart may structure detailed data differently from other data marts. In each case the data warehouse provides the granular foundation for all of the data found in all of the data marts. Because of the singular data warehouse foundation that all data marts have, all of the data marts have a common heritage and are able to be reconciled at the most basic level.

There are several factors that lead to the popularity of the data mart. As data warehouses quickly grow large, the motivation for data marts increases. More and more departmental decision-support processing is carried out inside the data warehouse, as a result, resource consumption becomes a real problem. Data becomes harder to customize. As long as the data in the data warehouse are small, the users can afford to customize and summarize data every time a decision support analysis is done. But with the increase in magnitude, the user does not have the time or resources to summarize the data every time a decision support analysis is done. The cost of doing processing in the data warehouse increases as volume of data increases. The software that is available for the access and analysis of large amounts of data is not nearly as elegant as the software process smaller amounts of data. As a result of these factors, data marts have become a natural extension of the data warehouse.

There are organizational, technological, and economic reasons why the data mart is so beguiling and is a natural out of the data warehouse. Data marts are attractive for various reasons:

- *Customization.* When a department has its own data mart, it can customize the data as the data flows into the data mart from the data warehouse. The department can sort, select, and structure their own departmental data without consideration of any other department.
- *Relevance.* The amount of historical data that is needed is a function of the department, not the corporation. In most cases, the department can select a much smaller amount of historical data than that which is found in the data warehouse.
- *Self-determination.* The department can do whatever decision-support processing they want whenever they want, with no impact for resource utilization on other departments. The department can also select software for their data mart that is tailored to fit their needs.
- *Efficiency.* The unit cost of processing and storage on the size of machine that is appropriate to the data mart is significantly less than the unit cost of processing and storage for the facilities that houses the data warehouse.

## B. Types of Data Marts

There are several kinds of data marts strategies, the following are two important types:

1. *Dependent data marts.* The architectural principles evolved into the concept of dependent data marts, which are smaller subsets of the enterprise warehouse specifically designed to respond to departmental or line-of business issues. In this strategy, data is loaded from operational systems into the enterprise warehouse and then subdivided into the smaller data marts. These marts rely on the central warehouse for their data and metadata rather than obtaining these from the operational systems. While these data marts can solve some of the performance issues and even some of the political issues, the financial problems and strategic issues were, if anything, exacerbated because the enterprise warehouse must be built before the data marts can be implemented.
2. *Independent data marts.* Independent data marts have been viewed as a viable alternative to the top-down approach of an enterprise warehouse. An organization can start small and move quickly, often realizing limited results in three to six months. Proponents of this approach argue that after starting with a small data mart, other marts can proliferate in departments or lines of business that have a need. In addition, by satisfying the various divisional needs, an organization can build its way to a full data warehouse by starting at the bottom and working up.

## C. Multiple Data Marts

In today's environment, data marts have evolved to more cost effectively meet the unique needs of business lines. These marts can be built more quickly and contain only the data relevant to the specific business unit. For example, data marts may be divided based on departmental lines or by product type. These marts challenge the systems group in terms of managing the ongoing changes along with ensuring data consistency across the different marts. The architecture must provide for simplifying and reducing the costs in this multiple data mart management process. A data mart can overlap another data mart. Kimball et al. (1998) advised to consider 10 to 30 data marts for a large organization. For an organization, careful studies and plans are needed for developing multiple data marts. A matrix method for identifying all the possible data marts and dimensions was introduced.

Subsets of the data are usually extracted by subject area and stored in data marts specifically designed to provide departmental users with fast access to large amounts of topical data. These data marts often provide several levels of aggregation and employ physical dimensional design techniques for OLAP.

If a corporation begins their information architecture with an enterprise data warehouse, they will quickly realize the need for subsets of the data levels of summary. A recommended approach is through top-down development: we can spawn marts and select subject area and summary from the enterprise data warehouse. The other approach is where a data mart is built first to meet specific user group requirements, but is built according to a data plan and with roll-up as a goal. The need will arise for the marts to participate in a hierarchy in which detailed information from several subject-area data marts is summarized and consolidated into an enterprise data warehouse.

## D. Networked Data Marts

Increasingly, multiple data mart systems cooperate in a larger network creating a virtual data warehouse. This results in *networked data marts*. A large enterprise may have many subject-area marts as well as marts in different divisions and geographic locations. Users or workgroups may have local data marts to address local needs. Advanced applications, such as the Web, extend data mart networks across enterprise boundaries.

In the network data mart world, users must be able to look at and work with multiple warehouses from a single client workstation, requiring location transparency across the network. Similarly, data mart administrators must be able to manage and administer a network of data marts from a single location.

Implementation and management of data mart networks not only imposes new requirements on the mart relational database management system (RDBMS), but more importantly requires tools to define, extract, move, and update batches of information as self-consistent units on demand. It also requires a whole new generation of data warehouse management software to support subset, catalog, schedule, and publish/subscribe functions in a distributed environment.

# VI. METADATA

## A. Basics of Metadata

Data warehousing must not only provide data to knowledge workers, but also deliver information about the data that defines content and context, providing real meaning and value. This information about data is called metadata. The coming of data warehouses and data mining has significantly extended the role of metadata in the classical DBMS environment. Metadata describe the data in the database, they include information on access methods, index strategies, and security and integrity constraints, as well as policies and procedures (optional).

Metadata become a major issue with some of the recent developments in data management such as digital libraries. Metadata in distributed and heterogeneous databases guides the schema transformation and integration process in handling heterogeneity, and are used to transform legacy database systems to new systems. Metadata can be used for multimedia data management (metadata itself could be multimedia data such as video and audio). Metadata for the Web includes information about various data sources, locations, and resources on the Web as well as usage patterns, policies and procedures.

Metadata (such as metadata in repository) can be mined to extract useful information in cases where the data themselves are not analyzable. For example, the data are not complete, or the data are unstructured. The coming of data warehouses and data mining has significantly extended the role of metadata in the classical DBMS environment. Metadata describe the data in the database, they include information on access methods, index strategies, security and integrity constraints, as well as policies and procedures (optional). Every software product involved in loading, accessing, or analyzing the data warehouse requires metadata. In each case, metadata provides the unifying link between the data warehouse or data mart and the application processing layer of the software product.

## B. Metadata for Data Warehousing

Metadata for warehousing include metadata for integrating the heterogeneous data sources. Metadata can guide the transformation process from layer to layer in building the warehouse, and can be used to administer and maintain the warehouse. Metadata is used to extract answers to the various queries posed.

Figure 2 illustrates metadata management in a data warehouse. The metadata repository stores and maintains information about the structure and the content of the data warehouse components. In addition, all dependencies between the different layers of the data warehouse environment, including operational layer, data warehouse layer, and business layer, are represented in this repository.

Figure 2 also shows the role of three different types of metadata:

1. *Semantic (or business) metadata.* These kinds of data intend to provide a business-oriented description of the data warehouse content. A repository addressing semantic metadata should cover the types of metadata of the conceptual enterprise model, multidimensional data model, etc., and their interdependencies.
2. *Technical metadata.* These kinds of data cover information about the architecture and schema with respect to the operational systems, the data warehouse, and the OLAP databases, as well as the dependencies and mappings between the operational sources, the data warehouse, and the OLAP databases on the physical and implementation level.
3. *Core warehouse metadata.* These kinds of data are subject-oriented and are based on abstractions of the real world. They define the way in which the transformed data are to be interpreted, as well as any additional views that may have been created.

A successful data warehouse must be able to deliver both the data and the associated metadata to users. A data warehousing architecture must account for both. Metadata provides a bridge between the parallel universes of operational systems and data warehousing.

The operational systems are the sources of metadata as well as operational data. Metadata is extracted from individual operational systems. This set of metadata forms a model of the operational system. This metadata includes, for example, the entities /records/ tables and associated attributes of the data source.

The metadata from multiple operational data sources is integrated into a single model of the data warehouse. The model provides data warehouse architects with a business model through which they can understand the type of data available in a warehouse, the origin of the data, and the relationships between the data elements. They can also provide more suitable terms for naming data than are usually present in the operational systems. From this business model, physical database design can be engineered and the actual data warehouse can be created.

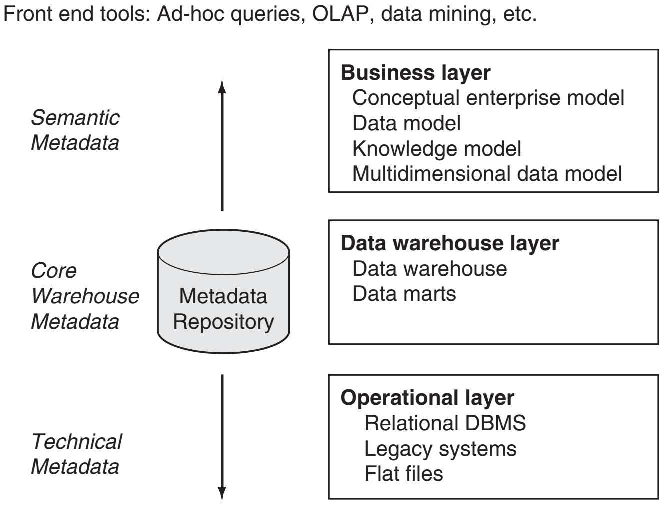


**Figure 2** Metadata management in a data warehouse.

The metadata contained in the data warehouse and data mart models is available to specialized data warehousing tools for use in analyzing the data. In this way the data and metadata can be kept in synchronization as both flow through the data warehouse distribution channel from source to target to consumer.

## C. Metadata in Data Marts

Metadata in the data mart serves the same purpose as metadata in the data warehouse. Data mart metadata allows the data mart decision-support user to find out where data are in the process of discovery and exploration.

Note that types of metadata form a hierarchy: on the topmost are the metadata for the data warehouse, underneath are metadata for mappings and transformations, followed by metadata for various data sources. This observation explains the relationship between metadata and *multitiered data warehouse,* which is built to suit the customers' needs and economics, spanning the spectrum from an enterprise-wide data warehouse to various data marts. Since multitiered data warehouses can encompass the best of both enterprise data warehouses and data marts, they are more than just a solution to a decision support problem. Multitiered implies a hierarchy, with a possible inclusion of a networked data mart layer within the hierarchy. In order to build a hierarchy of data warehouses, a sound data plan must be in place for a strong foundation. It makes no difference whether a corporation starts at the bottom of the hierarchy or the top—they must have a goal in mind and a plan for relating the various levels and networks. The data plan cannot be constructed or managed without an active metadata catalog.

Development of data marts could provide tremendous contribution to the metadata. Along with a robust metadata catalog, a tool that reverse-engineers the various data marts' metadata into a logical unit would be of tremendous value. Reliable algorithms can be used to scan the catalog and group the related items from each data mart, suggesting how they should be combined in a higher level data warehouse.

# VII. DATA WAREHOUSE AND THE WEB

An appropriate platform for building data warehouses and for broader deployment of OLAP is the Internet/intranet/World Wide Web, for various reasons.

- The Web provides complete platform independence and easy maintenance.
- The skills and techniques used to navigate and select information, as well as the browser interface are the same as for all other web-based applications.
- With increased security, the Internet can be used as an inexpensive wide area network (WAN) for decision-support and OLAP applications.

Web-enabled data warehouses deliver the broadest access to decision support. In order to understand how to create a web-based data warehouse architecture for maximum growth and flexibility, we need to take a look at issues related to the Internet, as well as intranets.

## A. The Impact of the Internet

The Internet has opened up tremendous business opportunities needing nontraditional categories of information. The true promise of the Internet is in making OLAP a mainstream technology, that is, moving OLAP from the domain of analysts to consumers. E-commerce has emerged as one of the largest applications of the Internet in decision support. The basic concepts of data warehousing and aggregation have naturally made their way onto the Web. In fact, some of the most popular web sites on the Internet are basically databases. For example, search engines such as Alta Vista and Lycos attempt to warehouse the entire Web. Aggregation as a means to navigate and comprehend the vast amounts of data on the Internet has to also be recognized. Directory services such as Yahoo and Excite attempt to aggregate the entire Web into a category hierarchy and give users the ability to navigate this hierarchy. The infrastructure for decision support is also in the process of improvement.

## B. Intranets

*Intranets* are essentially secure mini-internets implemented within organizations. An intranet can offer a single point of entry into a corporate world of information and applications. Intranets provide a powerful solution for data warehousing. A key benefit of the intranet technology is the ability to provide up-to-date information quickly and cost-effectively. Some advantages are listed below:

- Intranets allow the integration of information, making it easy to access, maintain, and update. Data warehouses can be made available worldwide on public or private networks at much lower cost. Web browsers can provide a universal application delivery platform for any data mart or data warehouse user.

As a result, the enterprise can create, integrate, or deploy new, more robust applications quickly and economically. Use of intranets enables the integration of a diverse computing environment into a cohesive information work.

- Information disseminated on an intranet enables a high degree of coherence for the entire enterprise (whether the data content comes from data marts or a central data warehouse), because the communications, report formats, and interfaces are consistent. An intranet provides a flexible and scalable nonproprietary solution for a data warehouse implementation. The intuitive nature of the browser interface also reduces user support and training requirements.
- The Internet can be easily extended into WANs or extranets that serve remote company locations, business partners, and customers. External users can access internal data, drill through, or print reports through secure proxy servers that reside outside the firewall.

## VIII. DATA WAREHOUSE PERFORMANCE

### A. Measuring Data Warehouse Performance

The massive amount data (in terabytes) of data warehouses makes high performance a crucial factor for the success of data warehousing techniques. Successful implementation of a data warehouse on the World Wide Web requires a high-performance, scalable combination of hardware, which can integrate easily with existing systems. Data warehousing involves extracting data from various sources transforming, integrating, and summarizing it into relational management systems residing over a span of World Wide Web servers. Typically as part of the client/server architecture, such data warehouse servers may be connected to application servers which improve the performance of query and analysis tools running on desktop systems. Possibly the most important factor to consider in arriving at a high-performance data warehouse environment is that of end-user expectations. These expectations represent unambiguous objectives that provide direction for performance tuning and capacity planning activities within the data warehouse environment.

The basis for measuring query performance in the data warehouse environment is the time from the submission of a query to the moment the results of the query are returned. A data warehouse query has two important measurements:

1. The length of time from the moment of the submission of the query to the time when the first row/record is returned to the end user
2. The length of time from the submission of the query until the row is returned

The data warehouse environment attracts volumes of data that have never before been experienced in the information processing milieu. In previous environments, volumes of data were measured in the thousands (kilobytes) and millions (megabytes) of bytes of data. In the data warehouse environment volumes of data are measured in gigabytes and terabytes of data. Thus, there are many orders of magnitude of difference between these measurements. Some aspects of improving performance have already been discussed earlier in this article, such as indexing and denormalization. There are also other important aspects, such as use of hardware architecture that is parallelized.

Optimizing data structures in the data warehouse environment is an important, but difficult issue, because many different sets of requirements must be satisfied all at once. Therefore great care must be taken in the physical organization of the data in the warehouse.

### B. Performance Issues and Data Warehousing Activities

Performance issues are closely tied to data warehousing activities at various stages:

- **Base level architecture-hardware and software—** Issues that need to be considered include whether the hardware platform supports the volume of data, the types of users, types of workload, and the number of requests that will be run against it, whether the software platform organizes and manages the data in an efficient and optimal manner, as well as others.
- **Design and implementation of the data warehouse platform based on usage and data—** There are various issues related to different aspects, such as:
  1. *Database design.* For example, we need to know whether the different elements of data have been profiled so that the occurrences of data that will exist for each entity are roughly approximated.
  2. *Usage and use profiles.* For example, we need to know whether the database design takes into account the predicted and/or known usage of the data.

- **Creation of the programs and configuration of tools that will make use of the data—**For example, we need to know whether the queries or other programs that will access the data warehouse have been profiled, information about the programmers, etc.
- **Post warehousing development—**After programs are written and the data warehouse is being populated, the ongoing system utilization needs to be monitored and system guidelines—service management contracts—need to be established.

If an organization follows these guidelines and carefully considers performance at each appropriate point in time, the organization will arrive at a point where performance is truly optimal.

A final remark that must be put here is data mart performance. Although performance related to a data mart shares many considerations with data warehouse as a whole, limiting a data mart's scope ensures that the resulting data mart will fit within the scalability limits of an OLAP database server and permits the analysis at hand to be conducted without the distractions presented by extraneous data. Using an OLAP database server, in turn, allows the use of OLAP indexing and presummarization techniques to deliver rapid response times and intuitive access.

# IX. DATA WAREHOUSES, OLAP, AND DATA MINING

## A. Basics of OLAP

In this article we have repeatedly emphasized the close relationship between data warehousing and OLAP. In addition, in many cases data warehouses also play an enabling technique for data mining. Therefore, it is important to examine the relationship among data warehouses, OLAP, and data mining.

OLAP, as a multidimensional analysis, is a method of viewing aggregate data called measurements (e.g., sales, expenses, etc.) along a set of dimensions such as product, brand, stored, month, city, state, etc. An OLAP typically consists of three conceptual tokens. Each *dimension* is described by a set of attributes. A related concept is *domain hierarchy;* for example, "country," "state," and "city" form a domain hierarchy. Each of the numeric *measures* depends on a set of dimensions, which provides the context for the measure. The dimensions together are assumed to uniquely determine the measure. Therefore, the multidimensional data view a measure as a *value* in the multidimensional space of dimensions.

There are several basic approaches to implementing an OLAP:

- *ROLAP (relational OLAP).* OLAP systems that store all information (including fact tables) as relations. Note that the aggregations are stored with the relational system itself.
- *MOLAP (mulitdimensional OLAP).* OLAP systems that use arrays to store multidimensional datasets.

In general, ROLAP is more flexible than MOLAP, but has more computational overhead for managing many tables. One advantage of using ROLAP is that sparse data sets may be stored more compactly in tables than in arrays. Since ROLAP is an extension of the matured relational database technique, we can take advantage of using standard query language (SQL). In addition, ROLAP is very scalable. However, one major advantage is its slow response time. In contrast, MOLAP abandons the relational structure and uses a sparse matrix file representation to store the aggregations efficiently. This gains efficiency, but lacks flexibility, restricts the number of dimensions (7–10), and is limited to small databases. (Remark on dimension: a relation can be viewed as a 2D table or $n - D$ table (each attribute represents a dimension). One advantage of using MOLAP is that dense arrays are stored more compactly in the array format than in tables. In addition, array lookups are simple arithmetic operations which result in an instant response. A disadvantage of MOLAP is long load times. Besides, MOLAP design becomes massive very quickly with the addition of multiple dimensions. To get the best of both worlds, we can combine MOLAP with ROLAP. Other approaches also exist.

## B. Relationship between Data Warehousing and OLAP

Having described the basic architecture of data warehouses, we may further describe the relationship between data warehousing and OLAP as follows. Decision-support functions in a data warehouse involve hundreds of complex aggregate queries over large volumes of data. To meet the performance demands so that fast answers can be provided, virtually all OLAP products resort to some degree of these aggregates. According to a popular opinion from OLAP Council, a data warehouse is usually based on relational technology, while OLAP uses a multidimensional view of aggregate data to provide quick access to strategic information for further analysis. A data warehouse stores tactical information that answers “who” and “what” questions about past events. OLAP systems go beyond those questions; they are able to answer “what if” and “why” questions. A typical OLAP calculation is more complex than simply summarizing data.

Most data warehouses use *star schemas* to represent the multidimensional data model. In a star schema there is a single fact table (which is at the center of a star schema and contains most of the data stored in the data warehouse), and a set of dimension tables which can be used in combination with the fact table (a single table for each dimension). An example of star schema is shown in Fig. 3.

The star schema model of data warehouses makes join indexing attractive for cross-table search, because the connection between a fact table and its corresponding dimension tables are the foreign key of the fact table and the primary key of the dimension table. Join indexing maintains relationships between attribute values of a dimension and the corresponding rows in the fact table. Join indices may span multiple dimensions to form composite join indices. Data mining algorithms can take advantage of this kind of facility. In fact, usually data mining algorithms are presented on a single table. Join indexing makes such assumptions reasonable.

Join operations in a star schema may be performed only between the fact table and any of its dimensions. Data mining has frequently been carried out on a view that is joined by the fact table with one or more dimension tables, followed by possible project and select operations. In addition, to facilitate data mining, such a view is usually materialized.

A very useful concept for OLAP is *data cube,* which presents a materialized view involving multidimensional data. The two most well-known operations for OLAP queries are

1. *Roll-up.* This operation takes the current data object and does a further group-by on one of the dimensions. For example, given total sale by day, we can ask for total sale by month.
2. *Drill-down.* As the converse of rule-up, this operation tries to get more detailed presentation. For example, given total sale by model, we can ask for total sale by model by year.

Other operations include: *pivot* (its result is called a cross-*tabulation*), *slice* (it is an equality selection, reducing the dimensionality of data), and *dice* (it is the range selection).

## C. Integration of OLAP and Data Mining in Data Warehouses

It has been noted that there are significant semantic differences between data mining and OLAP. Although both OLAP and data mining deal with analysis of data, the focus and methodology are quite different. In this section, we provide a much needed discussion on this issue and use several examples to illustrate these dif-

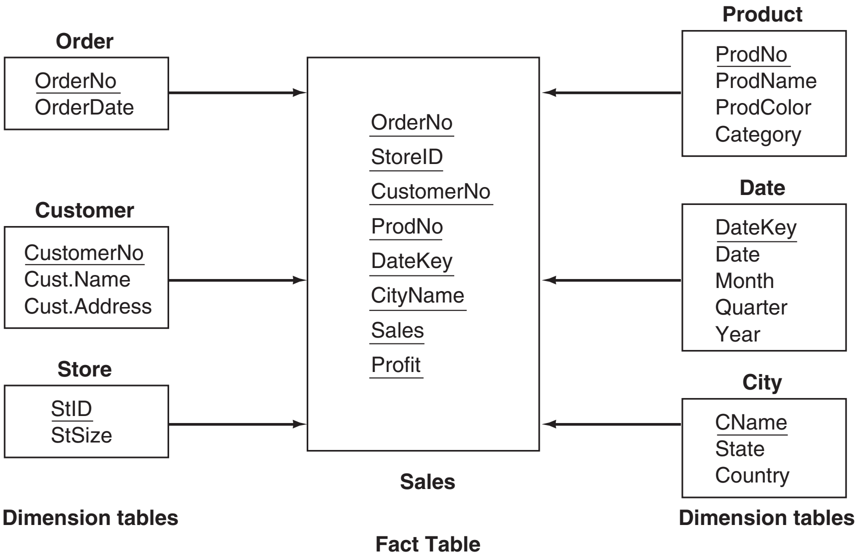


**Figure 3** A star schema.

ferences. We point out the difference of data mining carried out at different levels, including how different types of queries can be handled, and how different semantics of knowledge can be discovered at different levels, as well as how different heuristics may be used.

We point out that different kinds of analysis can be carried out at different levels: What are the features of products purchased along with promotional items? The answer for this query could be association rule(s) at the granularity level, because we need to analyze actual purchase data for each transaction which is involved in promotional items (we assume information about promotional items can be found in product price).

- What kinds of products are most profitable? This query involves aggregation, and can be answered by OLAP alone.
- What kinds of customers bought the most profitable products? This query can be answered by different ways. One way is to analyze individual transactions and obtain association rules between products and customers at the granularity level. An alternative way is to select all most profitable products, project the whole set of customers who purchased these products, and then find out the characteristics of these customers. In this case we are trying to answer the query by discovering characteristic rules at an aggregation level. (For example, customers can be characterized by their addresses.)

The above discussion further suggests that data mining at different levels may have different semantics. Since most people are familiar with semantics of knowledge discovered at the granularity level, here we will provide a discussion emphasizing what kind of difference is made by the semantics of knowledge discovered at aggregation levels (which will be referred to as aggregation semantics). Nevertheless, OLAP and data mining should be and can be integrated for decision support.

## X. CONCLUSION

In this article we discussed the basics of data warehousing techniques, as well as related issues such as OLAP and data mining. Due to the rich contents and rapid development of this area, we can only present the most fundamental ideas. Readers who are interested in more details of data warehousing are referred to Inmon (1996), Kimball et al. (1998), and Singh (1999). Readers interested in the relationship between data warehousing and data mining, particularly detailed techniques of data mining, can find useful materials in Han and Kamber (2000). Finally, for users who are interested in a more global picture but without much technical detail, Chen (1999) and Chen (2001) should not make you disappointed.

There are also plenty of web sites available on data warehousing and data marts. For example, http://www.dwinfocenter.org/articles.html contains numerous articles related to data warehousing and data marts applications. On the other hand, for research-oriented readers, http://www-db.stanford.edu/ contains very useful materials from the well-known Stanford data warehousing project, including discussions on various future research directions related to data warehouses.

## SEE ALSO THE FOLLOWING ARTICLES

Data Mining • Data Modeling: Entity-Relationship Model • Data Modeling: Object-Oriented Model • Distributed Databases • Model Building Process • Network Database Systems • On-Line Analytical Processing (OLAP)

## BIBLIOGRAPHY

Chaudhuri, S., and Dayal, U. (1997). An overview of data warehousing and OLAP technology. *SIGMOD Record,* 26(1), 65–74.

Chen, Z. (1999). *Computational intelligence for decision support.* Boca Raton, FL: CRC Press.

Chen, Z. (2001). *Data mining and uncertain reasoning: An integrated approach.* New York: John Wiley.

Han, J., and Kamber, M. (2000). *Data mining: Concepts and techniques.* San Francisco: Morgan Kaufmann.

Harinarayan, V. (1987). Issues in interactive aggregation. *Data Engineering Bulletin,* 20(1), 12–18.

Kimball, R., Reeves, L., Ross, M., and Thornthwaite, W. (1998). *The data warehouse lifecycle toolkit.* New York: John Wiley.

Inmon, W. H. (1996). *Building the data warehouse.* New York: John Wiley.

Inmon, W. H. (1999). *Data warehouse performance.* New York: John Wiley.

Rundersteiner, E. A., Koeller, A., and Zhang, X. (2000). Maintaining data warehouses over changing information sources. *Communications of the ACM,* 43(6), 57–62.

Silberschatz, A., Korth, H., and Sudarshan, S. (1997). *Database system concepts,* 3rd ed. New York: McGraw-Hill.

Singh, H. (1999): *Interactive data warehousing.* Upper Saddle River, NJ: Prentice Hall.

# Decision-Making Approaches

**Theodor J. Stewart**
*University of Cape Town*

## GLOSSARY

**cognitive bias** Any tendency by human decision makers to adopt simplifying heuristics, which result in the selection of certain classes or types of action, for reasons other than genuinely informed preference.

**criterion** Any particular point of view or dimension of preference according to which decision alternatives or courses of action can be compared in more-or-less unambiguous terms.

**decision support system** A computer system which assists decision makers in exploring the consequences of decisions in a structured manner and in developing an understanding of the extent to which each decision alternative or option contributes toward goals.

**goal programming** A mathematical programming technique for identifying decision alternatives or courses of action which approach decision-making goals most closely in an aggregate sense.

**multiple criteria decision analysis (MCDA)** A branch of management science in which complex decision problems are first decomposed into underlying unitary criteria; once clarity has been obtained regarding preference orderings for each individual criterion, these are gradually reaggregated to develop a holistic preference ordering.

**outranking** An approach to MCDA in which decision alternatives are compared pairwise in order to establish the strength of preference for and against the assertion that one alternative is at least as good as the other.

**value function** A numerical function constructed so as to associate a preference score with each decision alternative (either holistically or for one or more individual criteria) in order to provide a tentative rank ordering of such alternatives in a defensible manner.

**value measurement** An axiom-based theoretical foundation for constructing value functions.

**DECISION MAKING** is a fundamental activity of all management, and research and literature concerning decision-making processes and paradigms occur in many fields, including management theory, psychology, information systems, management science, and operations research. As the present volume is devoted to information systems, the primary focus of this article will be the manner in which the decision-making process can be supported and guided by appropriate *decision support systems.* We shall, on one hand, broaden the view of "decision support" to include not only the provision of interactive computer systems, but also other modes of facilitation by which decision-making effectiveness may be enhanced. On the other hand, we shall restrict the concept of computerized decision support systems to those systems which assist the decision maker directly in exploring and evaluating alternative courses of action. This is in contrast to other common uses of the term which refer to any system which processes or analyzes data needed for decision making (including "data mining"), but without necessarily providing direct support to the search for a preferred course of action.

Our approach will be in essence *constructive* (sometimes termed *prescriptive*) in the sense that our main thrust will be to present models that are neither fully *descriptive,* i.e., indicating how unaided decisions are actually made, nor *normative* in claiming that this is how decisions should be made. By the constructive (prescriptive) approach, we mean a process by which decision makers can be assisted to develop preferences between alternative courses of action in an internally consistent and coherent manner, so that they can have confidence that the emerging decision does in fact satisfy their long-run goals. In discussing this approach, we shall look at the structuring of decision problems, empirical research on the psychology of decision making, and the provision of quantitative decision support aids.

## I. THE DECISION-MAKING PROBLEMATIQUE

At an elementary level, decision making may be viewed simply as a choice between a number of available courses of action. This is naive. In virtually all real-world settings, potential courses of action are seldom, if ever, self-evidently available, simply awaiting a choice between them.

At strategic planning levels, decision problems will initially be totally *unstructured.* There may be no more than a general feeling of unease that things are not going as they should, that we are not achieving what we ought, and that "something needs to be done." For example, a company might have previously had an effective monopoly in a particular market, but over the past year costs have been escalating, while one or two other players are entering the market so that the simple expedient of raising prices could cause rapid erosion of market share. There is, however, incomplete knowledge of the causes of the cost escalations and of how strong the emerging competition may prove to be, but nevertheless "something has to be done" before the situation gets out of hand. At this stage, however, there may be a substantial lack of awareness, not only about what courses of action are open but even about what the organizational or personal goals actually are. What are we trying to achieve by the interventions we might be planning to make?

Even when the decision problem may seem superficially to be more clear-cut, the problem is in most cases still only *partially structured.* The apparent decision problem may present itself as a choice between some clear alternatives: Do we launch this new product or not? Do we purchase this software system or the other? Which of the options offered by our travel agent should we take up for our vacation this year? We say that these are only partially or *semi-structured,* as there remain many issues that are not yet resolved. Even for the options on the table, the consequences have probably not been completely explored. Decision makers may still need to identify their decision goals and objectives before they even know what consequences are relevant to explore. The initial apparent options may only be symptomatic of deeper underlying problems, and when these become better understood there may be a need to seek other alternative options. In fact, as the deeper issues are given consideration, it may become evident that the problem is still to a large extent unstructured.

What the previous two paragraphs have sought to indicate is that nontrivial management decision problems are essentially never fully *structured* at the outset in the sense that the *alternative courses of action,* the *criteria* by which each need to be assessed, and the *consequences* of each alternative in terms of these criteria are fully and completely known. Thus, the first step in any decision support process is to provide an adequate degree of structuring, so that informed decisions can be made in accordance with a coherent set of objectives. We shall return to this point in Section II.

Even once a requisite structure has been developed for a decision problem, it is not always true that the decision itself is a simple choice between alternatives. Roy (1996), for example, provides a classification into a number of decision *problematiques,* which with some modification and extension we might summarize as follows:

1. Simple choice of one from among a number of explicit alternatives.
2. Sorting of alternatives into a number of categories: For example, a company might wish to classify potential subcontractors into "good," "acceptable," and "poor" categories for purposes of awarding future contracts.
3. Ranking of alternatives: For example, potential development projects might be ranked from highest to lowest priority, with the intention being that projects would be worked on in priority order until time or other resources are exhausted.
4. Designing alternatives or creating portfolios in the sense of creating a complete strategy from component elements in a coherent manner: For example, policies for future land use in a region may be constructed by a combination of public sector investments, legislative programs controls, tax incentives, etc.

The decision support approaches described particularly in Section IV can, in principle, be applied to any of the situations described above, but the precise implementation may need to be adapted to the relevant problematique.

Ultimately, decisions may be approached in a number of different ways. At a risk of over-simplification, we can identify at least three distinct approaches which may be adopted in decision-making processes as follows:

1. The *political* or *advocacy* approach: Champions for different courses of action prepare and present the arguments favoring their preferred option. This may occur in a formalized open forum or by a process of lobbying. Options backed by stronger and better arguments tend to survive at the expense of others, until either by consensus or formal voting procedures a winner ultimately emerges. Political approaches tend to dominate in multistakeholder contexts, when different groups may have substantially divergent agendas.
2. The *organizational* approach: This is similar to the political advocacy approach, but less directly confrontational. Different sectors of the organization may be tasked with establishing "pros" and "cons" of alternative courses of action from different perspectives. The organizational approach may be found to dominate in group decision-making contexts when group members do, to a large extent, share common goals.
3. The *analytical* (or "*rational*") approach: Here the attempt is made to identify the full set of alternatives from which the choice has to be made and the degree to which each satisfies explicitly stated management goals. In principle, then, it remains only to select the alternative for which this degree of satisfaction is maximized. A purely rational approach probably only applies for decision contexts involving a single decision maker or at most a small relatively homogeneous group. However, as we will be discussing, a degree of analysis can provide valuable support to the political and/or organizational approaches.

It is probable that none of the above approaches is used in isolation, but that aspects of each are used in any real problem. The modern decision support philosophy aims at providing some form of analytical basis from which the political and organizational components of the decision process can be supported and made more effective and internally coherent. The decision support system thus facilitates communication between stakeholders in the process and the emergence of a decision that is broadly satisfactory and concordant with the organization's goals. It is this view of decision support which underlies the methods of "decision analysis" discussed in the remainder of this article.

## II. APPROACHES TO PROBLEM STRUCTURING

As indicated at the start of the previous section, decision problems seldom, if ever, present themselves in a neatly structured form as a simple choice between a number of alternatives with the aim of achieving well-defined objectives. On the other hand, the application of the decision support procedures described particularly in Section IV does require that the problem be given some degree of structure at least. Most nontrivial decision-making contexts involve multiple stakeholders, so that the structuring phase will in general be a group effort, as will be assumed for purposes of the discussion which follows.

The management literature is filled with many suggestions for structuring decision problems, ranging from relatively informal SWOT analysis (strengths, weaknesses, opportunities, and threats) to sophisticated computer-assisted brainstorming techniques. The management science literature includes such concepts as soft-systems methodology and cognitive mapping, many of which are reviewed in the volume edited by Rosenhead and Mingers (2001). The underlying basis of all of these methodologies is a two-phase approach, which may be described as:

1. A *divergent phase,* in which the aim is simply to identify all issues and concerns relevant to the decision at hand
2. A *convergent phase,* in which the connections and interactions between these issues and concerns are systematically explored in order to structure, classify, and cluster them.

It has been our experience that the divergent phase is often best carried out in a relatively low technology manner in order to stimulate a maximum degree of human interaction between participants, although some groupware systems have been developed to mimic the informal processes described here. A simple approach is to bring group members together in some form of workshop, at which each participant is provided with a set of cards or self-adhesive notelets on which to jot down key points in response to a well-defined question (such as, "What issues do we need to take into consideration when responding to the entry of new competitors into the market?"). These can then be placed up on a wall, initially in random positions, to serve as a focus for discussion, during which the cards or notelets can be moved around into groups or clusters representing similar issues. In this way, a shared vision of the decision problem emerges.

It is useful to guide the clustering by making use of simple checklists, often represented by some form of acronym. For example, the soft-systems methodology developed by Checkland (1981) makes use of "CATWOE" (Customers, Actors, Transformations, Worldview, Owners of the problem and Environment, i.e., external constraints and demands). The author and colleagues have found it useful to build up a structure around a "CAUSE" framework defined in the following:

- **C**riteria: The various points of view or perspectives against which different courses of action need to be evaluated and compared, for example, costs, public image, worker morale, environmental impacts. Management goals will ultimately need to be expressed in terms of levels of performance for each criterion.
- **A**lternatives: The courses of action which are available to decision makers.
- **U**ncertainties: These may include both potentially resolvable uncertainties due to lack of current knowledge (e.g., whether or not a competitor will be launching a new product this year) and fundamentally unknowable risks (e.g., a major earthquake in California).
- **S**takeholders: Who has a concern in the outcome of the decision? Who can influence the consequences of a decision (including the possibility of sabotaging it)? Who has the political or organizational power to veto certain actions?
- **E**nvironment: What external constraints and pressures limit the decision makers' freedom of action?

Once the key components building up a structured view of the decision problem have been identified as above, it is useful to guide the group into identifying causative and associative links between these elements, with the ultimate aim of building up a shared vision of the manner in which choice of different courses of action (i.e., the decisions) impact on the criteria (i.e., on goal achievement). The soft-systems methodologies of Checkland (1981) and the cognitive mapping concepts described by Eden and Ackermann (1998) provide useful tools in this regard, the latter being well-supported by the Decision Explorer software. At the end of the day, for purposes of applying formal decision analytical approaches, it is necessary to summarize the problem structure which emerges from the structuring process in terms of a set (often just a finite list) of alternative courses of action and a description of the impacts of each alternative on levels of performance of each criterion, where any degree of uncertainty in evaluating these impacts is clearly identified. The structured decision problem thus becomes that of selecting the alternative which best satisfies the goals implied by each criterion.

## III. COGNITIVE BIASES IN DECISION MAKING

In the next section we shall be discussing technical procedures of decision support. These generally require that the user, or "decision maker," provide a variety of value judgements as part of the process of identifying a desired course of action. It is necessary, therefore, for the decision support analyst to have some understanding of the manner in which decision makers may respond to the questions asked and the potential for these responses to bias the results coming from the decision support system.

Nontrivial decision-making situations must involve decision makers in two key issues, namely, (1) management goals and objectives and (2) the uncertainties and risks associated with achievement of them. For decision-making processes to be effective, both decision makers and analysts supporting them need to be aware of biases and heuristics inherent in human judgements regarding these issues.

In a purely "rational" approach, all objectives would be explicit at the start of the decision-making process, and it would remain only to assess the extent to which alternative courses of action satisfy these objectives. Life is, however, seldom like that. Simon (1976) argued that people exhibit bounded rationality and tend to "satisfice" rather than to "optimize." This means that decision makers tend to focus on a relatively small number of objectives, namely, those which are perceived to be the most critical or currently least well satisfied. In developing and/or evaluating alternative courses of action, effort is directed initially at seeking improvement on these objectives. Once an adequate level of satisfaction has been achieved for these objectives, then attention turns to other objectives which now appear more pressing. Satisficing is thus a common heuristic approach to decision making. It can also be a powerful and useful heuristic, and for this reason it is sometimes incorporated into formal decision support systems, particularly the goal programming approaches to be described in Section

IV. Decision makers and decision analysts need to be aware, however, that the quality of results obtained will be dependent upon goals for each objective being demanding but still realistic. If such goals are insufficiently demanding, the process might terminate too early, but if the goals for the initial objectives are made too unrealistically demanding, then potential gains on other objectives may never be realized.

Certain biases derive from the group context in which important decisions are often made (see, for example, Janis and Mann, 1977). On the one hand, defensive avoidance of conflict by some group members can lead to procrastination (postponing a decision in the hope that the conflict will vanish); attempts to shift responsibility for the decision to other people or organizational structures; and overemphasis on the favorable consequences of alternative courses of action, while downplaying unfavorable consequences. Good decision making requires that these conflict-avoidance biases be recognized and confronted. A related problem in the group decision-making context is that of "group-think," in which the group moves to acceptance of a consensus (and overconfidence in the correctness of this consensus decision) without fully exploring consequences in terms of all goals or criteria. Such group-think may arise, *inter alia,* through members' fear of ridicule or accusations of time-wasting should they speak against a perceived majority view. The multiple criteria decision analysis approach described in Section IV seeks to counter these group biases by seeking to establish all relevant "criteria" quickly and before conflict reaches destructive levels.

Some of the more detailed studies of biases in decision making have been undertaken within the context of risk and uncertainty, the pioneering work being that of the decision psychologists Kahneman and Tversky. Examination of these biases does suggest that they may be applicable to a broader range of human judgmental tasks than those of probability assessment and inductive inference and may thus be relevant to the designers of the decision support system. It is thus useful to briefly record some of these biases here. It is beyond the scope of the present article to review this topic in detail (see, for example, Kahneman, Tversky and Slovic, 1982), but in the following few subsections we present an outline of three commonly observed biases.

## A. Availability

When asked to assess the probabilities or risks associated with specified events, people tend to associate greater credibility with those events for which it is easy to recall examples of similar outcomes in the past or for which examples are easy to imagine. The problem is that some events generate considerably more publicity than others and thus are easier to recall. For example, a major airline crash will receive wide news coverage for days or weeks after the event, whereas fatalities from motor accidents receive, at most, minor coverage in local newspapers. As a result, many people may tend to overestimate the risk from airline accidents and to underestimate the risk of road accidents. Similarly, greater probabilities will be associated with events for which it is easy to construct imaginary scenarios, which may or may not be related to the inherent propensity for such events to happen.

A particular problem caused by the availability bias, and related also to the next bias which we discuss ("representativeness"), is that people find it easier to recall instances which confirm prior prejudice (e.g., that a particular class of driver is more reckless than others) than those which contradict it. This, in turn, leads to enhanced estimates of the associated probabilities and reinforcing of the prejudice.

Good decision-making practice should thus allow time for free-thinking and brainstorming to enable to the decision maker to explore and to imagine a wider range of outcomes before committing to a final decision. Good decision support system design needs to encourage and facilitate such processes.

## B. Representativeness

Suppose we observe a sequence of eight tosses of a coin. Most people will judge a sequence of heads and tails given by HHHHHTTT as inherently more surprising and less likely than a sequence such as HHTTHTHH. Yet statistically both outcomes are equally probable (1 in 256). The reason for the fallacious judgement appears to be that the latter is more characteristic or "representative" of what we expect from a random sequence. In other words, people rate as more likely those outcomes which are viewed as more representative of their expectations. For example, when given a personality description of an individual unknown to them, people will tend to consider it most likely that the individual belongs to the occupational group (such as engineer, lawyer, psychologist) for which the personality type appears typical without taking into consideration important factors such as the proportion of each occupational group in the relevant population.

The representativeness bias may lead decision makers to ignore some critical information, such as base rate frequencies of outcomes as in the classification into the occupational groups above. It can also lead to substantially false conclusions regarding the existence or otherwise of patterns in data. For example, the HHHHHTTT outcome may result in a conclusion that "luck" has turned in some predictable manner and that more tails can now be expected. In the same way, managers may easily overreact to recent apparent trends, leading to poor decision-making practice.

As with the availability bias, the best antidote to the representativeness bias would be to take time to critically examine the data and to ensure that all relevant data are properly taken into consideration, generally through the use of formal statistical analysis.

### C. Anchoring and Adjustment

In forecasting risks (i.e., probabilities) or future events such as prices or demands, people will frequently start from some initial nominal value and then adjust this up or down as other factors are taken into consideration. Such adjustments tend, however, to be insufficient to account for the new information, so the estimates remain too tightly anchored to the initial values. The initial values may be entirely randomly generated (for example, during preliminary unstructured discussions) or may be linked to current situations or to simple statistical trends. The result is that the ranges of future variation may be seriously underestimated and that decision makers may be overconfident in their projections or forecasts so that risks are inadequately taken into consideration during the decision process.

Once again, the bias needs active compensation by building formal means of identifying alternative futures at an early stage of deliberation before estimates are too solidly "anchored." One means of achieving this is by incorporation of scenario planning concepts (see van der Heijden, 1996) into the total decision-making process. We shall return to this concept in the discussion of risk and uncertainty in Section V.

## IV. MULTIPLE CRITERIA DECISION AID AND SUPPORT

### A. General Principles

Perhaps the most critical demand on decision makers is the need to achieve balance between conflicting goals or objectives. If consequences of decisions were entirely one dimensional (e.g., maximization of profit) or if it were possible to simultaneously optimize all objectives, then no true "decision making" is involved; the process can be left to a computer. Human decision making, involving the making of value judgements or tradeoffs, comes into play when decision makers recognize many different goals or criteria for comparing different courses of action that are to a greater or lesser extent in conflict with each other.

Much of the management theory and management science literature does recognize the existence of multiple goals, but very often this recognition is implicit rather than explicit. A recent notable exception has been the "balanced scorecard" concept introduced by Kaplan and Norton (1996), in which the authors clearly identify the need for balance between financial, customer-oriented, internal operation, and learning and growth goals in any organization, each of which can be subdivided further. They give attention particularly to the management structures necessary to ensure such balance.

The most explicit recognition of multiple goals may be found in the range of management science techniques which have been classified as multiple criteria decision-making (MCDM) methods, or *multiple criteria decision analysis* or *aid* (MCDA). The characterizing feature of these approaches is the establishment of formal and to some extent quantified procedures for the following three phases of the problem:

1. Identification of relevant criteria, i.e., points of view or axes of preference according to which possible courses of action can be distinguished.
2. Ranking, or possibly more extensive evaluation, of alternative courses of action according to each identified criterion.
3. Aggregation across criteria to establish an overall preference ranking for the alternatives.

We now briefly expand on these three phases before turning to some more detailed description of the tools of MCDA, which as we shall see can be grouped into three broad schools, namely, value measurement or scoring methods, goal and reference point methods, and outranking methods.

#### 1. Identification of Criteria

A *criterion* is defined in this context as any concern, interest, or point of view according to which alternative courses of action can (more-or-less) unambiguously be rank ordered. In selecting criteria for use in decision analysis, the following properties of the set being chosen should be borne in mind:

- *Complete:* Ensure that all substantial interests are incorporated.
- *Operational:* Ensure that the criteria are meaningful and understandable to all role players.
- *Decomposable:* Ensure as far as is possible that the criteria are defined in such a way that meaningful rank orders of alternatives according to one criterion can be identified without having to think about how well the alternatives perform according to other criteria. (The so-called condition of *preferential independence.*)
- *Nonredundant:* Avoid double counting of issues.
- *Minimum size:* Try to use as few criteria as possible consistent with completeness, i.e., avoid introduction of many side issues which have little likelihood of substantially affecting the final decision.

Typically, the identification of appropriate criteria to be used requires a variety of brainstorming techniques, but a review of these is beyond the scope of this article. These issues are discussed further in some of the literature listed in the Bibliography. In many cases it is useful to structure the criteria into a hierarchical value tree, starting with a broad overall goal at the top, systematically broken down into increasingly precise subgoals, until at the lowest level we have the required set of criteria as described above. Such a value tree is illustrated in Fig. 1, which is based on experiences in applying MCDA to land use and water resources planning in the eastern escarpment regions of South Africa. The criteria are the right-hand boxes, namely, household income, number of jobs, and so on down to flood levels. The advantage of such a hierarchical structure is that the application of MCDA can be decomposed, for example, by first evaluating alternatives within a subset of criteria (for example, the three contributing to social benefits) and then aggregating these to give a preference ordering according to "social" issues (thus forming a supercriterion). At a later stage, a further aggregation can combine social, economic, and environmental concerns.

Sometimes a criterion may directly be associated with some measurable quantitative attribute of the system under consideration, for example, cost measures or many of the benefits listed in the value tree of Fig. 1. Such an association may facilitate the next phase of the analysis, but is not critical to the use of the tools to be described below. These can generally be applied even for entirely qualitative criteria such as the personal well-being of employees, provided that alternatives can at least be compared with each other on this basis (i.e., which of the two options contributes more to the "personal well-being").

## 2. Within-Criterion Comparison of Alternatives

At this stage, alternatives are compared and evaluated relative to each other in terms of each identified criterion. The alternatives may be real courses of action or may be hypothetical constructs (performance categories as described below) built up to provide a set of benchmarks against which the real alternatives can

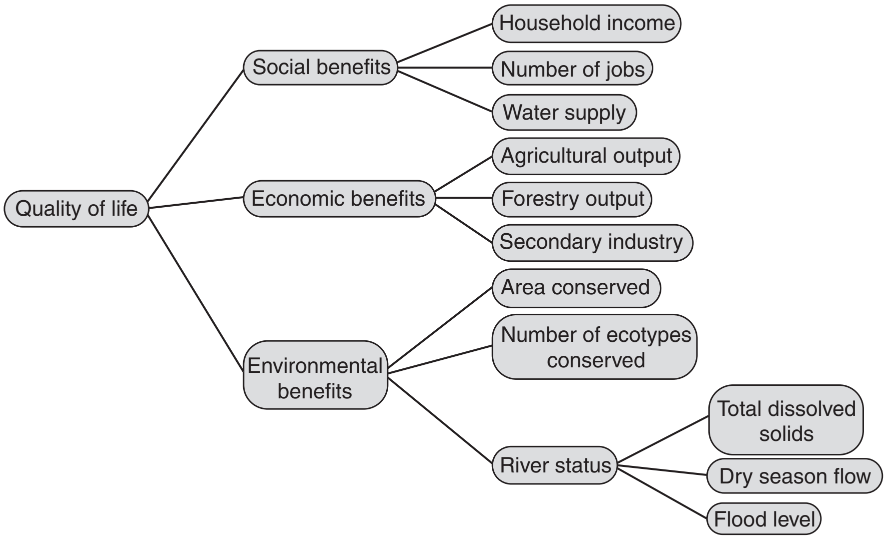


**Figure 1** Illustration of a value tree for a regional land use planning problem.

be evaluated. In both cases, however, the fundamental requirement is to be able to rank the alternatives from best to worst in terms of the criterion under consideration. If this cannot be done, then the definition of the criterion needs to be revisited.

An important feature of this process is that it is carried out separately for each criterion and does not need reduction to artificial measures such as monetary equivalents. All that is required is for the decision maker, expert, or interest group to be able to compare alternatives with each other in terms of their contribution to the goals represented by the criterion under consideration. In some cases, criteria will be qualitative in nature (for example, a criterion such as personal well-being), so the rank ordering will have to be subjective or judgmental in nature. For smaller numbers of alternatives (say up to about seven or nine), this creates no problem as it will generally be possible to compare alternatives directly to generate the required rank orderings or evaluations in an unambiguous manner. For larger numbers of alternatives, however, direct comparisons become more difficult, and it is convenient to define a small number of *performance categories,* i.e., descriptions of different levels of performance that may be achieved, expressed as mini-scenarios (the hypothetical alternatives or outcomes mentioned earlier). Each actual alternative is then classified into that category which best matches its performance in terms of this criterion or possibly classified as falling between two adjacent categories. Since the categories are preference ordered, this implies a partial ordering of the alternatives, which is usually adequate for the application of many MCDA procedures (especially when linked to extensive sensitivity studies).

### 3. Aggregation across Criteria

This is perhaps the most crucial phase, in which the generally conflicting preference orderings corresponding to the different criteria need to be reconciled or aggregated to produce a final overall preference ordering. The process can never be exact, as it must inevitably involve imprecise and subjective judgements regarding the relative importance of each criterion. Nevertheless, with due care and sensitivity analysis, a coherent picture can be generated as to which are the most robust, equitable, and defensible decisions.

An important point to recognize is that the method of aggregation is critically dependent upon the methods of evaluation of alternatives used in the previous phase. Aggregation inevitably involves some assessment of the importance of each criterion relative to the other criteria. This is typically expressed in terms of some form of quantitative "*weight*." The meaning, interpretation, and assessment of importance weights is an often controversial aspect of MCDA practice, since the appropriate numerical weights to be used in any MCDA procedure must depend both on the specific procedure and on the context of the alternatives under consideration. It is, however, also true that many people will express judgements of relative importance (e.g., that environmental issues are "much more important," or even something like "three times as important," in comparison with economic issues) without concern for either the particular context or the methods of analysis being used. It is fallacious to incorporate such intuitive statements of relative importance uncritically into MCDA (although this often seems to be done). In the application of MCDA, care must be taken to match the elicitation of importance weights to methods used and the context.

The different schools of MCDA differ in the manner in which they approach the second and third phases mentioned earlier. The three schools are reviewed in subsections IV.B to IV.D. In order to describe the methods, it is useful at this stage to introduce some notation. For purposes of discussion, we shall suppose that a choice has to be made between a discrete number of alternatives denoted by $a,b,c,...$. Many of the methods described below are easily generalized to more complicated settings, for example, multiple objective linear programming, but this would unnecessarily complicate the present discussion. We suppose that $m$ criteria have been identified, which we shall index by $i = 1,2,...,m$. If criterion $i$ can be associated with a quantifiable attribute of the system, we shall denote the value of this attribute for alternative $a$ by $z_i(a)$. Note that even if the attribute is naturally expressed in categorical terms (very good, good, etc.), this is still "quantifiable" in our sense as we can associate some numerical value with each category to represent the ordering.

## B. Value Measurement or Numerical Scoring Approaches

In this approach, we seek to construct some form of value measure or score, $V(a)$, for each alternative $a$. In principle, the value measures do not need to possess any particular numerical properties apart from preservation of preference order, i.e., such that $V(a) > V(b)$ if and only if $a$ is preferred to $b$.

Within the usual framework of MCDA, we start by extracting partial values or scores for the alternatives as evaluated in terms of each criterion. These we denote by $v_i(a)$ for $i = 1,2,...m$. Where criteria are associated with quantifiable attributes $z_i(a)$, it is evident that these partial values need to be functions of the attributes, i.e., $v_i(a) = v_i(z_i(a))$. We shall return to this case shortly, but let us first consider the general case without necessarily making the assumption of the existence of such attributes.

Clearly, $V(a)$ must be some function of the partial values $v_1(a), v_2(a),...,v_m(a)$. We shall suppose that the selection of a family of criteria satisfies the property of preferential independence discussed under selection of criteria and that the partial values are constructed so as to satisfy an *interval scale* property [i.e., such that equal increments in any specific $v_i(a)$ have the same impact or value in terms of tradeoffs with other criteria, no matter where they occur in the available range of values]. It can be shown that under these assumptions, it is sufficient to construct $V(a)$ as an additive function of $v_i(a)$, i.e.,

$$V(a) = \sum_{i=1}^{m} w_i v_i(a) \qquad (1)$$

where $w_i$ is an importance weight associated with criterion $i$.

In applying value measurement theory, the key practical points are those of assessing the partial values and the weights.

Partial values can be assessed by direct comparison of alternatives or indirectly through an associated quantitative attribute $z_i$. Let us first examine the *direct comparison* approach. A useful way to assess partial values in this case is by means of the so-called "thermometer scale" illustrated in Fig. 2. For example, in a problem such as that on which the value tree of Fig. 1 is based, we might need to compare $m = 6$ policy alternatives, for example, involving three different patterns of land use (farming, forestry, and conservation) with and without the construction of a proposed large dam. For convenience, we might label the alternatives as "scenarios" A–F. Now consider a criterion such as water supply to undeveloped rural communities in the area. Since the desirability of each scenario from the point of view of this criterion may involve consideration of a number of poorly quantified issues such as convenience of access to sufficient clean water, it may not be possible to define a simple measure of performance. By the process of direct comparison on the thermometer scale, however, we can still get a meaningful evaluation for use in the value function model.

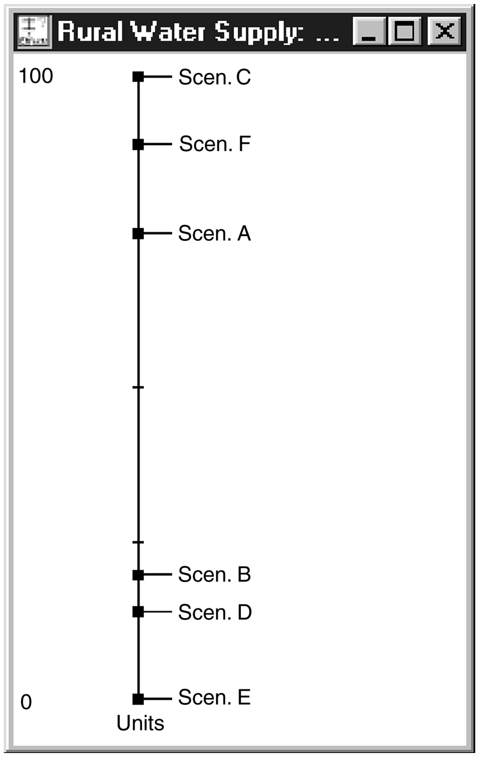


**Figure 2** Illustration of a thermometer scale.

We start simply by identifying the best and worst of the six alternatives according to this criterion of rural water supply. (This judgment is left to those considered best able to make such an assessment.) Suppose that these are identified as scenarios C and E, respectively. Then C is placed at the top of the scale (denoted for convenience in Fig. 2 by an arbitrary score of 100) and E is placed at the bottom of the scale (denoted again for convenience at the 0 point of the scale). A third alternative, say scenario A, is then selected for evaluation by those performing the assessment. It is placed on the scale between C and E in such a way that the magnitudes of the relative spacings, or "gaps," between C and A and between A and E represent the extent to which A is better than E but worse than C. For example, the position shown for scenario A in Fig. 2 is at about the 75% position, suggesting that the gap from E to A (the extent to which A is better than E) is about three times the gap from A to C. Put in another way, we could say that moving from E to A achieves ¾ of the gain realized by moving all the way from E to C. There is generally no need to be overly precise in these judgments, as long as the sizes of the gaps appear qualitatively correct.

Thereafter, each of the remaining alternatives are examined one at a time and placed firstly in the correct rank position among the previously examined alternatives. For example, B may then be placed below

A. Once the ranking is established, the precise position of the alternative is assessed, again taking into consideration the gaps between it and the two alternatives just above and below it in the rank ordering. In this process, the user may wish to readjust the positions of the previously examined alternatives. Figure 2 illustrates a final thermometer scale for all six policy scenarios (alternatives) evaluated according to this criterion of "rural water supply." The full rank ordering of the scenarios is C-F-A-B-D-E. The gap between C and F is perceived to be relatively small, and even A is not far behind, so that C, F, and A are all judged to be relatively good in terms of this criterion. There is then a big gap between A and B, so that the remaining three alternatives are perceived to be much less satisfactory than C, F, and A, although there is little choice between B and D which are still somewhat better than E. It seems that people from widely differing backgrounds can relate relatively easily to diagrams such as Fig. 2 and do participate freely in adjusting the gaps to correspond to their own perceptions of the values of the alternatives. Thus, the thermometer scale diagram is not only a useful tool for assessing partial values, but also for communication between groups.

*Indirect evaluation* requires the association of quantitative attributes $z_i(a)$ with all criteria. The evaluation consists of two stages. We first evaluate a *value* function, say $v_i(z_i)$, which associates scores with all possible values of the associated attribute $z_i$ between a specified minimum and maximum. In theory this should be a smooth continuous function, but in practice it is usually sufficient to use a piecewise linear function with no more than four segments. Such a function can be constructed using the thermometer scale idea described previously, but applied to (say) five evenly spaced numerical values for the attribute rather than to policy alternatives directly. For example, one of the other criteria shown in Fig. 1 was "dry season flow" in the river. This was assessed by hydrologists in terms of the percentage reduction in streamflows below current conditions. Over the alternatives under consideration, values for this attribute ranged between 0 and 20% below current levels. The value function was thus approximated by comparing the impacts of five possible levels (0, 5, 10, 15, and 20%) relative to each other on a thermometer scale. The resulting value function could then be represented as in Fig. 3. Once the function has been assessed, the partial value score for any particular alternative is obtained simply by reading off the function value (on a graph such as that illustrated in Fig. 3) corresponding to its attribute value $z_i(a)$.

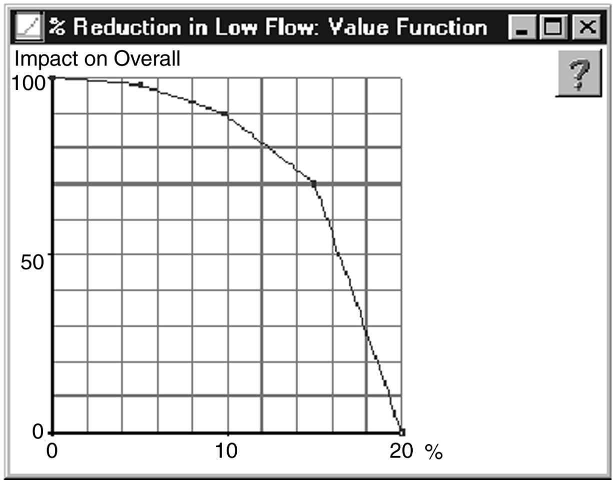


**Figure 3** Illustration of a value function.

It is worth noting the nonlinearity in the shape of the function in Fig. 3. This is quite typical. One of the big dangers in using scoring methods such as those described here is that users and analysts often tend to construct straight-line functions as the easy way out (often even viewing this as the "objective" or "rational" approach). Research has shown clearly that the results obtained from scoring methods can be quite critically dependent upon the shape of the function, so it is incumbent upon users of these tools to apply their minds to the relative value gaps between different levels of performance. Quite frequently, it is found that the functions exhibit systematically increasing or decreasing slopes (as in Fig. 3 where the slopes become increasingly negative) or have an "S" (sigmoidal) shape (or reverse S shape).

Once the partial values have been assessed as above, the weights can also be evaluated. The algebraic implication of Eq. (1) is that the weights determine the desirable tradeoffs between the partial value scores for the different criteria, and for this reason it is important to delay assessment of weights until the people involved in the assessment have established a clear understanding of the ranges of outcomes relevant to each criterion. Various procedures have been suggested for the weight assessment, but one of the simplest and easiest to apply is that of "swing weighting." The users are presented with a hypothetical scenario in which all criteria have the same score on the partial value function scales. Often the 0 point is suggested in the literature, but in our experience people find it easier to start from a less unrealistically extreme position, for example, one in which all partial values are 50. The question is then posed: "If you could choose one and only one criterion to swing up to the maximum partial value score of 100, which one would it be?" This establishes the criterion having the largest weight $w_i$ in Eq. (1). The question is then repeated, excluding the previously chosen criterion, to establish the second largest weight, and so on. Once we have the rank ordering of the weights in this way, we can compare each criterion with the one known to have the maximum weight, and we can then pose the second question: "What is the value of the swing on this criterion, relative to that for the criterion with maximum weight, expressed as a percentage?" In some software, the presentation of this question is facilitated by use of bar graphs, with the heights of the bars representing the relative importance. This gives relative values for the weights, which are usually then standardized in some convenient manner, e.g., so that the weights sum to 1.

The above methodology for fitting a preference model of the form given by Eq. (1) is based on what is sometimes termed "SMART" (simple multiattribute rating technique). An alternative and apparently widely used approach to fitting the same type of model is the technique termed the *analytic hierarchy process* (AHP). This approach is based on first assessing the scores by pairwise comparison of the alternatives on each criterion. In other words, each alternative is compared with every other alternative in terms of the relative importance of its contribution to the criterion under consideration. The comparisons are expressed in ratio terms, interpreted as estimates of $v_i(a)/v_i(b)$ for the pair of alternatives *a* and *b*, and these ratios are used to derive the individual scores. This process is repeated for each criterion. The weights $w_i$ are assessed in the same way by pairwise comparisons of criteria, structured hierarchically (i.e., criteria at one level in the hierarchy are compared only with others sharing the same parent criterion at the next hierarchical level).

The AHP approach has appeared to be very popular and is widely described in most management science texts. Reasons for the popularity seem to include the natural language (semantic) scales on which the comparisons may be made and the availability of user friendly supporting software. Nevertheless, the process is for most users somewhat of a black box, which in the view of the author may hinder rather than facilitate good decision-making practice. A number of theoretical objections to the validity of the process have also been raised, and references to discussions of these are provided in Belton and Stewart (2002). It should, however, also be noted that a number of modifications to the basic AHP approach have been proposed, aimed at circumventing the more critical of these theoretical objections.

## C. Goal and Reference Point Approaches

Goal Programming is a separate article in this encyclopedia. It is, however, useful to summarize some key concepts within the broader multicriteria decision-making framework discussed here.

Goal and reference point approaches are used primarily when the criteria are associated with quantifiable attributes $z_i(a)$ and are thus possibly most appropriate to technical phases of analysis (i.e., in order to shortlist alternatives for more detailed evaluation according to qualitative, intangible, and subjective criteria). The principle is quite simple. Instead of evaluating tradeoffs and weights (as in Section IV.B), the user simply specifies some *desirable goals* or *aspirations,* one for each criterion. These aspirations define in a sense a *prima facie* assessment by the user of what would constitute a realistically desirable outcome.

Let $g_i$ be a goal or aspiration level specified for criterion *i.* The interpretation of $g_i$ will depend on the manner in which the corresponding attribute is defined:

- *Maximizing sense:* If the attribute is defined such that larger values of $z_i(a)$ are preferred to smaller values, all other things being equal (typically some form of "benefit" measure), then the implied aim is to achieve $z_i(a) \geq g_i$. Once this value is achieved, further gains in $z_i(a)$ are of relatively much lesser importance.
- *Minimizing sense:* If the attribute is defined such that smaller values of $z_i(a)$ are preferred to larger values, all other things being equal (typically some form of "cost" measure), then the implied aim is to achieve $z_i(a) \leq g_i$. Once this value is achieved, further reductions in $z_i(a)$ are of relatively much lesser importance.

Sometimes planners like to target some form of intermediate desirable value, possibly something like a water temperature which should not be too hot or too cold. In this case, values of $z_i(a)$ in the vicinity of the target value $g_i$ are desirable, with greater deviations on either side to be avoided. Since the reasons for avoiding deviations in each direction will generally be different, it is usually convenient to define two separate criteria ("not too hot" and "not too cold"), each using the same attribute, but with different aspiration levels. For example, if the desired temperature range is 15–18°C, then the goal for the not too cold criterion will be *temperature* ≥15°C, while that for the not too hot criterion will be *temperature* ≤18°C. Thus, for the purposes of further explanation, we shall assume that all attributes will be defined in one of the two senses defined by the above-bulleted items.

The general thrust of the so-called *goal programming* or *reference point* approaches to MCDA is based firstly on defining deviational variables $\delta_i(a)$ corresponding to the performance of each alternative in terms of each criterion, measuring the extent to which the goal is not met by alternative *a*, that is,

$$\delta_i(a) = \max\{0, g_i - z_i(a)\}$$

for attributes defined in a maximizing sense and

$$\delta_i(a) = \max\{0, z_i(a) - g_i\}$$

for attributes defined in a minimizing sense.

Algebraically (for purposes of inclusion in mathematical programming code), the deviational variables may be defined implicitly via constraints of the form:

$$z_i(a) + \delta_i(a) \geq g_i$$

for attributes defined in a maximizing sense and

$$z_i(a) - \delta_i(a) \leq g_i$$

for attributes defined in a minimizing sense, linked to some process which minimizes all deviations as far as is possible.

The key question at this stage relates to what is meant by minimizing all deviations. Very often, a simple and effective approach is simply to choose the alternative for which the sum of (possibly weighted) deviations is minimized. This is the basis of conventional goal programming. Without going into any detailed review at this stage, it is this author's view that a more robust approach is to use the so-called Tchebycheff norm popularized in the approaches termed *reference point techniques.* In essence, we then identify the alternative *a* which minimizes a function of the form

$$\max_{i=1}^{m} [w_i\delta_i(a)] + \epsilon \sum_{i=1}^{m} w_i\delta_i(a) \qquad (2)$$

where $\epsilon$ is a suitably small positive number (typically something like 0.01) and $w_i$ are weights reflecting the relative importance of deviations on each goal. It is important to emphasize that these weights are related to tradeoffs between attributes in the vicinity of the aspiration levels and are dependent upon the specific scale of measurement used. The best way to assess these weights is to evaluate the allowable tradeoffs directly.

The above process can be applied in either the discrete choice or the mathematical programming contexts. For discrete choice, the calculations for each alternative are easily set up in a spreadsheet. For example, suppose that we are evaluating six policy alternatives in a regional water planning context and that four critical criteria have been identified, associated with the four quantitative attributes: investment cost ($m), water quality (ppm of contaminant), minimum flow levels in the river (m$^3$/sec), and recreational access (thousands of person days per annum). Suppose that the values of these criteria for the six alternatives are as follows:

| Alternative | Costs ($m) | Quality (ppm) | Minimum flow (m$^3$/sec) | Recreational access (person days) |
|---|---|---|---|---|
| Scenario A | 93 | 455 | 1.8 | 160 |
| Scenario B | 127 | 395 | 1.9 | 190 |
| Scenario C | 88 | 448 | 1.5 | 185 |
| Scenario D | 155 | 200 | 2.5 | 210 |
| Scenario E | 182 | 158 | 3.1 | 255 |
| Scenario F | 104 | 305 | 1.7 | 220 |

Note that the first two attributes require minimization and the latter two attributes require maximization. Suppose that goals are specified as follows: $120m for cost, 280 ppm for quality, 2.5 m$^3$/sec for minimum flow, and 225 person days for recreational access. The unweighted deviations ($\delta_i$(a)) can be computed as follows:

| Alternative | Costs | Quality | Minimum flow | Recreational access |
|---|---|---|---|---|
| Scenario A | 0 | 175 | 0.7 | 65 |
| Scenario B | 7 | 115 | 0.6 | 35 |
| Scenario C | 0 | 168 | 1 | 40 |
| Scenario D | 35 | 0 | 0 | 15 |
| Scenario E | 62 | 0 | 0 | 0 |
| Scenario F | 0 | 25 | 0.8 | 5 |

Suppose that the following tradeoffs have been assessed as follows: a reduction of 0.1 m$^3$/sec in the minimum flow would be equivalent in importance to changes of $4m in costs, 10 ppm in contaminants, and 10,000 person days for recreational access. Arbitrarily setting $w_3 = 1$ (numerical weight for the minimum flow criterion), these tradeoffs translate into the following weights for the other criteria: $w_1 = 0.025$ (costs), $w_2 = 0.01$ (quality), and $w_4 = 0.01$. Using these weights and $\epsilon = 0.01$, we obtain the following values of the function given by Eq. (2) for each of the alternatives:

| | |
|---|---|
| Scenario A | 1.781 |
| Scenario B | 1.173 |
| Scenario C | 1.711 |
| Scenario D | 0.885 |
| Scenario E | 1.566 |
| Scenario F | 0.811 |

Scenario F is then indicated as the best compromise, followed closely by scenario D. The remainder are shown to be considerably worse in the sense of having large deviations for one or more criteria.

For a small number of alternatives, as in the above example, the goal programming or reference point approach does not generate too much insight. The methods come much more into their own, however, when there are a large number of alternatives that have to be screened and especially when the problem has a mathematical programming structure. In the linear programming case, the trick is to minimize a new variable $D$, subject to the constraints $D \geq w_i \delta_i(a)$, to the constraints described above for implicitly defining the deviational variables and to the natural constraints of the problem. The proper setting up of the problem for solution would generally require the assistance of a specialist skilled in (multiobjective) linear programming, and we shall not attempt to provide all the details here.

## D. Outranking Approaches

In essence, the outranking approach attempts to characterize the evidence for and against assertions such as "alternative *a* is at least as good as alternative *b*," rather than to establish any form of optimal selection per se. Initially, alternatives are compared in terms of each criterion separately, much as in value function approaches. The tendency is to make use of attribute measures [which we have previously termed $z_i(a)$] to facilitate this comparison, although these attributes may be expressed on some form of nominal scale. The attribute values tend to be used in a relatively "fuzzy" sense, however, so that (for example) alternative *a* will only be inferred as definitely preferred to *b* if the difference $z_i(a) - z_i(b)$ exceeds some threshold level.

In determining whether alternative *a* can be said to be "at least as good as" alternative *b*, taking all criteria into account, two issues are taken into consideration:

1. Which criteria are *concordant* with the assertion? A measure of *concordance* is typically defined as the sum of weights associated with those criteria for which *a* is distinctly better than *b*, when the weights are standardized to sum to 1. It must be emphasized that the weights have a very different meaning to the tradeoff interpretation described for the other two schools of MCDA. For outranking, the weights may best be seen as a "voting power" allocated to each criterion, representing in an intuitive sense the power to influence outcomes that should be vested in each criterion.
2. Which criteria are strongly *discordant* with the assertion, to the extent that they could "veto" any consensus? A measure of *discordance* for attributes defined in a maximizing sense is typically defined by the magnitude of $z_i(b) - z_i(a)$ (since by assumption $z_i(a) \leq z_i(b)$ for discordant attributes, when attributes are defined in a maximizing sense) relative to some predefined norm. The overall measure of discordance is then the maximum of the individual measures for each discordant criterion.

In order to illustrate the concordance and discordance principles, consider the hypothetical comparison of two locations for a large new reservoir in an environmentally sensitive area which also contains a number of villages. Suppose that the options are to be compared in terms of four criteria: cost (in $m), number of people displaced, area of sensitive ecosystems destroyed (in thousands of acres), and impact on aquatic life (measured on a 0–10 nominal scale, where 0 implies no impact which is ecologically most desirable). Suppose assessments for the two locations have been made as follows:

| | **Cost ($m)** | **Number displaced** | **Area lost ('000 acres)** | **Ecological impact** |
|---|---|---|---|---|
| Location A | 18 | 200 | 30 | 7 |
| Location B | 25 | 450 | 5 | 4 |
| Criterion weight | 0.35 | 0.25 | 0.25 | 0.15 |
| Norm for assessing discordance | 10 | 350 | 30 | 9 |

Location A is better than location B on cost and number displaced, and thus the concordance index for A versus B is 0.35 + 0.25 = 0.6. Correspondingly, the concordance for B versus A is 0.4.

The discordant criteria for A compared to B are area lost and ecological impact, with relative magnitudes 25/30 = 0.83 and 3/9 = 0.33, respectively, so that the overall measure of discordance is 0.83. Similarly, the measure of discordance for B compared to A is the maximum of 0.7 and 0.71, i.e., 0.71.

The methods based on outranking principles compare all pairs of available alternatives in the above manner. Any one alternative *a* is said to outrank *b* if the concordance is sufficiently high and the discordance is sufficiently low. In some implementations, the outranking is viewed as "crisp," i.e., an alternative either does or does not outrank another, with the decision being based on whether the concordance exceeds a predefined minimum level and the discordance does not exceed a predefined maximum level. In other implementations a fuzzy degree of concordance is constructed from the concordance and discordance measures. In either sense, the result is a measure of the extent to which the evidence favors one alternative over another. This could lead to elimination of some alternatives and/or the construction of a short list of alternatives for deeper evaluation.

The techniques by which outranking methods establish partial or tentative rank orders of the alternatives are technically very complicated and beyond the scope of this article. Some details may be found in the books of Roy (1996) and of Belton and Stewart (2002).

Outranking methods are relevant to situations in which (1) there are a discrete number of alternatives under consideration and (2) preference information such as detailed value trade-offs are not easily available (typically because the analysis is being carried out by expert groups on behalf of political decision makers who have been unwilling or unable to provide the sort of information required by the other two schools of MCDA).

## V. RISK AND UNCERTAINTY IN DECISION MAKING

All nontrivial decision making has to contend in some way with issues of risk and uncertainty. The future is always unknown, and consequences of decisions can never be predicted with certainty. Thus, even with the most careful and detailed analysis, perhaps as described in the previous section, the unexpected has to be expected! We thus conclude this article with a brief review of approaches that can be used in dealing with risk and uncertainty in decision making.

### A. Informal Sensitivity and Robustness Analysis

One approach is simply to subject the results of decision analysis, such as that described in Section IV, to intensive sensitivity analysis. Each of the input assumptions will be critically evaluated in order to establish a plausible range of values or outcomes (recognizing, however, the dangers inherent in the anchoring and adjustment biases described in Section III.C, which may lead to underestimation of the range). The analysis may then be repeated for different assumptions across these ranges. The aim ultimately is to identify the course of action which is most robust in the sense of performing well over all plausible ranges of inputs, rather than that which optimizes performance for a single set of assumptions or inputs.

Such sensitivity and robustness analysis is always to be recommended as part of the decision-making process. It must be realized, however, that this is not the panacea for all problems of risk and uncertainty. One of the difficulties is that the sensitivity analysis tends to have to proceed in a fairly *ad hoc* fashion by changing one or two input parameters at a time. In complex systems, results may be insensitive to changes in single inputs, but substantially more sensitive to certain combinations of inputs, and it is in general very difficult to identify these critical combinations. Some of the techniques from the soft-systems methodology of Checkland (1981) can be of value in this regard.

### B. Statistical Decision Theory

A separate article deals with decision theory, and the reader is referred to that for more details. In essence, however, statistical decision theory proceeds by establishing probability distributions on outcomes, using both subjective information and available data. However, we need to recall again the potential biases inherent in subjective probability assessments, as discussed in Section III. Decision-making values and goals are then captured in some form of utility function, representing the desirability of different outcomes. With this information, we can, in principle, identify the course of action which maximizes expected utility.

This is an approach which falls very much into the category of rational approaches. While decision theory can provide useful insights (as much through the construction of probabilities and utilities as through the subsequent analysis), it needs to be used with caution. Some of the underlying theoretical assumptions have been challenged; consequently, the very existence of a "utility function" which can be assessed separately from the probabilities has been questioned. Furthermore, the extension of the utility function concept to the multicriteria type of problem discussed in Section IV raises a number of practical difficulties. Stronger assumptions are required in this case, which are even more difficult to verify, and the construction of the resultant utility models requires judgmentally rather demanding inputs from the decision maker.

## C. Scenario Planning

The concepts around the use of scenarios to represent future structural uncertainties while considering strategic planning options appear to have been developed within the Royal Dutch/Shell corporation and have been thoroughly documented by van der Heijden (1996). The idea is to construct, through an intensive brainstorming session, a small number of future scenarios. These are meant to be internally consistent descriptions of possible futures, describing a trajectory of changes in future conditions that band together in a coherent manner. Typically, a relatively small number (3–5) of detailed scenarios will be constructed, as it is important for decision makers to be able to compare decision alternatives across different scenarios, which becomes well nigh impossible if the number of scenarios exceeds the "magic number 7."

Scenario planning is similar to sensitivity analysis in the sense that there is value in identifying courses of action which are robust across all scenarios. The approach is, however, much more structured, and greater attention is paid to relationships between variables and to avoidance of biases such as anchoring and adjustment. To be done properly, scenario planning requires a considerable investment in time by senior management, but this is well justified for strategic decision making with far-reaching consequences.

## D. Risk as a Criterion

In some cases, it is possible to include avoidance of risk as a criterion and to handle it in the same way as the other criteria discussed in Section IV. This is, for example, routinely done in much of the portfolio investment analysis theory, where expectation and standard deviation of returns may be viewed as distinct decision criteria. In this case, the expectation is a measure of return under standard or expected conditions, while the standard deviation measures probable range of outcomes, i.e., the risk. What is thus intrinsically a monocriterion decision problem under uncertainty is analyzed as a bicriterion problem in which the uncertainty is subsumed into one of the criteria.

The representation of risk, or risk avoidance, as a criterion allows the powerful tools of MCDA (Section IV) to be applied, which can be done at any level of detail appropriate to the decision context. In some cases, relatively "quick and dirty" tools will suffice if the decision consequences are limited in extent; in other cases, a deep analysis of risk preference is possible. The effective use of MCDA for this purpose is still a matter for future research. It should be noted, however, that measures of risk other than standard deviation (e.g., probabilities of certain undesirable or catastrophic consequences) appear to be more appropriate in some situations. The reader should also be warned that when using value function models for the MCDA, nominal "riskiness" scales (as an alternative to standard deviation) may easily violate the underlying assumptions of preferential independence and of the interval scale property and should be used with considerable caution.

## SEE ALSO THE FOLLOWING ARTICLES

Corporate Planning • Data Mining • Decision Support Systems • Decision Theory • Goal Programming • Strategic Planning for/of Information Systems • Uncertainty

## BIBLIOGRAPHY

Belton, V., and Stewart, T. J. (2002). *Multiple Criteria Decision Analysis: An Integrated Approach,* Kluwer Academic Publishers, Boston, MA.

Checkland, P. (1981). *Systems Thinking, Systems Practice,* Wiley, New York.

Eden, C., and Ackermann, F. (1998). *Making Strategy: The Journey of Strategic Management,* SAGE Publications, London.

Goodwin, P., and Wright, G. (1997). *Decision Analysis for Management Judgement,* 2nd ed., Wiley, New York.

Janis, I. L., and Mann, L. (1977). *Decision Making,* The Free Press, New York.

Kahneman, D., Tversky, A., and Slovic, P. (1982). *Judgement under uncertainty: Heuristics and biases.* Cambridge UP, Cambridge.

Kaplan, R. S., and Norton, D. P. (1996). *The Balanced Scorecard,* Harvard Business School Press, Boston, MA.

Keeney, R. L. (1992). *Value-Focused Thinking: A Path to Creative Decision Making,* Harvard Univ. Press, Cambridge, MA.

Rosenhead, J., and Mingers, J., Eds. (2001). *Rational Analysis for a Problematic World Revisited,* Wiley, New York.

Roy, B. (1996). *Multicriteria Methodology for Decision Aiding,* Kluwer Academic Publishers, Dordrecht/Norwell, MA.

Simon, H. A. (1976). *Administrative Behavior,* 3rd ed., The Free Press, New York.

van der Heijden, K. (1996). *Scenarios: The Art of Strategic Conversation,* Wiley, New York.

Von Winterfeldt, D., and Edwards, W. (1986). *Decision Analysis and Behavioral Research,* Cambridge Univ. Press, Cambridge, UK.

# Decision Support Systems

**Clyde W. Holsapple**
*University of Kentucky*

## GLOSSARY

**artificial intelligence** A field of study and application concerned with identifying and using tools and techniques that allow machines to exhibit behavior that would be considered intelligent if it were observed in humans.

**decision** The choice of one from among a number of alternatives; a piece of knowledge indicating a commitment to some course of action.

**decision making** The activity that culminates in the choice of an alternative; the activity of using knowledge as raw materials in the manufacture of knowledge about what to do.

**decision support system (DSS)** A computer-based system composed of a language system, presentation system, knowledge system, and problem-processing system whose collective purpose is the support of decision-making activities.

**descriptive knowledge** Knowledge about past, present, and hypothetical states of an organization and its environment.

**knowledge-based organization** An organization in which the primary, driving activity is the management of knowledge.

**knowledge management** The activity of representing and processing knowledge.

**knowledge-management technique** A technique for representing knowledge in terms of certain kinds of objects and for processing those objects in various ways.

**knowledge system** That subsystem of a decision support system in which all application-specific knowledge is represented for use by the problem-processing system. This includes knowledge of any or all types (e.g., descriptive, procedural, reasoning) represented in a variety of ways (e.g., as databases, spreadsheets, procedural solvers, rule sets, text, graphs, forms, templates).

**language system** The subsystem of a decision support system that consists of (or characterizes the class of) all acceptable problem statements.

**multiparticipant DSS** A decision support system that supports multiple participants engaged in a decision-making task (or functions as one of the participants).

**presentation system** The component of a DSS that consists of all responses a problem processor can make.

**problem-processing system** That subsystem of a decision support system that accepts problems stated in terms of the language system and draws on the knowledge system in an effort to produce solutions.

**procedural knowledge** Knowledge about how to produce a desired result by carrying out a prescribed series of processing steps.

**reasoning knowledge** Knowledge about what circumstances allow particular conclusions to be considered to be valid.

## I. INTRODUCTION

Building on initial concepts introduced and demonstrated in the 1970s, the field of decision support systems (DSS) has progressed to a stage where these systems are routinely used by decision makers around the world. Organizations expend large sums to ensure that their employees, customers, suppliers, and partners have computer-based systems that provide the knowledge they need to make timely, sound decisions. These DSS have many variants, being implemented for a wide variety of decisional applications, facilitating various aspects of decision-making processes, and utilizing a variety of technologies. They range from systems that support individuals making decisions to multiparticipant DSS, which support the collaboration of multiple individuals in making a joint decision, in making interrelated decisions, or in transorganizational decision making.

An appreciation of the objectives, characteristics, uses, development, and impacts of decision support systems begins with an understanding of *decisions* and decision making. This leads to an examination of *support,* why decision makers need support, and what kinds of decision support could be beneficial. Against this background, consideration of computer-based *systems* that can deliver support for decisions unfolds. Included is a consideration of characteristics that distinguish such systems from other types of business computing systems, an architecture of DSS components, and an examination of various classes of DSS. Classifications are based on technology used in developing the system and on whether the system supports an individual versus a multiparticipant decision maker.

## II. DECISIONS

Immersed in a competitive, knowledge-rich world, managers are daily confronted with the task of making decisions about allocations of their resources, about handling disturbances to their operations and plans, about taking advantage of new opportunities, and about interacting or negotiating with others. Each decision involves the use of knowledge of varying kinds of amounts, and many can benefit from (or even require) the use of technology known as DSS. Similarly, managers in an organization's suppliers, partners, and competitors are faced with their own decision challenges and accompanying knowledge needs; these, too, can benefit from systems that help meet those needs. Moreover, consumers make decisions about products and services to satisfy their preferences. Increasingly, web-oriented decision support systems are available to supply knowledge for consumers' decisional efforts.

The study of DSS has many technical aspects. But before delving into these, it is important to appreciate the setting in which they are used. The setting is a competitive, knowledge-rich world in which managers make decisions about what to do with their organizations' resources. Many decisions, ranging from simple to complex, are made every day. Each decision involves the use of knowledge of varying kinds and amounts, and many can benefit from (or even require) the use of technology known as DSS.

### A. Making a Decision

A classic view among management theorists is that a decision is a choice about a course of action, about a strategy for action, or leading to a desired outcome. The classic view of decision making is that it is an activity culminating in the selection of one from among multiple alternative courses of action.

Decision-making activity identifies alternative courses of action and selects one of them as the decision. The number of alternatives identified and considered could be very large. The work involved in becoming aware of alternatives often makes up a major share of a decision-making episode. It is concerned with such questions as where do alternatives come from, how many alternatives are enough, should more effort be devoted to uncovering alternatives, and how can large numbers of alternatives be managed so none is forgotten or garbled? One role of a DSS is to help decision makers cope with such issues.

Ultimately, one of the alternatives is selected. But, which one? This depends on a study of the alternatives in an effort to understand their implications. The work involved in selecting one of the alternatives usually makes up a major share of a decision-making episode. It is concerned with such questions as to what extent should each alternative be studied, how reliable is our expectation about an alternative's impacts, are an alternative's expected impacts compatible with our purposes, what basis should be used to compare alternatives to each other, and what strategy will be followed in arriving at a choice. Another role of a DSS is to support the study of alternatives. Some DSS may even recommend the selection of a particular alternative and explain the rationale underlying that advice.

Complementing the classic view of decisions, there is the knowledge-based view which holds that a decision is a piece of knowledge indicating the nature of an action commitment. A decision could be a piece of descriptive knowledge. For instance, "spend $10,000 on advertising in the next quarter" describes a commitment of what to do about advertising expenditures. This decision is one of many alternative descriptions (e.g., spend $5000) that could have been chosen. A decision could be a piece of procedural knowledge, involving a step-by-step specification of how to accomplish something. For instance, "determine the country with the most favorable tax structure, identify the sites within that country having sufficient qualified work forces, then visit those sites to assess their respective qualities of life, and, from among those that are acceptable, locate the new factory at the site with the best transportation infrastructure" is a chunk of procedural knowledge committing an organization to a certain sequence of actions. It is one of many alternative procedures that could have been chosen.

When we regard a decision as a piece of knowledge, making a decision means we are making a new piece of knowledge that did not exist before. We are manufacturing new knowledge by transforming or assembling existing pieces of knowledge. A DSS is a system that aids the manufacturing process, just as machines aid in the manufacturing of material goods. Not only is there the new piece of knowledge called a decision, but the manufacturing process itself may have resulted in additional new knowledge as byproducts. For instance, in manufacturing a decision, we may have derived other knowledge as evidence to justify our decision. We may have produced knowledge about alternatives that were not chosen, including expectations about their possible impacts. More fundamentally, we may have developed knowledge about improving the decision manufacturing process itself. Such by-products can be useful later in making other decisions.

## 1. Knowledge in Decision Making

A decision maker possesses a storehouse of knowledge, plus abilities to both alter and draw on the contents of that storehouse. This characterization holds for all types of decision makers—individuals, teams, groups, and organizations. In the multiparticipant cases, both the knowledge and the abilities are distributed among participants. When using a DSS, a decision maker's storehouse is augmented by computer-based representation of knowledge and the decision maker's ability to process knowledge is supplemented by the DSS's ability to process those representations. From a decision-oriented perspective, three primary types of knowledge have been identified: descriptive, procedural, and reasoning. Each of these can exist in explicit or tacit modes within a decision maker. Each can be explicitly represented in and processed by a DSS using a variety of computer-based techniques.

Knowledge about the state of some world is called descriptive knowledge. Commonly referred to as data or information, it includes descriptions of past, present, future, and hypothetical situations. A decision maker can acquire descriptive knowledge via observation and can produce it by transforming or assembling existing pieces of knowledge. Knowledge about how to do something is quite different from knowledge of a state. Because it is concerned with step-by-step procedure for accomplishing some task, it is called procedural knowledge. As a decision maker comes into possession of more or better procedural knowledge, we say that decision maker is more skilled.

Reasoning knowledge specifies the conclusion that can be drawn when a specified situation exists. A code of conduct, a set of regulations, a customer service policy, rules that prescribe forecasting approaches, and rules used to diagnose causes of situations are all examples of reasoning knowledge. Whereas procedural knowledge is "know how" and descriptive knowledge is "know what," reasoning knowledge is "know why." By putting together pieces of reasoning knowledge, we can reach logical conclusions and justify them by citing other reasons. This activity is known as drawing inferences. The reasoning knowledge that fuels an inference may be acquired or derived by a decision maker. Either way, as a decision maker comes to possess more or better knowledge, we say that decision maker is more of an expert.

The raw materials that can go into a decision-making process are pieces of descriptive, procedural, and reasoning knowledge. These are common ingredients in decision-making recipes. During the process, varying amounts of descriptive, procedural, and reasoning knowledge may be added at different times in different combinations. That is, pieces of different types of knowledge can be made to interact, and the value of one piece may depend on having another available at the proper time. DSSs can store and use these types of knowledge to supply it when needed in a decision process or produce new knowledge for the decision process.

## 2. Structured versus Unstructured Decisions

When issues relevant to making a decision are well understood, the decision tends to be structured. The alternatives from which the choice is made are clear-cut, and each can be readily evaluated in light of the organization's purposes and goals. Put another way, all the knowledge required to make the decision is available in a form that makes it straightforward to use. Often times, however, the issues pertinent to producing a decision are not well understood. Some issues may be entirely unknown to the decision maker, which is a hallmark of unstructured decisions. The alternatives from which a choice will be made are vague, are difficult to compare and contrast, or cannot be easily evaluated with respect to the organization's purposes and goals. It may even be that there is great difficulty in attempting to discover what the alternatives are. In other words, the knowledge required to produce a decision is unavailable, difficult to acquire, incomplete, suspect, or in a form that cannot be readily used by the decision maker. A semistructured decision lies between the two extremes: some aspects of the decision-manufacturing activity may be structured, whereas others are not. Decision support systems can be developed to assist in structured, semistructured, or unstructured situations.

Consider the structured decision of selecting a travel plan for making a regular monthly inspection visit to a major supplier's factory. From destination, duration, allowable dates to travel, budget limits, traveler preferences, and prior travel plans, the parameters for the decision are well known. All that is missing for deciding on a satisfactory travel plan is a characterization of alternatives that are available for the upcoming trip (e.g., costs, times, amenities). Characterizations of these alternatives could be found and presented by a DSS each time such a decision is to be made; the DSS might even rank the alternatives based on known criteria. As part of making a semistructured decision about amounts of a part to order from various suppliers, a DSS might solve such problems as estimating demand or deriving an "optimal" allocation scheme. The decision maker uses such solutions along with other knowledge (e.g., supplier dependability in part quality and delivery times, importance of cultivating ongoing supplier relationships, impacts on orders of other parts from the same suppliers) in reaching the decision. In the course of making an unstructured decision about how to react to a revolutionary technological or competitive change that may impact the viability of a current product offering, a DSS may be useful in "what if" analysis that shows impacts of alternative courses of action or a DSS may help explore internal and external knowledge sources in order to stimulate or provoke insights about coping with the unprecedented situation.

## 3. Decision-Making Phases and Problem Solving Flows

Three decision-making phases are widely recognized: intelligence, design, and choice. Each one is susceptible to computer-based support. The intelligence phase is a period when the decision maker is alert for occasions to make decisions, preoccupied with collecting knowledge, and concerned with evaluating it in light of the organization's purpose. The design phase is a period when the decision maker formulates alternative courses of action, analyzes those alternatives to arrive at expectations about the likely outcomes of choosing each, and evaluates those expectations with respect to the organization's purpose. During the design phase, the decision maker could find that additional knowledge is needed. This would trigger a return to the intelligence phase to satisfy that need before continuing with the design activity.

In a choice phase, the decision maker exercises authority to select an alternative. This is done in the face of internal and external pressures related to the nature of the decision maker and the decision context. It can happen that none of the alternatives are palatable, that several competing alternatives yield very positive evaluations, or that the state of the world has changed significantly since the alternatives were formulated and analyzed. Nevertheless, there comes a time when one that is "good enough" or "best" must be selected. If that time has not yet been reached, the decision maker may return to one of the two earlier phases to collect more up-to-date knowledge, formulate new alternatives, reanalyze alternatives, reevaluate them, etc.

Within each phase of a decision-making process, the decision maker initiates various subactivities. Each of these activities is intended to solve some problem. The decision maker might need to solve such problems as acquiring a competitor's sales figures, predicting the demand for a product, assessing the benefits and costs of a new law, inventing a feasible way of packaging a product into a smaller box, or finding out the cultural difficulties of attempting to market a certain product in foreign countries. The overall task of reaching a decision is a superproblem. Only if we solve subproblems can we solve the overall decision problem.

A decision-making process is fundamentally one of both recognizing and solving problems along the way toward the objective of producing a decision. For structured decisions, the path toward the objective is well charted. The problems to be surmounted are recognized easily, and the means for solving them are readily available. Unstructured decisions take us into uncharted territory. The problems that will be encountered along the way are not known in advance. Even when stumbled across, they may be difficult to recognize and subsequently solve. Ingenuity and an exploratory attitude are vital for coping with these types of decisions.

Decision support systems are developed to facilitate the recognizing and/or solving of problems within a decision-making process. In the case of a multiparticipant decision maker, they can assist in communications and coordinating the problem-solving flows among participants working on various problems simultaneously, in parallel, or in some necessary sequence.

## B. The Need for Support

Computer systems to support decision makers are not free. Not only is there the cost of purchasing or developing a DSS, there are also costs associated with learning about, using, and maintaining a DSS. It is only reasonable that the benefits of a DSS should be required to outweigh its costs. Although some DSS benefits can be difficult to measure in precise quantitative terms, all the benefits are the result of a decision maker's need for support. When a decision maker needs support it is because of cognitive, economic, or time limits, or because of competitive pressures.

Cognitive limits refer to limits in the human mind's ability to store and process knowledge. Because decision making is a knowledge-intensive activity, cognitive limits substantially restrict an individual's decision-making efficiency and effectiveness. If these limits are relaxed, decision-maker productivity can improve. A DSS serves as an extension to a person's innate knowledge-handling skills, allowing problems to be solved more reliably or rapidly.

To relax cognitive limits as much as possible, we could consider forming a very large team. But as a team incorporates more and more participants, the proportion of activity spent in solving communication and coordination problems rises relative to the problem solving directly concerned with making the decision. Thus increasing a team's size runs into economic limits not only in terms of paying and equipping more participants, but also with respect to increased communication and coordination costs. Decision support systems can be a less expensive alternative by substituting for participants in performing knowledge handling tasks or by facilitating the communication and coordination among the participants in a decision process.

A third limit that decision makers commonly encounter is a time limit. A decision maker may be blessed with extraordinary cognitive abilities and vast monetary resources but very little time. Time limits can put severe pressure on the decision maker, increasing the likelihood of errors and poor-quality decisions. There may not be sufficient time to consider relevant knowledge, to solve relevant problems, or to employ a desirable decision-making strategy. Because computers can process some kinds of knowledge much faster than humans, are not error-prone, work tirelessly, and are immune to stresses from looming deadlines, DSSs can help lessen the impacts of time limits.

Aside from relaxing limits on a decision maker, DSSs are needed for another important reason. Decision makers and organizations often find themselves in situations where their continued success—or even their outright survival—depends on being competitive. If one competitor successfully uses DSSs for better decision making and another does not, then the second competitor will be at a competitive disadvantage. To keep pace, it is prudent to consider using DSSs internally and providing them to customers, suppliers, and partners. Beyond this, some organizations actively seek out opportunities for using DSSs in innovative ways in order to achieve competitive advantages.

To summarize, the nature of support a DSS can offer to its user will normally include at least one of the following:

1. It alerts the user to a decision-making opportunity or challenge.
2. It recognizes problems that need to be solved as part of the decision-making process.
3. It solves problems recognized by itself or by the user.
4. It facilitates or extends the user's ability to process (e.g., acquire, transform, explore) knowledge.
5. It offers advice, expectations, evaluations, facts, analyses, or designs to the user.
6. It stimulates the user's perception, imagination, or creative insight.
7. It coordinates or facilitates interactions among participants in multiparticipant decision makers.

# III. DECISION SUPPORT SYSTEM FUNDAMENTALS

One purpose of a DSS is to help problem-solving flows go more smoothly or rapidly: stimulating the user to perceive problems needing to be solved, breaking problems posed by the user into subproblems, actually solving problems posed by a user, and possibly combining and synthesizing solutions of subproblems into the solution of a larger problem. Traditional DSS definitions suggest that the purpose of a DSS is to aid decision makers in addressing unstructured or semistructured decisions. However, some DSSs are used to help with structured decisions by handling large volumes of knowledge or solving complex subproblems more rapidly and reliably than humans. Nevertheless, the DSS emphasis is definitely on supporting decisions that its users regard as less than fully structured. Ultimately, the purpose of a DSS is to help a decision maker manage knowledge. A DSS accepts, stores, uses, derives, and presents knowledge pertinent to the decisions being made. Its capabilities are defined by the types of knowledge with which it can work, the ways in which it can represent these various types of knowledge, and its skills in processing these representations.

## A. DSS Forerunners

One way to appreciate the characteristics of a DSS is to compare and contrast them with traits of two other major types of business computing systems: data processing systems and management information systems (MIS). Both predate the advent of computer-based decision support systems. All three share the trait of being concerned with record keeping. On the other hand, the three kinds of business computing systems differ in various ways, because each serves a different purpose in the management of an organization's knowledge resources

In the 1950s and 1960s, data processing (DP) systems dominated the field of business computing. Their main purpose was and is to automate the handling of large numbers of transactions. At the heart of a DP system lies a body of descriptive knowledge (i.e., data), which is a computerized record of what is known as a result of various transactions having happened. In addition, a DP system endows the computer with two major abilities related to this stored data: record keeping and transaction generation. The first enables the computer to keep the records up to date in light of incoming transactions. The second ability is concerned with the computerized production of outgoing transactions based on the stored descriptive knowledge, transmitted to such targets as customers, suppliers, employees, or governmental regulators. Administrators of a DP system are responsible for seeing that record keeping and transaction generation abilities are activated at proper times.

Unlike a DP system, the central purpose of MIS was and is to provide managers with periodic reports that recap certain predetermined aspects of an organization's past operations. Giving managers regular snapshots of what has been happening in the organization helps them in controlling their operations. Whereas DP is concerned with transforming transactions into records and generating transactions from records, the MIS concern with record keeping focuses on using this stored descriptive knowledge as a base for generating recurring standard reports. An MIS department typically is responsible for development, operation, and administration of DP systems and the MIS.

Information contained in standard reports from an MIS certainly can be factored into decision-making activities. When this is the case, an MIS could be fairly regarded as a kind of DSS. However, the nature of support it provides is very limited due to several factors: its reports are predefined, they tend to be issued periodically, and they are based only on descriptive knowledge. The situation surrounding a decision maker can be very dynamic. Except for the most structured kinds of decisions, information needs can arise unexpectedly and change more rapidly than an MIS can be built or revised by the MIS department.

Even when some needed information exists in a stack of reports accumulated from an MIS, it may be buried within other information held by a report, scattered across several reports, not presented in a fashion that is most helpful to the decision maker, or in need of further processing. Report generation by an MIS typically follows a set schedule. However, decisions that are not fully structured tend to be required at irregular intervals or unanticipated times. Knowledge needed for these decisions should be available on an ad hoc, spur-of-the-moment, basis. Another limit on an MIS's ability to support decisions stems from its exclusive focus on managing descriptive knowledge. Decision makers frequently need to manage procedural and/or reasoning knowledge as well. They need to integrate the use of these kinds of knowledge with ordinary descriptive knowledge.

## B. DSS Traits and Benefits

Ideally, a decision maker should have immediate, focused, clear access to whatever knowledge is needed on the spur of the moment in coping with semistructured or unstructured decisions. The pursuit of this ideal separates DSS from their DP and MIS ancestors and suggests traits we might expect to observe in a DSS:

1. A DSS includes a body of knowledge that describes some aspects of the decision-maker's world, may specify how to accomplish various tasks, and may indicate what conclusions are valid in various circumstances.

2. A DSS has an ability to acquire and maintain descriptive knowledge and possibly other kinds of knowledge as well.
3. A DSS has an ability to present knowledge on an ad hoc basis in various customized ways as well as in standard reports.
4. A DSS has an ability to select any desired subset of stored knowledge for either presentation or for deriving new knowledge in the course of problem recognition and/or problem solving.
5. A DSS can interact directly with a decision maker or a participant in a decision maker in such a way that the user has flexibility in choosing and sequencing knowledge management activities.

These traits combine to amplify a decision maker's knowledge-management capabilities and loosen cognitive, temporal, and economic constraints.

The notion of DSSs arose in the early 1970s. Within a decade, each of the traits had been identified as important and various DSSs were proposed or implemented for specific decision-making applications. By the late 1970s new technological developments were emerging that proved to have a tremendous impact on the DSS field and the popularization of DSSs in the 1980s and beyond: microcomputers, electronic spreadsheets, management science packages, and ad hoc query interfaces.

Specific benefits realized from a particular DSS depend on the nature of the decision maker and the decision situation. Potential kinds of DSS benefits include the following:

1. In a most fundamental sense, a DSS augments the decision maker's own innate knowledge management abilities. It effectively extends the decision maker's capacity for representing and processing knowledge in the course of manufacturing decisions.
2. A decision maker can have the DSS solve problems that the decision maker alone would not even attempt or that would consume a great deal of time due to their complexity and magnitude.
3. Even for relatively simple or structured problems encountered in decision making, a DSS may be able to reach solutions faster and/or more reliably than the decision maker.
4. Even though a DSS may be unable to solve a problem facing the decision maker, it can be used to stimulate the decision maker's thoughts about the problem. For instance, the decision maker may use the DSS for exploratory browsing, hypothetical analysis, or getting advice about dealing with the problem.
5. The very activity of constructing a DSS may reveal new ways of thinking about the decision domain or even partially formalize various aspects of decision making.
6. A DSS may provide additional compelling evidence to justify a decision-maker's position, helping the decision maker secure agreement or cooperation of others. Similarly, a DSS may be used by the decision maker to check on or confirm the results of problems solved independently of the DSS.
7. Due to the enhanced productivity, agility, or innovation a DSS fosters within an organization, it may contribute to an organization's competitiveness.

Because no one DSS provides all these benefits to all decision makers in all decision situations, there are frequently many DSSs within an organization helping to manage its knowledge resources. A particular decision maker may make use of several DSSs within a single decision-making episode or across different decision-making situations.

## C. The Generic Architecture

Generally, DSS can be defined in terms of four essential aspects: a language system (LS), a presentation system (PS), a knowledge system (KS), and a problem-processing system (PPS). The first three are systems of representation. An LS consists of all messages the DSS can accept. A PS consists of all messages the DSS can emit. A KS consists of all knowledge the DSS has stored and retained. By themselves, these three kinds of systems can do nothing. They simply represent knowledge, either in the sense of messages that can be passed or representations that have been accumulated for possible processing. These representations are used by the fourth element: the PPS, which is the active part of a DSS, the DSS's software engine. As its name suggests, a PPS is what tries to recognize and solve problems during the making of a decision.

Figure 1 illustrates how the four subsystems of a DSS are related to each other and to a DSS user. Using its knowledge-acquisition ability, a PPS acquires knowledge about what a user wants the DSS to do or what is happening in the surrounding world. Such knowledge is carried in LS messages that serve as user requests or system observations. The PPS may draw on KS contents when using its acquisition ability. The knowledge-acquisition in the KS or an interpreted message can cause the PPS's other abilities to spring into action. When a user's request is for the solution to some problem, the knowledge-selection/derivation

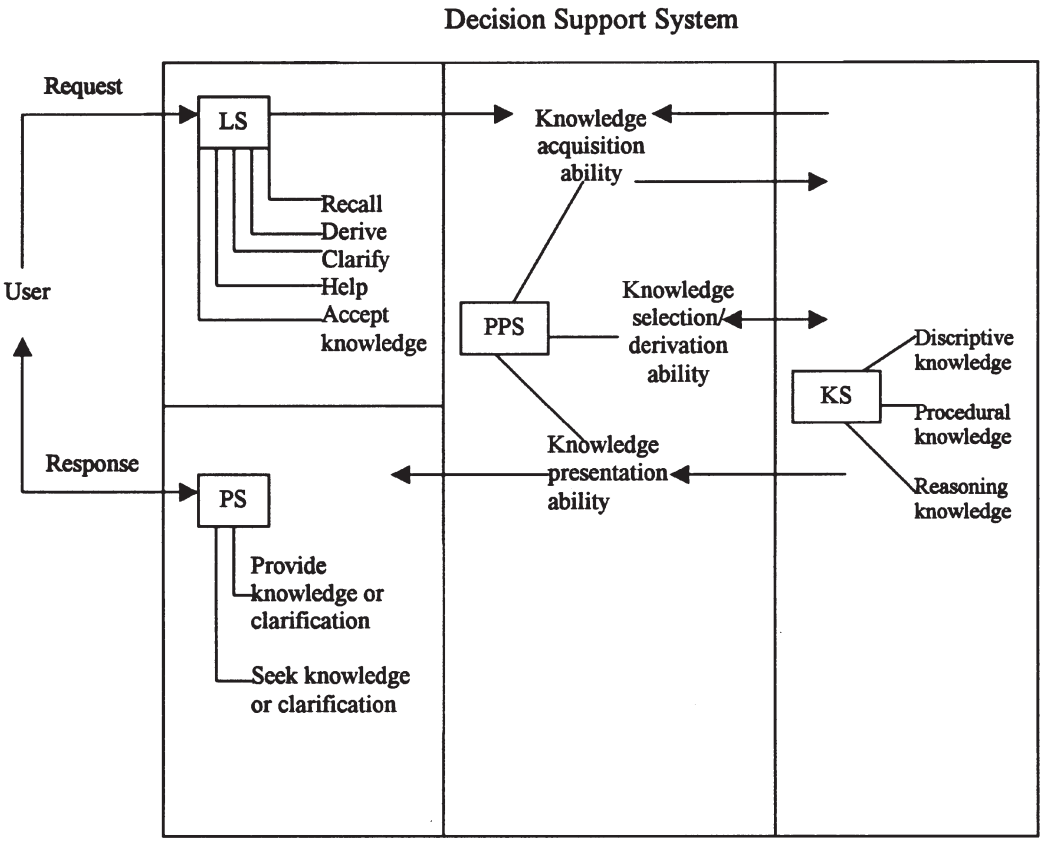

Decision Support System
Request
LS
User
Response
PS
Recall
Derive
Clarify
Help
Accept knowledge
Provide knowledge or clarification
Seek knowledge or clarification
Knowledge acquisition ability
PPS
Knowledge selection/ derivation ability
Knowledge presentation ability
KS
Discriptive knowledge
Procedural knowledge
Reasoning knowledge

**Figure 1** A generic framework of decision support systems.

ability comes into play. The PPS selectively recalls or derives knowledge that forms a solution. When a user's request is for clarification of a prior response or for help in stating a request, the selection/derivation ability may or may not be exercised, depending on whether it needs the KS to produce the content of its response. The PPS can issue a response to the user, by choosing to present one of the PS elements. The presentation choice is determined by the processing, often drawing on KS contents.

This simple architecture captures the fundamental aspects common to all DSSs. To fully appreciate the nature of any particular DSS, one must know about requests that make up its LS, responses that make up its PS, knowledge representations allowed or existing in its KS, and knowledge-processing capabilities of its PPS.

## D. Tools for Developing DSSs

Development tools are essential for building DSSs. The tools chosen for developing a particular DSS strongly influence not only the development process, but also the features that the resultant DSS can offer to a user. A particular tool is oriented toward one or more knowledge-management techniques (e.g., text, spreadsheet, database, solver, or rule management). Conversely, a particular technique (in its many possible variants) is offered by more than one development tool. Thus, tools can be categorized in terms of the knowledge-management techniques they furnish. A spreadsheet tool offers some variant of the spreadsheet technique for knowledge management, a database tool provides some variant of a database technique for managing knowledge, etc. Although many tools tend to emphasize one technique or another, vestiges of additional techniques are often apparent. Some tools furnish healthy doses of multiple techniques.

Tools can play different roles in a DSS development process. An intrinsic tool (e.g., Microsoft's Excel software) serves as the PPS of the developed DSS and tends to furnish a ready-made LS and PS. With such a tool, DSS development becomes a matter of populating the KS with representations that the tool is able to process. Because they do not require the programming of a PPS, intrinsic tools are widely used by nontechnical persons to build their own DSSs. A partially intrinsic tool furnishes part of the DSS's problem processor. The database control system used to operate on database repositories is an example of such a tool. An extrinsic tool does not participate in a PPS, but helps the developer produce all or part of the PPS or to create some portion of the KS contents. Tools in these two categories are of interest primarily to experienced or professional developers.

Another angle from which to examine development tools involves the types of integration they permit within DSSs. This approach is relevant whenever multiple knowledge-management techniques are employed within the bounds of a single DSS. These techniques may be integrated within a single tool or across multiple tools. In the former case, nested and synergistic integration are distinct possibilities. In the latter case, integration can be via direct format conversion, clipboard, or common format approaches.

## IV. CLASSIFICATION OF DECISION SUPPORT SYSTEMS

Decision support systems can be classified in terms of the knowledge-management techniques used to develop them. In many cases, the focus is on a single technique, but compound DSSs employing multiple techniques are also common. A different kind of classification distinguishes DSSs that incorporate artificial intelligence methods from those that do not. Yet another is concerned with DSSs for multiparticipant decision makers.

### A. Technique-Oriented Classes

One way of looking at KS contents and PPS abilities is in terms of the knowledge-management techniques employed by a DSS. This gives rise to many special cases of the generic DSS architecture, each characterizing a certain class of DSSs by restricting KS contents to representations allowed by a certain knowledge-management technique and restricting the PPS abilities to processing allowed by that technique. The result is a class of DSSs with the generic traits suggested in Fig. 1, but specializing in a particular technique for representing and processing knowledge.

#### 1. Text-Oriented DSSs

For centuries, decision makers have used the contents of books, periodicals, and other textual repositories of knowledge as raw materials in the making of decisions. The knowledge embodied in text might be descriptive, such as a record of the effects of similar decision alternatives chosen in the past or a description of an organization's business activities. It could be procedural knowledge, such as a passage explaining how to calculate a forecast or how to acquire some needed knowledge. The text could embody reasoning knowledge, such as rules indicating likely causes of or remedies for an unwanted situation. Whatever its type, the decision maker searches and selects pieces of text to become more knowledgeable, to verify impressions, or to stimulate ideas.

By the 1980s, text management had emerged as an important, widely used computerized means for representing and processing pieces of text. Although its main use has been for clerical activities, it can also be of value to decision makers. A text-oriented DSS has a KS comprised of textual passages of potential interest to a decision maker. The PPS consists of software that performs various manipulations on contents of any of the stored text. It may also involve software that can help a user in making requests. The LS contains requests corresponding to the various allowed manipulations. It may also contain requests that let a user ask for assistance covering some aspect of the DSS. The PS consists of images of stored text that can be projected on a console screen, plus messages that can help the decision maker use the DSS.

When a DSS is built with a hypertext technique, each piece of text in the KS is linked to other pieces of text that are conceptually related to it. There are additional PPS capabilities allowing a user to request the traversal of links. In traversing a link, the PPS shifts its focus (and the user's focus) from one piece of text to another. Ad hoc traversal through associated pieces of text continues at a user's discretion. The benefit of this hypertext kind of DSS is that it supplements a decision maker's own capabilities by accurately storing and recalling large volumes of concepts and connections that he or she is not inclined personally to memorize. The World Wide Web furnishes many examples of hypertext DSSs.

#### 2. Database-Oriented DSSs

Another special case of the DSS framework consists of those systems developed with a database (e.g., relational) technique of knowledge management. These have been used since the early years of the DSS field. Like text-oriented DSSs, they aid decision makers by accurately tracking and selectively recalling knowledge that satisfies a particular need or serves to stimulate ideas. However, the knowledge handled by a database-oriented DSS tends to be primarily descriptive, rigidly structured, and often extremely voluminous. The computer files that make up its KS collectively are called a database. The PPS has three kinds of software: a database control system, an interactive query processing system, and various custom-built processing systems. One, but not both, of the latter two could be omitted from the DSS.

The database control system consists of capabilities for manipulating database structures and contents. These capabilities are used by the query processor and custom-built processors in their effort at satisfying user requests. The query processing system is able to respond to certain standard types of requests for data retrieval (and perhaps for help). These requests constitute a query language and make up part of the DSS's language system. Upon receiving an LS request, the query processor issues an appropriate sequence of commands to the database control system, causing it to extract the desired values from the database. These values are then presented in some standard listing format for the user to view. Users may prefer to deal with custom-built processors rather than standard query processors (e.g., faster responses, customized presentation of responses, more convenient request language). Such a processor is often called an application program, because it is a program that has been developed to meet the specific needs of a marketing, production, financial, or other application.

## 3. Spreadsheet-Oriented DSSs

In the case of a text-oriented DSS, procedural knowledge can be represented in textual passages in the KS. About all the PPS can do with such a procedure is display it to the user and modify it at the user's request. It is up to the user to carry out the procedure's instructions, if desired. In the case of a database-oriented DSS, extensive procedural knowledge is not easily represented in the KS. However, the application programs that form part of the PPS can contain instructions for analyzing data retrieved from the database. By carrying out these procedures the PPS can show the user new knowledge (e.g., a sales forecast) that has been derived from KS contents (e.g., records of past sales trends). But, because they are part of the PPS, a user cannot readily view, modify, or create such procedures, as can be done in the text-oriented case.

Using the spreadsheet technique of knowledge management, a DSS user not only can create, view, and modify procedural knowledge held in the KS, but also can tell the PPS to carry out the instructions they contain. This gives DSS users much more power in handling procedural knowledge than is achievable with either text management or database management. In addition, spreadsheet management is able to deal with descriptive knowledge. However, it is not nearly as convenient as database management in handling large volumes of descriptive knowledge, nor does it allow a user to readily represent and process data in textual passages or hypertext documents.

Spreadsheet-oriented DSSs are in widespread use today, being especially useful for studying implications of alternative scenarios. The KS of such a DSS is composed of spreadsheet files, each housing a spreadsheet. Each spreadsheet is a grid of cells, each having a unique name based on its location in the grid. In addition to its name, each cell can have a definition and a value. The definition can be a constant (i.e., descriptive) or a formula (i.e., procedural). The PPS allows a user to change cell definitions, calculate cell values, view those values, and customize the LS and PS (e.g., via macros).

## 4. Solver-Oriented DSSs

Another special class of DSS is based on the notion of solvers. A solver is a procedure consisting of instructions that a computer can execute in order to solve any member of a particular class of problems. For instance, one solver might be able to solve depreciation problems while another solves portfolio analysis problems, and yet another solves linear optimization problems. Solver management is concerned with the storage and use of a collection of solvers. A solver-oriented DSS is frequently equipped with more than one solver, and the user's request indicates which is appropriate for the problem at hand. The collection of available solvers is often centered around some area of problems such as financial, economic, forecasting, planning, statistical, or optimization problems.

There are two basic approaches for incorporating solvers into a DSS: fixed and flexible. In the fixed approach, solvers are part of the PPS, which means that a solver cannot be easily added to or deleted from the DSS nor readily modified. The set of available solvers is fixed, and each solver in that set is fixed. About all a user can choose to do is execute any of the PPS solvers. This ability may be enough for many users' needs. However, other users may need to add, delete, revise, and combine solvers over the lifetime of a DSS. This flexibility is achieved when solvers are treated as pieces of knowledge in the KS. With this flexible approach, the PPS is designed to manipulate (e.g., create, delete, update, combine, coordinate) solvers according to user requests.

In either case, the KS of a solver-oriented DSS is typically able to hold data sets. A data set is a parcel of descriptive knowledge that can be used by one or more solvers in the course of solving problems. It usually consists of groupings or sequences of numbers organized according to conventions required by the solvers. For example, PPS editing capabilities may be used to create a data set composed of revenue and profit numbers for each of the past 15 years. This data set might be used by a statistics solver to give the averages and standard deviations. The same data set might be used by a forecasting solver to produce a forecast of next year's profit, assuming a certain revenue level for the next year.

In addition to data sets, it is not uncommon for a solver-oriented DSS to hold problem statements and report format descriptions in its KS. Because the problem statement requests permitted by the LS can be very lengthy, fairly complex, and used repeatedly, it may be convenient for a user to edit them (i.e., create, recall, revise them), much like pieces of text. Each problem statement indicates the solver and mode of presentation to be used in displaying the solution. The latter may designate a standard kind of presentation or a customized report. The format of such a report is knowledge the user specifies and stores in the KS.

### 5. Rule-Oriented DSSs

The knowledge-management technique that involves representing and processing rules evolved within the field of artificial intelligence, giving computers the ability to manage reasoning knowledge. A rule has the basic form: *If* (premise), *Then* (conclusion), *Because* (reason). A rule says that if the possible situation can be determined to exist, then the indicated actions should be carried out for the reasons given. In other words, if the premise is true, then the conclusion is valid. The KS of a rule-oriented DSS holds one or more rule sets. Each rule set pertains to reasoning about what recommendation to give a user seeking advice on some subject. It is common for the KS to also contain descriptions of the current state of affairs, which can be thought of as values that have been assigned to variables.

The problem processor for a rule-oriented DSS uses logical inference (i.e., reasons) with a set of rules to produce advice sought by a user. The problem processor examines pertinent rules in a rule set, looking for those whose premises are true for the present situation. This situation is defined by current state descriptions and the user's request for advice. When the PPS finds a true premise, it takes the actions specified in that rule's conclusion. This action sheds further light on the situation, which allows premises of still other rules to be established as true, causing actions in their conclusions to be taken. Reasoning continues in this way until some action is taken that yields the requested advice or the PPS gives up due to insufficient knowledge in its KS. The PPS also has the ability to explain its behavior both during and after conducting the inference.

## B. Compound DSS

Each of the foregoing special cases of the generic DSS framework supports a decision maker in ways that cannot be easily replicated by a DSS oriented toward a different technique. If a decision maker would like the kinds of support offered by multiple knowledge-management techniques, the options are use multiple DSSs (each oriented toward a particular technique) or a compound DSS which is a single DSS that encompasses multiple techniques. Just like a single-technique DSS, a compound DSS is a special case of the generic framework shown in Fig. 1. Its PPS is equipped with the knowledge-manipulation abilities of two or more techniques and its KS holds knowledge representations associated with each of these.

An oft-cited type of compound DSS combines the database and flexible-solver techniques into a single system in which the KS is comprised of a "model base" and a database. Here, model base is the name given to the solver modules existing in a KS (i.e., procedural knowledge). The notion of data sets in a KS is replaced by a formal database (i.e., descriptive knowledge). Correspondingly, the PPS includes "model base-management system" software for manipulating solver modules in the model base portion of the KS by selecting those pertinent to a problem at hand and by executing them to derive new knowledge. This PPS also includes database management system software for manipulating data in the form of records held by the database portion of the KS by selecting what data are to be used by solver modules for the problem at hand. A "dialog generation and management system" is the name given to the PPS's knowledge acquisition and presentation abilities; it is concerned with interpreting user requests, providing help, and presenting responses to a user.

A widely used kind of compound DSS combines database and fixed-solver techniques. The database (relational or multidimensional in structure) may be a repository of real-time information or a data warehouse, which is an archive of data extracted from multiple MISs and DSSs. Warehouse contents are not updated, but rather replaced periodically with a new composite of extracted data. Aside from offering ad hoc query facilities, the PPS had a built-in portfolio of solvers. Execution of these solvers is called on-line analytical processing (OLAP).

## C. Artificially Intelligent DSSs

Artificially intelligent DSSs are systems that make use of computer-based mechanisms from the field of artificial intelligence (AI). Researchers in the AI field endeavor to make machines such as computers capable of displaying intelligent behavior, or behavior that would reasonably be regarded as intelligent if it were observed in humans. A cornerstone of intelligence is the ability to reason. This ability, in turn, represents a principal area of research in the AI field, concerned with the discovery of practical mechanisms that enable computers to solve problems by reasoning. An example is the inference engine, which lies at the core of the rule-management technique. Other AI advances finding their way into DSS implementations include natural language processing, machine learning, pattern synthesis, and pattern recognition. An example of the latter is data mining, which attempts to discover previously undetected patterns in large repositories of data (e.g., a data warehouse). Whereas OLAP is directed toward deriving knowledge that satisfies some goal, data mining is more exploratory in its attempt to discover knowledge not previously conceived.

## D. Multiparticipant DSS

Decision support systems that support multiple persons jointly involved in making a decision or a series of interrelated decisions are called multiparticipant DSSs (MDSSs). They are subject to all the generic features of DSS described previously. However, they have added features, making them suitable for supporting participants organized according to some structure of interrelated roles and operating according to some set of regulations. Multiparticipant DSS fall into two major categories: group decision support systems (GDSS) and organizational decision support systems (ODSS).

A GDSS is devised to support situations involving little in the way of role specialization, communication restrictions, or formal authority differences among participants in a decision. An ODSS supports a decision situation in which participants play diverse roles, do not have completely open communication channels, and have different degrees of authority over the decisions. Cutting across the GDSS and ODSS categories is another class of systems known as negotiation support systems. A negotiation is an activity in which participants representing distinct (often conflicting) interests or viewpoints attempt to reach an agreement (i.e., joint decision) about some controversial issue. A negotiation support system (NSS) is intended to help the participants achieve an agreement.

### 1. The Generic MDSS Architecture

As Fig. 2 shows, an MDSS has an LS, PPS, KS, and PS. Several kinds of users can interact with an MDSS: participants in the decision maker being supported, an optional facilitator who helps the participants make use of the MDSS, optional external knowledge sources that the MDSS monitors or interrogates in search of additional knowledge, and an administrator who is responsible for assuring that the system is properly developed and maintained. The MDSS itself is generally distributed across multiple computers linked into a network. That is, the PPS consists of software on multiple computers. The associated KS consists of a centralized knowledge storehouse and/or decentralized KS components affiliated with many computers.

Public LS messages are the kind that any user is able to submit as a request to the MDSS. A private LS message is one that only a single, specific user knows how to submit to the MDSS. Semiprivate LS messages are those that can be submitted by a subset of users. When an MDSS does not provide a strictly public LS, some users are allowed to issue requests that are off limits to others or able to make requests in ways unknown to others.

Similarly, a presentation system can have two kinds of messages: public and private. Public PS messages are those that do or can serve as responses to any user. A private PS message is a response that can go to only a single, specific user. A semiprivate PS message is a response available to some users. An MDSS that has only a public PS presents the same kind of appearances to all users. When private or semiprivate messages are permitted in a PS, some users are allowed to get knowledge that is off limits to others or able to see it presented in ways unavailable to others. A PS response can be triggered by a request from a user or by the PPS's recognition that some situation exists (e.g., in the KS or in the environment).

Figure 2 depicts a threefold classification of knowledge: knowledge about the system itself, knowledge about those with whom the system is related, and knowledge about the decision domain. Shown in the

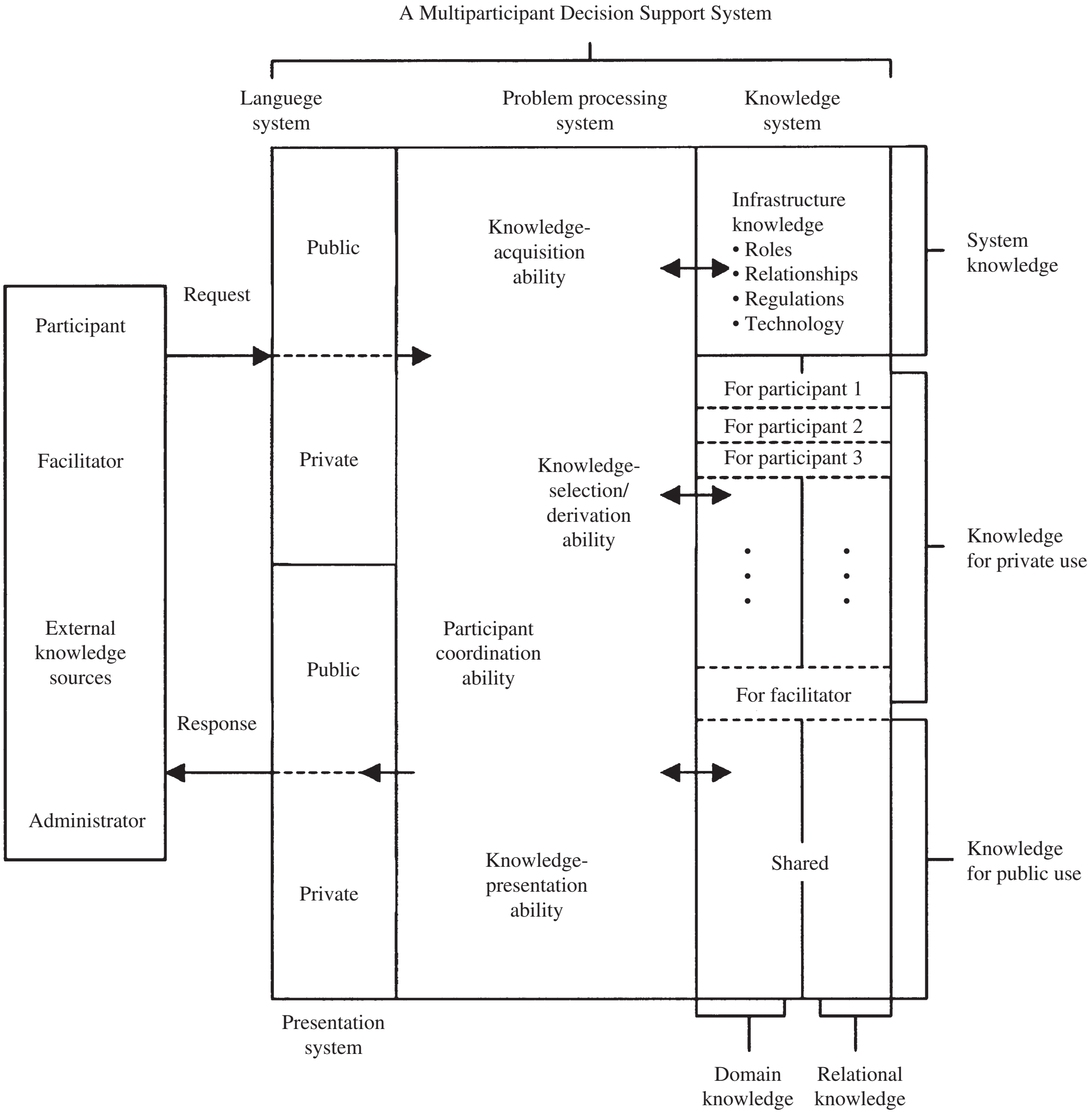


**Figure 2** A generic MDSS architecture.

upper right corner of Fig. 2, system knowledge includes knowledge about the particular infrastructure of which the MDSS is the technological part (i.e., knowledge about participant roles and relationships with which the MDSS must be concerned, plus knowledge of the regulations that it must follow, facilitate, or enforce), and knowledge about technical specifics of the computers involved and their network linkages. The remainder of the KS in Fig. 2 is partitioned into two vertical slices: domain knowledge shown on the left and relational knowledge shown on the right. Each of these categories can have public and private parts. Domain knowledge pertains to the subject matter about which decisions are to be made. It can involve any mix of knowledge types (descriptive, procedural, reasoning). Some of this is public—available to be shared by all interested users. Other domain knowledge may be private, being accessible only to a particular individual user. Relational knowledge can also be public or private. It is concerned with characterizing the users of the MDSS, as distinct from roles they fill.

Observe that the problem-processing abilities identified in Fig. 2 include all those shown in Fig. 1: knowledge acquisition, knowledge selection or deviation, and knowledge presentation. Depending on the MDSS implementation, each of these abilities can be exercised by a user doing some individual work and/or by all participants doing some collective work. As an example of the former, a participant may work to produce a forecast as the basis for an idea to be shared with other participants. As an example of the latter, participants may jointly request the MDSS to analyze an alternative with a solver or provide some expert advice.

The problem processor for an MDSS can have a participant-coordination ability. The coordination ability embodies the technological support for an organizational infrastructure's regulations. It helps regulate the filling of roles, behaviors of roles, and relationships between roles. It draws heavily on the KS's system knowledge; if there is no or little knowledge, the coordination behaviors are programmed directly into the PPS. In the latter case, the PPS rigidly supports only one approach to coordination. In the former case, it begins to approach the ideal of a generalized problem processor—serving as a general-purpose tool for building a wide variety of MDSSs involving diverse coordination mechanisms. With such a tool, development of each MDSS is accomplished by specifying the coordination mechanism (along with other knowledge) in the KS.

## 2. GDSS

Group decision support systems have been the most extensively studied type of MDSS. They are a response to the perceived need for developing better ways to aid the group decision processes that are so commonplace today. The objectives of a GDSS are to reduce the losses that can result from working as a group, while keeping (or enhancing) the gains that group work can yield. There is a growing body of empirical research that has examined the effects of GDSS use on group performance. In broad terms, GDSS effects appear to depend both on situational factors and specific aspects of the technology itself.

Three prominent situational factors are group size, task complexity, and task type. Research results consistently show that a GDSS increases group performance (versus not using a GDSS) more for larger groups than for smaller ones. Satisfaction of participants also tends to be greater as group size increases. There is evidence suggesting that GDSSs may be more appropriate for relatively complex decision tasks. For decision-related tasks of generating ideas, knowledge, alternatives, etc., GDSSs have been found to increase greatly participant performance and satisfaction.

Researchers have also studied factors related to GDSS technology itself: anonymity, parallelism, structuring, and facilitation. The evidence is split on whether the participant anonymity allowed by a GDSS yields better performance than lack of anonymity. It appears that the value of anonymity depends on the task being done and on the specific participants involved. Research has shown that enabling participants to work in parallel is a major benefit of GDSS technology. Investigation of structuring has found that group performance generally increases when a GDSS structures participant interactions. However, the specific structuring used must fit the situation; otherwise performance is impaired. Studies of using a facilitator versus not using one with a GDSS indicate that outcomes are better with a facilitator. Researchers have also found that repeated participant experiences with a GDSS tend to yield increased group performance.

## 3. ODSS

The notion of an ODSS has been recognized for a long time in the DSS field. One early framework viewed an organizational decision maker as a knowledge processor having multiple human and multiple computer components, organized according to roles and relationships that divided their individual labors in alternative ways in the interest of solving a decision problem facing the organization. Each component (human or machine) is an intelligent processor capable of solving some class of problems either on its own or by coordinating the efforts of other components—passing messages to them and receiving messages from them. The key ideas in this early framework for ODSS are the notions of distributed problem solving by human and machine knowledge processors, communication among these problem solvers, and coordination of interrelated problem-solving efforts in the interest of solving an overall decision problem.

Research has identified three main themes that ODSSs tend to have in common. First, an ODSS involves computer-based technologies and may involve communication technology as well. Second, an ODSS accommodates users who perform different organizational functions and who occupy different positions in the organization's hierarchic levels. Third, an ODSS is primarily concerned with decisions that cut across organizational units or impact corporate issues. This third theme should not be regarded as a necessary condition for all ODSSs. However, it is particularly striking in the case of enterprise resources planning (ERP) systems. Although primarily touted as transaction handling and reporting systems, ERP systems have been shown by recent research to have substantial decision support benefits.

## V. CONCLUSION

Over the past half century, advances in computer-based technology have had an important impact on the way in which organizations operate. In the new millennium, continuing advances will revolutionize the way in which we think about organizations and work. The very nature of organizations is changing from an emphasis on working with materials to an emphasis on working with knowledge. Work with material goods will increasingly be seen as a secondary or almost incidental aspect of an organization's mission. It will be little more than an automatic consequence of knowledge processing and resultant decisions. Furthermore, managing an organization's human and financial resources will also become exercises in knowledge management.

The knowledge-management efforts of tomorrow's knowledge workers will be aided and supported by computers in ways described here, as well as in ways just beginning to be apparent. Computer coworkers will not only relieve us of the menial, routine, and repetitive, they will not only be reactive, responding to user's explicit requests, but they will also actively recognize needs, meet some of those needs on their own, stimulate insights, offer advice, and facilitate knowledge flows. They will highly leverage an organization's uniquely human skills, such as intuition, creative imagination, value judgment, and the cultivation of effective interpersonal relationships.

The knowledge-management perspective of decision making and decision support is symptomatic of a major shift in the way in which organizations are viewed. This shift is still in a relatively early stage, but it will strongly affect the management of organizations, decision-manufacturing processes, decision support needs, and even the fabric of society. In a most fundamental sense, organizations will increasingly be regarded as joint human-computer knowledge-processing systems engaged in rich patterns of decision-making episodes. Human participants in these systems, from the most highly skilled to the least skilled positions, are knowledge workers whose decisional efforts will be supported by the increasingly powerful knowledge management capabilities of their computer-based counterparts: decisions support systems.

## ACKNOWLEDGMENT

Portions of this article have been reproduced by permission from *Decision Support Systems: A Knowledge-Based Approach* by Holsapple and Whinston, 1996, West Publishing Company.

## SEE ALSO THE FOLLOWING ARTICLES

Data, Information, and Knowledge • Decision-Making Approaches • Decision Theory • Executive Information Systems • Expert System Construction • Group Support Systems • Hybrid Systems • Knowledge Acquisition • Knowledge Management • Strategic Planning

## BIBLIOGRAPHY

Blanning, R., et al. (1992). Model management systems. *Information systems and decision processes* (E. Stohr and S. Konsynski, eds.). Los Alamites, CA: IEEE Computer Society Press.

Bonczek, R. H., Holsapple, C. W., and Whinston, A. B. (1981). *Foundations of decision support systems.* New York: Academic Press.

Dos Santos, B., and Holsapple, C. (1989). A framework for designing adaptive DSS interfaces. *Decision Support Systems,* Vol. 5, No. 1.

George, J. F. (1991). The conceptualization and development of organizational decision support systems. *Journal of Management Information Systems,* Vol. 8, No. 3.

Holsapple, C. W. (1995). Knowledge management in decision making and decision support. *Knowledge and Policy,* Vol. 8, No. 1.

Holsapple, C. W., Joshi, K. D., and Singh, M. (2000). Decision support applications in electronic commerce. *Handbook of electronic commerce* (M. Shaw, R. Blanning, T. Strader, and A. Whinston, eds.), Berlin: Springer-Verlag.

Holsapple, C. W., and Whinston (1987). Knowledge-based organizations. *The Information Society,* Vol. 5, No. 2.

Holsapple, C. W., and Whinston, A. B. (1996). *Decision support systems: A knowledge-based approach.* St. Paul, MN: West Publishing Company.

Jacob, V., and Pirkul H. (1992). Organizational decision support systems. *International Journal of Man-Machine Studies,* Vol. 36, No. 12.

Jessup, L.. and Valacich, J., eds. (1993). *Group support systems: New perspectives.* New York: Macmillan.

Keen, P. G. W., and Scott Morton, M. S. (1978). *Decision support systems: An organizational perspective.* Reading, MA: Addison-Wesley.

Nunamaker, J., Jr. et al. (1989). Experiences at IBM with group support systems. *Decision Support Systems,* Vol. 5, No. 2.

Simon, H. A. (1960). *The new science of management decision.* New York: Harper & Row.

Sprague, R. H., Jr., and Carlson, E. D. (1982). *Building effective decision support systems.* Englewood Cliffs, NJ: Prentice Hall.

# Decision Theory

**Herbert A. Simon**
*Carnegie Mellon University*

## GLOSSARY

**alternative generation** Creating different potential courses of action among which a choice may be made.

**bounded rationality** Behavior that is goal-oriented, but only within the limits of human knowledge and of ability to predict and compute the consequences of particular courses of action.

**game theory** Theory of rational decisionmaking with multiple actors whose goals may coincide or conflict or both, and who have complete or incomplete information about each other's actions and their consequences.

**heuristic search** Search for good courses of action that examines only a few possible paths, choosing these on the basis of previous experience or by rules of thumb that have been effective in finding good (not necessarily optimal) outcomes.

**intuition in decision making** Recognizing patterns that give memory access to information that is useful for searching selectively for good (satisficing) alternatives.

**multi-agent decision making** Decision making that involves the interaction and collaboration of several (often many) people.

**optimizing** Finding the objectively best alternative in a situation, according to some criterion. In complex real life situations, people can seldom find the optimum, but must satisfice.

**organizational identification** Decision making aimed at furthering the organization's goals, often against self-interest. Identification derives from both cognition (focus of attention and information environment) and emotion (organizational loyalty.)

**rationality** Choice of behavior that is appropriate to specified criteria (goals). It may be appropriate to the real-world situation (substantively rational) or to the situation as perceived by the actor (procedurally and boundedly rational).

**satisficing** Choice of action that meets criteria judged by the actor to satisfy aspirations and to be attainable, but that does not claim to be the best of all objectively possible actions.

**DECISION THEORY** is concerned with the ways in which people choose courses of action, or ought (rationally) to choose them; that is, it describes and explains human decision-making processes, and it evaluates the rationality of the processes and the goodness of the outcomes. Rationality refers to the appropriateness of the actions chosen to the goals toward which they are directed. This article will examine both empirical and normative theories of decision.

## I. APPROACHES TO DECISION MAKING

"Decision" and "choice" may refer only to the selection of an action from a given set of alternatives, or may also embrace the discovery of the alternatives from which to choose. Observation of decision processes in daily life and in organizations reveals that in making decisions, especially the important ones, most of the thought and effort goes into finding and improving alternatives. This article considers how alternatives for decision are identified as well as how a choice is made among them.

This article is especially concerned with how the limits of human knowledge, and the limits of human ability to search, calculate, and compare affect the decision-making process. That is, it deals with the *bounded* rationality of people amidst the complexity of real-life decisions.

Section II discusses the components of the decision-making process and their relations. Section III reviews major theories of decision that are primarily normative and prescriptive. Section IV examines theories that mainly describe and explain choice empirically, with attention to the boundedness of human rationality. Section V discusses the role of learning processes in effective decision making.

Section VI examines the decisions made in organizations. The motivations, knowledge, and problem formulations of decision makers in hierarchical organizations of the kinds that conduct most economic and governmental affairs in a modern industrial society are greatly influenced by their organizational environments.

Almost all of the normative theories of Section III assume that alternatives are given. Theories treated in Section IV, which also encompass the activity of generating alternatives, are usually called "problem solving," "design," or even "discovery" theories. These labels disguise, but do not remove, their concern with decision making.

Research under the explicit label of "decision theory," has mostly been pursued in economics, statistics, operations research, and logic; research under the other labels, and with much more attention to the limits on human rationality, in cognitive psychology, marketing, engineering and architectural design, philosophy of science, and computer science. Until quite recently, there has been very little communication between these two groups of research communities and their literatures, an unfortunate chasm that this article seeks to bridge.

## A. Two Descriptions of Choice

The issues central to human decision making come sharply into focus when we contrast descriptions that approach it from opposite poles, one of them situated in behaviorist psychology, the other in economics and statistics—*behavioral* and *global* descriptions, respectively.

### 1. Behavioral Choice Theory

Human behavior, as described by classical psychology, consists of responses to stimuli. A particular signal to an organism leads it to behave one way rather than another. Of course the state of the organism (e.g., whether it is hungry and what it knows about finding food), also affects the response to the stimulus (e.g., a sniff of prey). The response is triggered by the combined internal and external signals.

Even an organism with multiple needs, but able only to take actions for satisfying one need at a time, can operate rationally to satisfy its many needs with little more machinery than that just sketched. It requires internal signals (thirst) that are transmitted when inventories of need-satisfying substances (water) are low, external signals that suggest directions of search for need-satisfying situations, and a mechanism that (1) sustains attention to sequences of acts aimed at satisfying an active need, and (2) tends to interrupt attention when another urgent need is signaled. A system with these characteristics is rational in choosing behaviors for surviving over indefinite periods of time.

The scheme can be elaborated further, for example, to a traditional agricultural economy, where little deliberative choice has to be made even among sowing, cultivating, and harvesting because appropriate sensory stimuli (seasons, the stages of plant growth) focus attention at each time on the appropriate goal and the activities relevant to reaching it.

In all of these scenarios, rationality means selecting actions that will meet currently signaled needs and signaling needs to replenish low inventories of the satisfying means. The system may incorporate various suboptimizations, but different wants are compared only when there is competition for attention. This kind of rationality, discussed in Section IV, is usually called *procedural,* and reflects the boundedness of human rationality.

### 2. Global Choice

At the other extreme, neoclassical economics and mathematical statistics assume that choice considers much more than immediate internal and external stimuli and (formally) treat decision making as choice among alternative lives. As L. J. Savage, a major early contributor to this theory, put it: ". . . a person has only one decision to make in his whole life. He must decide how to live, and this he might in principle do once and for all." This general notion is embedded in a structure of choice consisting of a utility function that permits the expected utilities of all possible "lives" to be compared and the life with highest expected utility to be chosen.

After making this claim of unbounded global rationality, Savage does hasten to replace it by the more modest claim that ". . . some of the individual decision situations into which actual people tend to subdivide the single grand decision do recapitulate in microcosm the mechanism of the idealized grand decision."

Of course, the reason why the global view of rationality requires all of life to be considered in each decision is that actions today affect the availability not only of all other present actions, but of future actions as well. And the reason why this global procedure is only followed piecemeal, if at all, lies in the boundedness of human rationality: the inability of human beings (with or without computers) to follow it.

Theories of rational decision vary widely in the attention they pay to the interrelatedness of choices at each moment and over periods of time. To the extent that they concern themselves with the optimal outcomes of choice in the real world independently of realizable choice processes, they are said to describe *substantive rationality*.

The theories of Section III are largely theories of substantive rationality. In actual application, they tend to be applied to relatively small slices of a life and of life's decisions. To the extent that a substantive rationality model uses simplified submodels of the real world, and subdivides decisions in such a way as to cut real-world linkages, as it inevitably must, it provides only one approximation among many alternatives, becoming in its actual application another procedural theory.

## II. ELEMENTS OF A THEORY OF CHOICE: THE ANATOMY OF DECISION MAKING

To understand decision making is to understand the nature and structure of actions, how potential actions originate, and the criteria of choice among actions. The examples of the previous section illustrate how wide a range of possibilities this framework encompasses.

### A. The Structure of Actions

A decision theory may deal with single actions, sequences of actions, permitting periodic revision, as new information becomes available, and actions in environments that contain other goal-oriented agents (people, animals, or robots). The theory becomes progressively more complex as it proceeds from each of these stages to the next.

#### 1. Discrete Decision-Action Sequences

The simplest actions, one-time actions, are carried out without significant revision during execution. Even such actions may be complex (e.g., constructing a building), and arriving at decisions about the particulars of the action (e.g., designing the building) may involve gradual discovery and evaluation of alternatives through a process of search.

#### 2. Temporal Sequences of Actions

Complex decisions require a whole structure of choices. Even in design that precedes action, the action is first defined in very general terms, then its details are gradually specified. As information revealed during the latter stages of design requires reconsideration of the earlier stages, there is no sharp line between generating alternatives and evaluating them. Thus, early stages in designing a bridge might opt for a suspension bridge, but subsequent discovery of unexpected difficulties in foundation conditions might cause a switch to a cantilever design.

A somewhat different scenario arises where choosing a present action will have continuing effects on the options for a whole sequence of future periods, but where the contingent decisions beyond the first can be revised at each successive decision point before new action is taken. Games like chess have this characteristic, as do investment decisions, and decisions about the rates of manufacturing operations. Each choice of a current action creates a new situation with consequences for the availability and outcomes of future actions.

A player in chess is only committed to a single move at a time, but looks ahead at alternative successive moves and the opponent's possible replies in order to evaluate the consequences of the immediate move. There is no commitment to subsequent moves after the opponent's reply. The decision maker forms a *strategy* in order to evaluate and choose an *initial action,* but the remaining steps of the strategy can be revised as the situation develops.

In the bridge design case, design decisions are reversible until the design is complete, but not (except for minor, and usually costly, corrections) after construction begins. All uncertainty about consequences is supposed to be resolved before that time. In the case of chess, future actions remain reversible until actual execution time. Where future actions are revisable, the passage of time will resolve some of the uncertainties of the present, and this new knowledge can be used to improve the later actions.

### 3. Decision Making with Multiple Agents

When outcomes depend on the simultaneous actions of other agents (people, animals, or robots), these other agents become an integral part of the uncertainty with which each agent is faced. Even when all are trying to achieve the same goal, each must try to guess (or learn) what the others will do before deciding what action is best for them (e.g., must predict on which side of the street the others will drive).

The problem of multiple independent agents with partially competing goals was first attacked systematically by Cournot in 1838, in his theory of two-firm oligopoly, and again, nearly a century later, by Chamberlin in 1933, then in a far more general way by von Neumann and Morgenstern in their 1944 treatise on game theory. The principal contribution of these monumental theorizing efforts was to demonstrate that multi-person situations attack the very foundations of decision theory, for they destroy the widely accepted definition of rationality: The optimal (goal-maximizing) choice for each person is no longer determinable without knowledge of the choices made by the others.

A plethora of alternative definitions of the criteria for rational decision have been proposed by game theorists in the past half century, with little resolution as to which are to be preferred, either for normative or descriptive theory. This problem will receive continuing attention throughout this chapter.

## B. The Criteria or Goals for Decision

In decision theory, the discussion of choice criteria has largely focused on two polar alternatives: optimizing, that is, selecting the best alternative according to some criterion; and satisficing, that is, selecting an alternative (not necessarily unique) that meets some specified standards of adequacy. Especially in the context of multiple agents, there has also been some discussion of variants of optimizing: minimaxing utility, or minimaxing regret.

### 1. Optimizing

Except in the multi-agent case, it seems at first blush to be hard to fault optimization as a criterion for rational choice. How can one argue for action B if action A is better in terms of the agreed-upon criterion? A first difficulty, as already observed, is that in most multi-agent choice environments there are as many distinct utility functions as agents, so that "best" is not defined, leaving no consensus about the criterion for choice.

Even when the definition of "best" is clear, computational complexity may hide the best alternative from people and from even the most powerful computers. In the finite game of chess (a space of perhaps 1020 branches), it still remains computationally infeasible to insure the discovery of the optimal move. Yet chess is a game of perfect knowledge, where every aspect of the situation is "knowable" to both players. Games like bridge and poker, where each player has private information, add another important layer of complexity.

In the case of a single agent, assuming a comprehensive utility function that embraces all dimensions of choice ignores the difficulty that people find, when placed in even moderately complex choice situations, in comparing utilities across dimensions—comparing apples with oranges. In the field of marketing, a substantial body of empirical evidence shows that consumers use a variety of choice procedures that avoid such comparisons.

One common form of pseudo-optimization meets computational limits by abstracting from the real-world decision situation enough of its complexities so that the optimum for the simplified situation can be found. Of course this choice may or may not be close to the real-world optimum. For example, in making sequential decisions, dynamic programming becomes a computationally feasible method for choice if (and usually only if) radical simplifying assumptions can be made.

Here "radical" could mean, for example, that the problem's cost functions can be closely approximated by sums of quadratic terms. Dynamic programming with a quadratic cost function has been used successfully to reduce costs by smoothing factory operations and inventory accumulations. This procedure simplifies the computations required for optimization by many orders of magnitude, and finds satisfactory, if not optimal, actions—provided that the quadratic approximation is not too far from the actual cost function. Moreover, with quadratic functions, only mean values of future uncertain quantities (e.g., sales) need be predicted, for the higher moments of the probabilities are irrelevant under these circumstances.

A closely related method is used in computer chess programs: a function is defined for evaluating the "goodness" of positions, approximating as closely as possible the probability that they will lead to a won game. Then, the legal moves and the opponent's legal replies are discovered by look-ahead search, as deep as the available time (regulated by tournament rules) permits. The estimated "goodness" is computed for each of the end branches of this tree of possible continuations; then each preceding move for the player is assigned the goodness of the best branch (maximum for the player), and each proceeding move for the opponent is assigned the goodness of the opponent's best branch (minimum for the player), and so on, until a "best" choice is reached for the impending move. This procedure is called minimaxing.

Pseudo-optimizing and minimaxing with approximate evaluation functions are not optimizations, but special cases of the satisficing methods that will be discussed below. If people had consistent and comprehensive utility functions and could actually compute the real-world optimum, pure reasoning, employing only knowledge of the utility function and of the external world, would allow people to achieve substantive rationality and economists to predict their behavior. There would be no need to study people's psychological processes. Neoclassical economics has generally behaved as if this research program were realizable.

As soon as we acknowledge that global optimization is wholly infeasible in a world where human knowledge and computational capabilities are limited, and where judgments of utility are altered by shifts in attention and other internal psychological changes, a host of alternative approximate decision methods present themselves, and predicting human behavior requires a knowledge of which of these approximate methods can be and actually are used by people to arrive at their decisions.

Later, we will see examples of many decision situations where, for example, the appearance of modern computers and of efficient algorithms for exploiting their computing powers have drastically changed the methods used to reach decisions. In its dependence on human knowledge, an empirical decision theory necessarily changes with the movements of history.

## 2. Satisficing

A large body of evidence shows that people rarely actually engage in optimization (except, as just mentioned, in pseudo-optimization after severe simplification of the full situation). Instead, they generate possible actions, or examine given ones until they find one that satisfices or that reaches an acceptable standard in terms of one or more criteria. Satisficing has a number of attractive characteristics for humans whose knowledge and computational abilities are limited, and who demonstrably have difficulty in comparing actions that differ along several, and often among many, dimensions; for example, choosing among buying a Buick car, a sailing sloop, or an oil painting.

1. Satisficing does not require that all possible actions (usually hard to define in any operational way) be made available for comparison.
2. It does not require computing the consequences of actions with precision.
3. It does not require deriving a single utility ordering from multidimensional goals.
4. Unless the satisficing criteria are too strict, it enormously simplifies the computational task of finding an acceptable action.

With respect to 1, it is sometimes proposed to retain optimization but to incorporate a search for alternatives by comparing the expected cost of generating the next alternative with its expected advantage over the best existing alternative, and halting the process when it does not pay to search further. This proposal requires estimating both the cost of generating alternatives and the expected value of the improvement, imposing a large additional computational burden (which usually also calls for unavailable information).

With respect to 3, observation of people selecting meals from restaurant menus shows how difficult choice is in a multidimensional space even when the dimensions are few. Empirical research, by Amos Tversky, Maurice Allais, and many others, has demonstrated that people do not possess a stable utility function over all of their wants and preferences that would enable them to choose consistently among unlike commodities. Choice depends heavily on context that directs attention toward particular aspects of the choice situation and diverts it from other aspects. This contingency of choice on the focus of attention is illustrated by, but extends far beyond, the well-known phenomenon of impulse buying.

Linear programming has been, for a half century, a widely applied computational tool for making complex decisions. One reason for its practical importance is that it permits satisficing under the guise of optimizing. In linear programming if there are $n$ applicable criteria, then one (for example, cost) is selected for optimization, and satisfactory levels of the other $n - 1$ are set as constraints on acceptable actions, thereby wholly avoiding the comparison of goals, but of course at the expense of ignoring the possibilities of substitutions with changes in price. In the restaurant example, this could lead one to order the cheapest meal meeting certain standards of taste and nutrition, or with a different choice of the dimension for optimization, to order the seafood meal under $25, not containing eel and with the least cholesterol.

With respect to 4, comparing the task of finding the sharpest needle in a haystack with the task of finding a needle sharp enough to sew with illustrates how satisficing makes the cost of search independent of the size of the search space, although dependent on the rarity of satisficing solutions. In a world rich in infinite search spaces, this independence is essential if a search method is to be practicable. Few people examine all the world's unmarried members of the opposite sex of appropriate age before choosing a spouse.

Satisficing is closely related to the psychological concept of aspiration level. In many situations, people set a plurality of goals or aspirations on the several dimensions of choice (for example, housing, careers, clothing, etc.) and use these to define a satisfactory lifestyle along each dimension. It is observed that when they reach their aspiration on any dimension, the aspiration tends to rise somewhat, and when they continue to fail to reach it for some time, it tends to decline. Trade-offs can reallocate income among dimensions without requiring a unified utility function, making comparisons of level only on single dimensions.

Such an apparatus for decision making amounts to a linear programming or integer programming scheme, with the constraints adjusting dynamically to what is attainable. No special dimension need be selected for maximization or minimization. The gradual adjustments of aspiration levels will bring about near uniqueness of available satisfactory alternatives, and will take advantage of improving environments and adapt to deteriorating ones. Which alternative will be selected is likely to be highly path dependent, varying with the history of changes in the environment and the decision maker's responses to them.

## C. Search Processes

Where the alternatives are not given at the outset, they must be discovered or designed by a search process, often called a design, problem solving, or discovery process. When the space of alternatives is large, or when uncertain consequences extend into the indefinite future, search may have to be highly selective, examining only a small fraction of the possibilities. The space of conceivable alternatives is seldom small enough to be searched exhaustively.

### 1. Selective Search

Some puzzle-like problems whose search space is almost trivially small are, nevertheless, quite difficult for people. For example, in the Missionaries and Cannibals puzzle, which involves carrying a group of people across a river in a boat of limited size requiring multiple trips, and with the proviso that cannibals must never be allowed to outnumber missionaries, there are only about 20 legal problem states. Yet intelligent people, on their first encounter with the problem, often take half an hour to solve it. In this puzzle one or more essential moves are counterintuitive, appearing to retreat from the final goal, instead of approaching it. In the subject's early attempts to solve the problem, these counterintuitive moves are usually not even considered.

In most real-world decision situations, however, the number of potential alternatives is very large and often infinite. These call for generating, with moderate computation, a small menu that contains at least one satisfactory alternative. A large part of problem-solving theory is concerned with describing powerful search heuristics for selecting promising paths. The substitution of satisficing for optimizing is an important heuristic of this kind.

### 2. Search under Uncertainty

In repeated decisions, the purpose of search goes beyond finding good initial alternatives and includes assessing future consequences of a choice. For example, in chess, it is trivial to generate the twenty or thirty legal moves that face a player at any moment; the formidable computational challenge is to evaluate, usually by generating a tree of subsequent replies and moves, which of these initial moves is better.

As the outcomes of future or present moves are seldom known with certainty, computer chess programs, as we have seen, assign to each terminal position in their search a value estimated from the numbers, kinds, locations, and mobilities of the pieces of each player. The scheme can also allow some kind of preference for lower or higher risk due to uncertainty. This approximate value is then used, as previously described, to work backward, minimaxing to evaluate the current move.

The fundamental differences between uncertainty about nature and uncertainty about the behavior of other actors have already been discussed. Where uncertainty resides in nature, a common procedure is to maximize expected value, with perhaps some bias for risk. Minimaxing in the face of nature's uncertainty is equivalent to regarding nature as malevolent and preparing for the worst. In the case of uncertainty about human actors, goals may range from the completely complementary to the wholly opposed, creating considerable difficulty in defining in any objective way how other actors will respond to a decision maker's choice, hence making it hard to choose a unique criterion for rational choice.

## D. The Knowledge Base

The decision process must use its knowledge about the external environment; but in virtually all real-world situations, this knowledge is only a crude approximation to reality.

Knowledge may be obtained by sensing the environment directly, by consulting reference sources, or by evoking information previously stored in some memory. To use any of these sources it must be accessed: sensory information, by focusing attention on specific parts of the stimulus; external reference sources, with the aid of more or less elaborate data-mining processes; and memory, using recognition processes to locate and bring to awareness relevant stored information. An operational theory of decision making must distinguish between the knowledge potentially available and the knowledge actually accessible during decision processes that use these modes of information retrieval.

Research has shown that a fundamental basis of expertise is the extensive use of memory, accessed by recognition of familiar patterns in the material being attended. These recognitions evoke already stored information relating to the patterns. The expert solves by recognition, and thereby by use of previous knowledge and experience, many problems or problem components that less expert persons can only solve (if at all) by extensive and time-consuming search. Much, and perhaps most, of the sudden "aha's," "insight," "intuition," and "creativity" that are characteristic of expert behavior result from the recognition of familiar patterns.

Both expert and novice behavior combine heuristic search with recognition of patterns. What mainly distinguishes expert from novice is the availability to the expert of a vastly bigger repertoire of patterns and associated information, making pattern recognition a much larger component, and search a smaller component of expert than of novice problem-solving behavior.

# III. NORMATIVE THEORIES OF DECISION

Formal decision theory began mainly as a normative science, initially providing advice to gamblers about good strategies. The early exchanges on this topic, in 1654, between Blaise Pascal and Fermat, and Jakob Bernoulli's treatise on The Art of Conjecture in 1713 are key events in these beginnings. Today, the theory exists in many versions; a number of the most important are described here.

## A. Classical Decision Theory

Classical decision theory has retained its early shape, but with continuing debate about the interpretation of probability and the means to be used for estimating it. The main line of development has led to the theory of maximizing expected utility.

Both frequency theories of probability and subjective theories of probability as "degree of warranted belief" have been entertained throughout the history of decision theory. Probability takes care of natural contingencies that may alter the consequences of choice. Frequency interpretations can be used when empirical evidence is available for estimating the probabilities, especially when there is enough evidence so that the law of large numbers assures close approximation of observed frequencies to probabilities.

As already noted, when uncertainties involve not only the natural environment but also the behavior of actors other than the decision maker, the situation becomes more complex, because all actors may be seeking to adjust their behaviors to the expected behaviors of their collaborators and competitors. In some special cases, however, like the economic theory of perfect competition, these complexities are absent.

Suppose there exists a price for a commodity at which the total quantity that will be offered by profit maximizing sellers is equal to the total quantity that will be offered by utility maximizing buyers. A seller who supplies a larger or smaller quantity will lose profit and a buyer who buys more or less will lose utility, so no one has an incentive to alter his/her behavior, and the equilibrium of the market (under certain assumptions about the dynamical processes for reaching equilibrium) is stable. Of course this stability depends critically upon the assumption of perfect competition, and uniqueness is not guaranteed without additional assumptions, or anything but local optimality for each buyer and seller.

The Nash equilibrium, the generalization of this result to any situation where no participant has an incentive to change behavior as long as the others maintain theirs, describes one of the few cases where the definition of rationality under optimizing assumptions is not problematic in the presence of more than a single agent. An application outside economics is the traffic flow problem, where cars are proceeding independently through a network of highways. In this case, we can usually expect one or more equilibrium distributions of traffic among the different highways such that no single motorist could, on average, shorten trip time by changing route as long as the others maintained their patterns.

There is no guarantee that alternative equilibria are equally efficient. A particular equilibrium pattern might be improved, for most or even all drivers, if they could all shift simultaneously to another pattern, but no driver has a motive to shift pattern without changes by the others; satisfaction of the conditions for a Nash equilibrium guarantees only a local optimum. In particular, it is not generally possible to reach global optima by market forces alone without auxiliary processes and institutions that coordinate individual behaviors to avoid inferior local optima.

## B. Statistical Decision Theories; Acceptance of Hypotheses

The standard procedures used to interpret statistical findings and to base decisions upon them are closely related to classical decision theory. Initially, the problem was conceived as that of computing a strength of justifiable belief—sometimes expressed as the probability of the truth of a hypothesis. Thus, if the question were whether a fertilizer made sufficient improvement in the yield of a crop to justify the cost of applying it, experimental data would be used to compute probability distributions of the yields with and without the fertilizer. If there was "reasonable certainty" that the yield with the fertilizer was enough greater than the yield without it to cover the extra cost, it would be rational to apply it.

But what was reasonable certainty? An early answer, called a "test of statistical significance," was: If the probability is "high" that the difference of yield is not enough to cover the cost, then don't use the fertilizer.

That simply raised the new question: What is high? In contemporary statistical practice, this question is often answered in purely conventional terms. By long custom, more than 1 chance in 20 (0.05), or more than 1 chance in a 100 (0.01) have come to be regarded as high. If the probability is greater than 0.01 or 0.05 that the fertilizer's effects will not cover its cost, then its effect is not "statistically significant," and it should not be used. Although there is no rational basis for this rule, in some domains of science and application it is deeply entrenched.

A step forward was taken by J. Neyman and E. S. Pearson, who pointed out that one could err either by using the fertilizer when it was uneconomical or by failing to use it when it more than covered its cost; consequently the decisionmaker should compare these "errors of type 1 and type 2." For example, if, prior to the field experiment, the evidence indicated a 50–50 chance that the fertilizer would be cost-effective, then (if one accepts Bayes' Principle), the fertilizer should be used whenever the mean added crop yield is worth more than the cost of fertilizer. This is quite different from odds of 0.05 or 0.01.

A next step forward, taken by Abraham Wald in 1950, was to argue that net costs should be assigned directly to errors of type 1 and type 2, that these costs should be weighted by the probabilities of committing the errors, and the alternative with the largest expected balance of benefits over cost should be chosen.

## C. Linear and Integer Programming

Linear and integer programming were mentioned in Section II as methods for dealing with multidimensional goals. They supplement a one-dimensional measure of utility with an unlimited number of linear constraints to represent the other dimensions. The alternative is selected that maximizes the chosen dimension of utility while satisfying all of the constraints. The optimal diet problem, described earlier, illustrated how this procedure avoids comparing the relative importance of incommensurable constraints, for all of them have to be satisfied.

Linear programming (LP) has a further important property: the alternatives that satisfy all the constraints form a convex set. Given one point in the set, any point in it with a higher utility can be found without any backtracking. The popularity of LP was quickly established after 1951, when George Dantzig introduced a powerful search algorithm, the simplex method, that exploits this convexity property and generally finds the optimum with acceptable amounts of computation even for problems containing thousands or tens of thousands of constraints.

As soon as powerful electronic computers became available, LP became a practical tool for making many complex managerial decisions: to take two classical examples, blending crude oils in the petroleum industry, and mixing commercial cattle feeds, which allocate billions of dollars of raw materials in our economy each year.

In this case, bounded human rationality (aided by computers) can be reconciled with optimization. But this is only achieved by LP after the problem has been redefined as a one-variable maximization problem subject to linear constraints. Moreover, the procedure assumes that the model embodies all the real-world consequences of the decision it is making, which is usually far from the case.

Here as elsewhere, the real-world situation must be drastically simplified to fit the formal tools. The procedure may satisfice in the real world by optimizing in an approximation to that world; it may even come close to optimizing, although we cannot usually verify this.

The story of integer programming (IP) is similar. Many real-world problems require integer solutions (you cannot equip a factory with 3.62 machines, but must settle for 3 or 4.). A variant of LP, which searched for the optimal integral solution, was introduced by Ralph Gomory in 1958, and increasingly powerful methods for finding such solutions have steadily appeared. Even when IP cannot find an optimum, it can often reach satisficing solutions substantially better than those attainable with less powerful methods.

## D. Dynamic Programming

Similar lessons have been learned with dynamic programming, whose computations are usually impracticable for problems of any generality. Systems of dynamic equations are solvable in closed form only in quite special cases, notably for linear systems with constant coefficients. The problem becomes much worse when uncertainty is introduced, for then a multitude of possible outcomes must be considered, increasing exponentially with the distance into the future the system peers.

Again, we have seen that computationally tractable approximations can be obtained by making the further simplifying assumption that the costs can be approximated by sums of quadratic terms. But this approximation yields a second important simplification: now only the expected values of the probabilities remain in the equations, so that only mean values over all outcomes have to be estimated for each period. These mean values are called certainty equivalents. The standard deviations and higher moments of the probability distributions now do not have to be estimated at all.

There is no guarantee, of course, that particular real-world situations can be approximated adequately in these ways, but these forms of approximation illustrate the advantages that can be gained in many situations by optimizing a gross approximation (i.e., satisficing) instead of attempting to optimize a realistic detailed model of the situation.

## E. Game Theory

Game theory in situations with multiple actors has already been discussed in Section II.A.3 and little needs to be added. The attempt to build a general normative theory of what constitutes rationality in the choice of moves in *n* person games has failed, not through any lack of talent among the researchers who tackled this problem, but from its fundamental intransigence. Progress can be made only by extending the theory to encompass both the degree of compatibility or incompatibility of the agents' goals and their means of communication and coordination of information and behaviors.

Here, useful definitions of rationality in terms of optimization are unlikely to be found, both because of goal conflict and competition in multi-agent situations and because of the complexity of these situations when communication and coordination mechanisms are brought into the picture.

But the matter need not be described quite so pessimistically. There has recently been, especially in experimental game theory, increasing exploration of satisficing solutions for games that allow most or all agents to reach satisfactory levels of aspiration. There are innumerable situations in the world today where ethnic, religious, national, and other divisive loyalties have created "unsolvable" social problems, creating a great need for normative procedures for arriving at satisficing arrangements that will be acceptable (if sometimes only minimally) to those engaged in such struggles. In situations like these, we cannot afford to sacrifice the attainable satisfactory for the unattainable "best."

## F. The Future of Normative Theories

For more than a century, the problem of defining rational decision making has been studied intensively with the help of powerful formal tools. That study arrived, first, at a body of theory focused on maximizing goal attainment, where goals are summarized in a hypothesized multidimensional utility function. At a second stage, uncertainty was brought into the picture, adding severely to the informational and computational demands upon the theory.

Then, the theory was gradually extended to multiagent systems, not only introducing new computational problems, but also making it extremely difficult to reach consensus even on the definition of "rationality," except in a few special cases.

In this literature, there were only a few attempts to deal with the generation of the new alternatives. Alternative generation, if considered at all, was viewed as a search balancing costs of locating new alternatives against their expected contribution to improving decisions; but (1) this approach calls for estimating search costs and benefits, only adding to an already excessive computational burden, and (2) it does not address the processes of alternative generation or the nature of the alternatives discovered, but simply hypothesizes a cost function for the process.

Although decision theorists are very far from solving these problems, they have learned much about the structure they are seeking. More and more, they are engaging in experimental and other behavioral studies to deepen their understanding of how people actually make decisions. As one consequence, the normative theory of decision is now making contact with positive, empirical study to an unprecedented degree. It has steadily become clearer that even a normative theory must give major attention to the cognitive abilities and limitations of the human agents who make the decisions, and to the capabilities and limitations of their computer aides.

The next section of this chapter turns to the empirical theory of decision making, but also explores the implications of the empirical theory for constructing a more general and viable normative theory that can be applied to practical affairs. For, as decision making is a goal-oriented, hence normative, activity, there is no great distance between an empirically supported positive theory of human decision making and a normative theory that respects bounded rationality and is applicable to real situations.

## IV. THE DECISIONS OF BOUNDEDLY RATIONAL ACTORS

The developments discussed in this section have mostly taken place in cognitive psychology (problem solving and learning theory), engineering and architecture (design theory), evolutionary theory (natural selection), and history and philosophy of science (scientific discovery). Cognitive science has emphasized the empirical side, whereas design is basically a normative endeavor, but one that must be highly sensitive to implementability, hence to the bounded rationality of people and machines.

Although there is a considerable consensus about the main features of these forms of decision theory, they are best discussed in relation to their several disciplinary origins.

### A. Problem Solving

Problem solving, which provided a major focus for research during the first decades of artificial intelligence and the so-called "cognitive revolution," remains a very active domain of study. Early research studied mainly puzzle-like problems where the solver did not require special information; emphasis was upon search strategies and the heuristics for achieving selectivity. The main exception was research on chess, which discovered the powerful role played by recognition and knowledge retrieval as an aid to expert search.

When given a problem, human subjects typically formulate a problem space to characterize the kinds and numbers of objects involved, their properties and the relations among them, and the actions that can be taken to change one situation into another while searching for one that satisfies the goal conditions. Subjects usually try to measure the "closeness" of the current problem situation to the goal in order to select hill-climbing actions that move closer to the goal.

A more selective and quite common search procedure seeks solutions by means-ends analysis. Subjects gradually learn that certain differences between current situation and goal situation can be removed by applying particular operators. They then can compare current situations with goal situations to discover the differences and apply the operators associated with these differences. In many problem domains, this means-ends process can solve problems by removing differences successively.

With the achievement of a rather rich theory of problem solving in puzzle-like domains, research moved on to domains where solution depended heavily upon special domain knowledge, and where search heuristics were much more domain specific. One typical heuristic for the game of chess is the rule: "If there is an open file (sequence of empty squares crossing the board), consider placing a rook on it"—i.e., when you notice a particular pattern on the board give serious consideration to a particular move. It has been found that high-level expert chess players (masters and grandmasters) hold in memory a quarter million or more patterns of pieces (each containing from two or three to a dozen or more pieces) that are seen in games, and that they associate with each pattern heuristic information, as in the example of the rook move, that guides the choice of moves.

Even more recently, two other directions of research on problem solving have gained prominence: (1) the study of the acquisition of problem-solving skills; and (2) the extension of research to more loosely structured problem domains like architectural design, scientific discovery, and even drawing and painting.

Problem-solving theory almost always takes account of uncertainty about the environment and about the consequences of actions, but less frequently (with the notable exception of research on chess and other games) uncertainty about the behavior of other agents.

## B. Design Theory

Designing systems requires innumerable decisions about system components and the relations among them. Since the initial recognition of the capabilities of computers to aid in design, and even to automate design processes, there has been a rapid development of design theory, focusing upon the generation and evaluation of alternatives. Today, the theory is pursued in computer science and cognitive science as well as in architecture and all of the engineering disciplines.

A notable early set of programs were written in 1955–1956 in FORTRAN by engineers at Westinghouse to design motors, generators, and transformers automatically from customers' specifications, producing designs that went directly to the manufacturing floor. The programs could design about 70% of the devices ordered that were not already shelf items. They were modeled on procedures already regularly used by the company's engineers: find products already designed that are similar to the customer's order; use known function and parameter tables to modify the design to fit the customer's specifications, and apply simple optimization methods to improve device components.

Today, we see increasing numbers of programs that assist and collaborate with human designers. Computer-aided design (CAD) and computer-aided manufacture (CAM) are progressing from an aid to draftsmen and schedulers to an increasingly automated component of the design process itself. A powerful genetic algorithm can design electrical circuits of high quality (e.g., low-pass filters with specific properties) with moderate computational effort. Design programs are used routinely in the chemical industry to discover and to assist in discovering reaction paths for synthesizing industrial chemicals and biochemicals. These are just examples from a sizeable population of design programs in current use.

These programs have required their inventors to develop the theory of the design process, which is no longer simply an intuitive "skill" to be taught at the engineering drawing board. The theory applies principles of pattern recognition and heuristic search like those that have emerged from the empirical study of human decision making and problem solving.

To be sure, when computers participate in design, their capabilities for rapid large-scale computation and for rapid storage of huge amounts of information—both far beyond human capabilities—are exploited, altering the balance between "brute force" and selective search. But in view of the magnitude of typical practical design problems, major use must be made in these programs of the principles that guide human problem solving: in particular, evocation of knowledge by recognition, highly selective search, and satisficing.

## C. Theory of Discovery

A central theme in the research on decision-making and problem-solving processes has been to expand the domain of the investigation continuously to new areas of knowledge and skill, beginning with well-structured and simple puzzle-like problems, soon moving to problems where domain knowledge is important, and gradually to problem areas that are poorly structured and where "intuition," "insight," and "creativity" come into play.

One important area in this last category is scientific discovery. Scientific discovery employs a collection of diverse processes that include discovering lawful regularities in data, planning and designing experiments, discovering representations for data, and inventing scientific instruments. Each of these activities is itself a complex decision process. This section provides two illustrations, from a much larger number, of how evidence from the domain of scientific discovery contributes to the theories of procedural (satisficing) rationality.

The first illustration is to discover a descriptive scientific law is to find pattern in a body of data; to discover an explanatory law is to show how pattern in data can be explained in terms of more detailed processes. Thus, Kepler's Third Law, P = aD3/2, where P is the period of revolution of a planet about the sun, and D is its mean distance from the sun, is a descriptive law. Newton later explained it by deducing it mathematically from the laws of gravitational attraction.

A law-discovery process, BACON, which has shown great power in finding historically important descriptive laws, is a very simple mechanism for choosing, one-by-one, patterns that might fit the data, testing for the goodness of fit, and if the fit is unsatisfactory, using information about the discrepancy to choose a different function. When one is found that fits adequately, a law has been discovered.

BACON uses the method of generate-and-test, with heuristics to guide generation of prospective laws. Its law generator is responsive to discrepancies between the hypotheses it produces and the data that guide it, using the feedback to shape the next alternative it generates. BACON frequently finds laws after generating only a few candidates, and has rediscovered more than a dozen historically important laws of physics and chemistry with no information other than the data to be explained, and with no changes in the program from one example to another.

BACON satisfices, terminating activity upon finding a pattern that meets its criterion of accuracy. When the standard is too low, it can find illusory "laws," as Kepler did in his first hypothesis about the revolution of the planets (P = aD2), and which he corrected a decade later. (With a stricter criterion, BACON finds this same "law" as the second function it tries, but rejects it.)

The second illustration is experimentation which, along with empirical observation, are major activities in science, generally regarded in the philosophy of science as means for testing hypothesis. In recent years, there has been growing attention to experiments and observation as the major means for generating new hypotheses. New and surprising phenomena, whatever motivates their discovery, then initiate efforts to explain them. The work of the great experimenter and theorist, Michael Faraday, provides striking examples of these processes.

Faraday, in about 1821, conjectured, from a vague belief in symmetry between electricity and magnetism, that, as currents created magnetic fields, magnetic fields "should" create adjacent electrical currents. After ten years of intermittent unsuccessful experimenting, Faraday, in 1831, found a way to produce a brief transient current in a wire when a nearby magnet was activated. In several months of work, without any strong guidelines from theory, but with feedback from a long series of experiments, Faraday modified his apparatus, finally producing a continuous current, the basis for the modern electric motor.

A computer program, KEKADA, capable of closely simulating Faraday's experimental strategy (as well as the very similar strategy of Hans Krebs in his discovery of the reaction path for the *in vivo* synthesis of urea), has demonstrated the mechanisms used in such searches. KEKADA's model of experimentation begins with extensive knowledge of the phenomena of interest, accesses it by recognizing patterns of phenomena, and uses heuristics to select promising experimental arrangements for producing relevant and interesting phenomena. For example, when its expectations of experimental results based on previous knowledge and experience are violated, KEKADA designs new experiments to determine the scope of the surprising phenomenon, and then designs experiments to discover possible mechanisms to explain it.

## V. DECISION THEORIES INCORPORATING LEARNING

Effective problem solving in a knowledge-rich domain depends vitally on prior knowledge of that domain, which must have been "programmed in" or learned, and stored with indexing cues to make it accessible. An implementable theory of decision making must specify what knowledge is available to the system, its organization, and its mechanisms for knowledge acquisition. Some knowledge and skill used in making decisions is applicable in many domains, but much of it is specific to one or a few domains.

Learning has been a central topic in psychology for a century and recently in decision theory. In systems aimed at either optimizing or satisficing, adaptive mechanisms may gradually improve outcomes by learning from experience. The improvement may move asymptotically toward an optimum or, more commonly, it may simply move toward higher satisficing levels. Adaptive mechanisms commonly undertake selective search for alternatives to the current strategy. Hence, the learning mechanisms are themselves decision mechanisms.

We cannot review the vast literature on learning, but will mention some examples of learning mechanisms to give some appreciation of their variety and power and their importance to rational decision making.

1. Many problem-solving systems store solution paths they find, and then use them as components of paths for solving new problems. Thus, a theorem-proving system may store theorems it has proved and use them in proving new theorems.
2. Learning can change one's knowledge of the

problem space, hence of the best directions for search; it can also be used to generate new alternatives that enlarge the problem space. In fact any alternative generator (e.g., BACON, discussed earlier) is a learning mechanism.

In recent years, there has arisen a strong interest in systems that create and evaluate new alternatives by using learning mechanisms that imitate evolution by natural selection. In genetic algorithms, systems change by mutation, and changes that improve performance are preferentially retained. A program was mentioned earlier that designs electronic circuits in this way by progressively assembling and modifying sets of elementary components.

3. Generalizing, wherever uncertainty is present, and where it is reasonable to assume some reliable, if stochastic, process underlying the uncertain events, and learning the relevant probability distributions can aid rational decision making. Research along this line divides into two streams; one, associated with classical decision theory, undertakes to describe optimal learning processes; the other, associated with psychological learning theory, undertakes to describe actual human learning processes.

In a wide class of stochastic decision processes, current decisions are optimal relative to current inexact estimates of the relevant parameters that describe the uncertain events, and at the same time, current data are used continuously to revise these estimates and update the parameters. A Bayesian probability model, for example, may be used for this purpose.

So-called "connectionist" and "neural net" models of cognitive processes make extensive use of probabilistic processes of these kinds for learning, seeking to maximize the probabilities of correct choices.

4. In addition to connectionist mechanisms, there are powerful non-probabilistic discrimination networks, like E. Feigenbaum's EPAM, that learn. Given a set of stimulus features, they apply successive tests to find features that discriminate among classes, building up classificatory trees that categorize stimuli.
5. When decision processes take the form of systems of if-then rules (productions), new rules can be learned by examining worked-out examples, and stored as productions that can be applied to subsequent problems. A worked-out example shows how the problem situation is steadily altered step by step by the application of operators, to remove differences between the current and the goal situations. The learning system detects the differences and the operators that remove them, then constructs a new production whose "if" part corresponds to the difference and whose "then" part corresponds to the operator: "If [difference] is present, then apply [operator]."
6. The ease or difficulty of learning may be strongly influenced by what the learner already knows, with existing knowledge sometimes discouraging new learning that moves in novel directions. Thus, it has been widely observed that radical technological changes in industry typically cause great difficulties in adaptation for existing organizations and give birth to new organizations to exploit them.

## VI. DECISION MAKING IN ORGANIZATIONS

The economy of Adam Smith's time was predominantly an economy of markets; business organizations of more than a modest size played in it an almost insignificant role. The industrial revolution saw a great burgeoning of large-scale organizations, greatly increasing manufacturing and marketing efficiency.

The earliest large body of theory of organizational decision making focused upon division of labor and coordination through communications and authority relations, comparing the efficiencies of various organizational arrangements. It was initially derived from the practices of the large governmental and military organizations that have existed for the past several thousand years, and has been further developed by the fields of public and business administration.

Within economics, an important effort has been made in recent years to incorporate organizations in (boundedly) rational decision theories by asking when various classes of economic transactions will be carried on through markets, and when within organizations. The transaction costs associated with particular activities when conducted by markets and by organizations are compared to determine where the economic advantage between these two institutional forms lies, under the usual assumption that agents will pursue their self-interests (opportunism). The research in this area pays a good deal of attention to empirical verification of its theories, especially through case studies of the experiences of particular industries and historical analysis, and in this way avoids the a priorism of classical theories of optimization.

The particular characteristics that distinguish organizations from other multi-agent structures are

(1) their hierarchical organization of authority and membership through an employment relation that implies acceptance by employees of the organizational authority, and (2) their central concern with problems of coordination, where the correctness of a member's decisions depends on the decisions being made by many others. Organization theory has to describe both the processes for dividing complex tasks into nearly independent subtasks and their corresponding organizational units, and the processes of coordinating units to exploit their remaining interdependencies, balancing the advantages of unit independence and coordination.

In addition to transaction costs and the problems of opportunism, several other major features must be added to create a credible theory of organizational decision making. In particular, members of organizations spend their working lives in environments that direct their attention to organizational rather than personal concerns, and that select the kinds of information they receive and the beliefs they acquire. Moreover, members typically acquire loyalties to goals of the organizations or of subdivisions within it. These processes of selective learning and acquisition of organizational loyalties, which collectively create organizational identification, greatly moderate the problem of keeping employees' decisions consistent with organizational goals.

We can conceive, today, of a computerized organization whose programs would make this consistency automatic. Real organizations fall far short of this condition, and a decision theory for organizations must address the problem of maintaining loyalty to organizational goals and dealing with the shirking problem. The decision processes of organizations, with their central task of coordinating the interdependent decisions of their component divisions and departments, cannot be described simply as market transactions.

## VII. CONCLUSION

Decision making is a ubiquitous human cognitive function that is accomplished by recognizing familiar patterns and searching selectively through problem spaces (which themselves may need to be chosen or even discovered) to arrive at a "satisficing," or "optimal" alternative.

In the course of time, two rather independent sets of decision theories have emerged. Theories of substantive rationality purport to deal with the outcomes of decision in the (idealized) real-world environment. They have been little concerned with the psychology of choice, beyond postulating that the agent orders preferences consistently among all the (given) alternatives, and they represent uncertainty by probability distributions. They seek to discover the decision that optimizes goal achievement in the external environment.

Theories of procedural rationality purport to describe how decisions are actually made by human beings, including how the alternatives are found. They focus upon the processes that people actually use, taking account of the limits on human knowledge and computational power.

It would be a mistake to draw an impassable line between these two bodies of theory; indeed, interaction between them accelerated rapidly in the closing years of the 20th century. First, concern with incomplete information called attention to the complications that uncertainty adds to the decision process. One response was to replace certain knowledge with probabilities and to optimize expected values. Another was to introduce specific assumptions of irrationality due to "ignorance." A third was to replace optimization with an adaptive learning system.

Second, growing interest in imperfect competition and rationality in multi-agent situations called attention to the severe difficulty of defining rationality in such situations, leading to experimental studies of how people actually behave in these circumstances and developing the psychologically motivated theory of satisficing as an alternative to optimizing.

Third, a new interest developed in normative theories of decision that could be implemented on computers. In the world of design, optimizing means finding a mathematical optimum in a simplified operative model that can find satisficing solutions to real-world problems.

The two streams of substantive and procedural theories are now converging, and the coming years will produce an improved understanding of human decision processes as well as a growing collection of partly or wholly automated man-machine systems for coping with the decisions that people, business organizations, and governments have to make.

## SEE ALSO THE FOLLOWING ARTICLES

Corporate Planning • Cybernetics • Decision-Making Approaches • Decision Support Systems • Game Theory • Information Measurement • Information Theory • Optimization Models • Systems Science • Uncertainty

## BIBLIOGRAPHY

Akin, O. (1986). *The psychology of architectural design.* London, England: Pion, Ltd.

Conlisk, J. (1996). Why bounded rationality, *Journal of Economic Literature,* 34, 669–700.

Dixit, A., and Nalebuff, B. (1991). *Thinking strategically: The competitive edge in business, politics, and everyday life.* New York: Norton.

Dym, C. L. (1994). *Engineering design: A synthesis of views.* Cambridge, UK: Cambridge University Press.

Earl, P. E. (1983). *The corporate imagination.* Armonk, NY: M.E. Sharpe.

Hogarth, R. M., and Reder, M. W., eds. (1986). *Rational choice: The contrast between economics and psychology.* Chicago, IL: University of Chicago Press.

Kleindorfer, P. R., Kunreuther, H. C., and Schoemaker, P. J. H. (1993). *Decision sciences: An integrative perspective.* Cambridge, UK: Cambridge University Press.

Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M. (1987). *Scientific discovery: Computational explorations of the creative process.* Cambridge, MA: The MIT Press.

March, J. G., and Simon, H. A. (1993). *Organizations,* 2nd ed. Oxford, UK: Blackwell.

Newell, A., and Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice Hall.

Rumelhart, D. E., and McClelland, J. L. (1986). *Parallel distributed processing* (2 vols.). Cambridge, MA: The MIT Press.

Simon, H. A. (1997). *Administrative behavior,* 4th ed. New York: Free Press, Macmillan.

Simon, H. A. (1996). *The sciences of the artificial,* 3rd ed. Cambridge, MA: MIT Press.

Smith, V. L. (1991) *Papers in experimental economics.* Cambridge, UK: Cambridge University Press.

von Neumann, J., and Morgenstern, O. (1944). *Theory of games and economic behavior.* Princeton, NJ: Princeton University Press.

Williamson, O. E. (1985). *The economic institutions of captialism.* New York: The Free Press.

# Desktop Publishing

**Reza Azarmsa**
*Loyola Marymount University*

## GLOSSARY

**alignment** The way text lines up on a page or in a column.

**ascender** The portion of a lowercase letter that rises above the main body or x height as in a "b."

**baseline** The imaginary line on which a line of type rests.

**bullet** A large, solid dot preceding text to add emphasis. Also known as a blob.

**color separation** The division of a multicolored original into the primary process colors of yellow, magenta, cyan, and black. A separate film is made for each color and these are each printed in turn, thus building up a color picture.

**descender** The portion of a letter that extends below the baseline, as in the letter y.

**dots per inch (dpi)** The measure of resolution for a video monitor or printer. High-resolution printers are usually at least 1000 dpi. Laser printers typically have a resolution of 300 dpi; monitors are usually 72 dpi.

**drop cap** A large initial letter at the beginning of the text that drops into the line or lines of text below.

**Encapsulated PostScript (EPS)** A file format that enables you to print line art with smooth (rather than jagged) edges and to see and resize the graphic on screen as it will print. These files can be produced in graphic programs that produce PostScript code (such as Illustrator and Free-Hand). The EPS images do not print well on non-PostScript printers.

**grid** A nonprinting design consisting of intersecting horizontal and vertical lines that can be used for the overall layout of a publication or for positioning drawn graphics, depending on the software. Also called layout grid.

**initial cap** Large, capital letters (often ornamental) that are found at the beginning of paragraphs or chapters. These date back to European manuscripts, where they were (and still are) considered works of art. Before printing presses replaced hand lettering, a few talented scribes drew the characters into spaces left in the manuscripts for that purpose.

**layout** The arrangement of text and graphics on a page.

**leading** (Pronounced "ledding.") The distance in points from the baseline of one line of type to the baseline of the next. Also called line spacing or interline spacing.

**phototypesetting** The method of setting type photographically.

**pica** A typographic unit of measurement equal to one-sixth of an inch. Twelve points equal one pica.

**pixel** (stands for picture element). Pixels are square dots that represent the smallest units displayed on a computer screen. Characters or graphics are created by turning pixels on or off.

**point** A basic unit of typographic measurement. There are 72 points to an inch.

**PostScript** Adobe Systems' page description language. Programs such as FreeHand use PostScript to create complex pages including text and graphics onscreen. This language is then sent to the printer to produce high-quality printed text and graphics.

**resolution** Sharpness of definition of a digitized image depending on the number of scan lines to the inch.

**reverse lines** White rules on a contrasting black, shaded, or colored background.

**sans serif** Typefaces that do not have the small "finishing" lines at the end of each stroke of a letter.

**serif** Typefaces that have small "finishing" lines at the end of each stroke of a letter.

**template** A dummy publication that acts as a model, providing the structure and general layout for another similar publication.

**tracking** The overall letterspacing in the text. Tracking can also be used to tighten or loosen a block of type. Some programs have automatic tracking options that can add or remove small increments of space between the characters.

**True Type** An outline font format developed by Apple Computer Corporation and adopted by Microsoft Corporation. These fonts can be used for both the screen display and printing, thereby eliminating the need to have font files for each typeface.

**typesetting** Text produced by a laser printer or high-quality machine known as a typesetter.

**weight** The degree of boldness or thickness of a letter or font.

**widow** A single line of text or word at the top of a page or column.

**WYSIWYG (what you see is what you get)** A display mode that enables the user to see an image on screen of how the text and graphics in a publication will appear on the printed page.

**x-height** The vertical height of a lowercase x and the height of the bodies of all lowercase letters. Also called mean line.

## I. AN OVERVIEW OF DESKTOP PUBLISHING

Desktop publishing (DTP) is a phenomenon of desktop computing. Very few developments in microcomputers have grabbed the attention of computer users quite like DTP. It began to be widespread when Apple Computer announced the LaserWriter laser printer in January 1985.

The term desktop publishing is quite recent. Paul Brainerd, "father" of the PageMaker software program, is credited with coining the phrase. Simply, DTP refers to the process of producing documents of typeset quality using equipment that can sit on a desktop. Users can combine text and illustrations (both graphics and photographs) in a document without the traditional paste-up procedure often used in print shops, at the cost for less than traditional offset printing.

Desktop publishing systems use the principle known as WYSIWYG (pronounced wizzy-wig): What you see is what you get. As the user keys text and graphics into the computer, he or she can see exactly how the output will appear on the page before printing the final copy.

Another kind of software that contributed importantly to the new publishing process was the page description language PostScript. The PostScript language, used by laser printers for page description, makes these printers versatile type compositors. The PostScript interpreter allows the page composition language that controls the printer to combine text, drawings, and photographs for printing.

Desktop publishing systems, including a microcomputer, a laser printer, word processing software, a page layout program, and graphic arts software, make it easy to communicate graphically as well as textually. Research indicates that textual communication is processed primarily in the left lobe of the brain and graphic communication is processed primarily in the right lobe. The combination of textual and graphic elements produces a powerful communications structure that facilitates simultaneous multichannel processing by both the left and right lobes of the brain. As a result, communication techniques that combine both textual and graphic elements produce effective results.

Research indicates that most people read typeset documents about 27% faster than they read nontypeset documents. They also tend to view typeset documents as more credible, more persuasive, and more professional than nontypeset documents.

Desktop publishing is changing the way people produce documents. It enhances the individual's ability to control all or most of the process of producing printed materials. This potential means that multitalented people with writing, illustrating, editing, and designing skills can do the work effectively.

Users gain knowledge in a variety of disciplines—writing, typesetting, graphic design, printing, and computing. Controlling the publication process permits authors to become more involved with the visual impact of their ideas on the reader. The relationship between form and content may take on new meaning when authors integrate ideas with words, type style, graphics, and the other features involved in the production of publications with a high level of visual impact. Creating visually informative text gives the author a chance to gain a heightened sense of categories, divisions, and orderly progression. Also affected are small businesses, designers, and large corporations that have brought the production process in-house.

## II. UNDERSTANDING DESKTOP PUBLISHING

DTP provides the capability to produce reader- or camera-ready originals without the need for complicated prepress operations. The tools of production generally consist of a computer system and relevant software applications and are sufficiently compact to fit on a desktop. They provide the user with total control over the content and form of the publication. In general, prerequisite skills include a working knowledge of word processing, a general understanding of graphics, and a sense of what constitutes pleasing and functional layout and design.

The significance of DTP is threefold. First, it provides the user with the tools to express thoughts and ideas in text, graphic, and sometimes photographic (halftone) form. These elements are pieces of the publication production process that have traditionally required the services of trade professionals to assemble into final form. Second, the user has immediate feedback on the appearance of the final publication. The page relationships of the elements, their sizes, and their physical characteristics are immediately visible and infinitely editable. Third, the paper output from the system is in finished form, ready for distribution to small numbers of readers or for further reproduction and subsequent mass distribution.

## III. THE PROCESS OF DESKTOP PUBLISHING

The DTP production process involves electronically placing text and graphics on a page and reproducing the page for distribution. Produce a newsletter, a proposal, a resume, or a high school yearbook and you follow the same procedures as publishers of books and magazines, albeit on a less elaborate scale. No matter the kind of document produced or the technique used, publishing procedure follows five basic steps:

1. Plan the publication
2. Prepare the text
3. Prepare the charts and illustrations
4. Make up the publication's pages (prepare them for reproduction)
5. Reproduce the publication

Desktop publishing computerizes and accelerates the publishing cycle. Often text is created in a software application and then edited. Charts, graphs, illustrations, rules (lines), boxes, or circles are drawn either with DTP built-in tools or by other application software. Text and charts are placed on the page and resized and reshaped according to the predetermined page specifications. When the page is complete, it may be produced on a computer printer and then can be reproduced on a photocopy machine or a commercial printing press. Figure 1 illustrates the process of desktop publishing.

## IV. MATERIALS PRODUCED BY DESKTOP PUBLISHING

Desktop publishing has enjoyed quick acceptance in most areas of document production within business, industry, government, and education. It is a step beyond word processing, providing users with the capability to produce better looking and more informationally potent documents.

The following is a list of some of the materials that can be produced using desktop publishing.

Books
Brochures
Bulletins
Calendars
Catalogs
Certificates
Charts
Cover sheets
Curriculum materials
Diplomas
Directories
Documentation
Flow charts
Flyers
Graphs
Greeting cards
Indexes
Instructional materials
Letterhead stationery
Magazines
Manuals
Maps
Memos
Menus
Name tags
Newsletters
Newspapers
Notices
Pamphlets
Poetry
Programs
Proposals
Prospectuses
Resumes
Schedules
Signs
Slides
Statements
Tabloids
Testing materials
Title pages
Visual aids

## V. BENEFITS OF DESKTOP PUBLISHING

Desktop publishing offers many advantages over the traditional printing process. Even though the initial costs of obtaining high-end DTP hardware and software

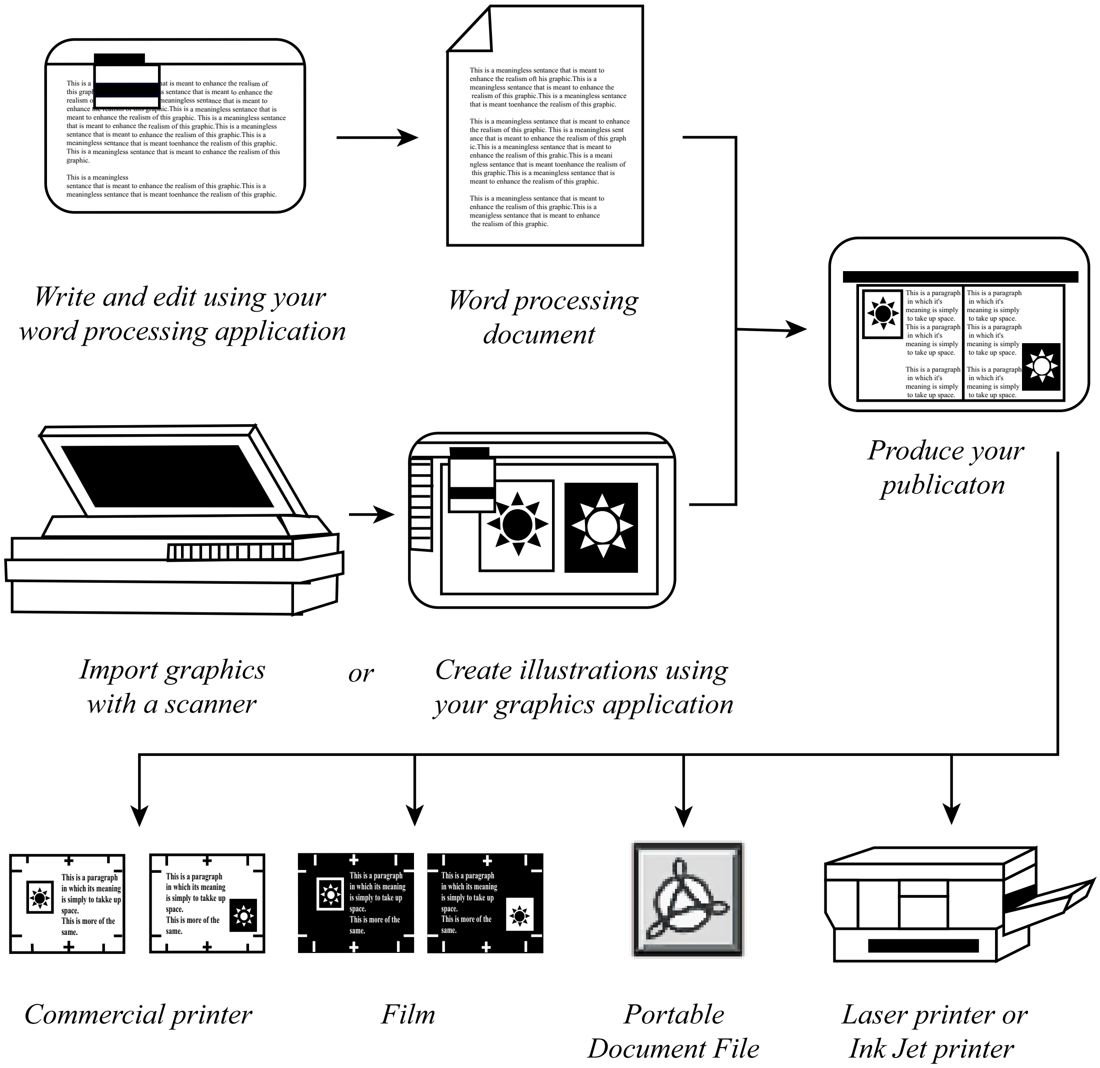

Write and edit using your word processing application
Word processing document
Produce your publicaton
Import graphics with a scanner
or
Create illustrations using your graphics application
Commercial printer
Film
Portable Document File
Laser printer or Ink Jet printer

**Figure 1** Process of DTP. The DTP program provides a tool with which users can assemble text and illustrations into a page format.

are high, the system actually often saves money. Usually, the sense of accomplishment that comes with the finished product and the flexibility regarding the printing process are enough to justify that time. Additional potential benefits to desktop publishing include:

1. Reduced typesetting time and costs
2. Better looking publications with improved readability, credibility, and prestige
3. More control over the publishing process
4. Increased convenience to designers
5. Increased cooperation among students
6. Possibility for electronic distribution

Reduced typesetting time and costs are a very tangible benefit of DTP. Because word processing and page composition programs allow you to directly create and lay out your documents, there is no need for a manual typesetting step. This eliminates typesetting costs and time. Typesetting can often take days and cost as much as $50 per page. For producing newsletters or brochures on a regular basis, typesetting costs and time can be a significant problem.

Desktop publishing can result in better quality publications because documents can be composed quickly, allowing more time for revisions and additions. Many documents that would not merit the time and expense of printing by traditional methods can be produced in near typeset form with DTP equipment. If documents are created using a word processing program, it is a small additional step to compose and print them.

More control is achieved over the publishing process because the results are immediately available.

This makes it much easier to try out ideas and see what various options look like. In addition, potential communication problems between the document designer and the printer are avoided because they often are the same person.

The availability of graphics and page composition programs makes design work much easier and faster. For example, a skilled artist can create technical illustrations much more productively using a computer than on paper. This is because it is possible to build up libraries of "clip art" that can be reused. Features such as automatic rescaling, rotating, and overlaying make the drawing process faster. Similarly, page layout can be accomplished much easier than manual paste-up due to features such as automatic pagination, line-spacing control, text wraparound, automatic hyphenation, and margin settings.

Finally, it is possible to send and exchange documents electronically, eliminating the time and costs of physical delivery. Suppose you are editing a newsletter that contains contributions from people all over the country. The articles can be transmitted to you from their personal computers to yours in a matter of minutes. You can then edit the articles, compose the newsletter, and transmit the complete newsletter to the printing site. Alternatively, the newsletter could be published electronically on the Internet, meaning that people read it on their own computers as soon as it is made available.

Desktop publishing creates interesting written communications far beyond what can be created with an ordinary word processor. Some of the advantages of a DTP program are as follows:

1. Improves the appearance of documents
2. Reduces the time and number of steps required to print pages
3. Produces customized documents
4. Produces camera-ready documents in a short time
5. Reduces production costs
6. Offers more capabilities and flexibilities
7. Encourages creativity
8. Increases productivity
9. Makes the job more fun

## VI. EVOLUTION OF DESKTOP PUBLISHING

A review of typesetting and printing advances offers a good starting point for understanding how DTP builds on and streamlines traditional publishing methods.

### A. Ancient Times

Publishing coincided with the development of written language. In ancient times, papyrus was used to produce printed material. Later, around the time of Jesus, the Chinese used bamboo to create a writing surface. Before the invention of the printing press, publications were hand-copied on pieces of paper or other writing surfaces. The first form of printing involved carving a message on a piece of stone or wood, coating it with some type of ink, and pressing that "plate" against a piece of paper. This method was mostly used for reproducing artwork rather than for printing characters.

### B. Movable Type

In the fifteenth century, Johannes Gutenberg's popularization of movable type became one of the milestones of European history—it reduced publishing time and expense so books became more plentiful. Until then, books had been handcopied or printed from blocks of wood on which raised characters (letters, numbers, and punctuation marks) and illustrations were laboriously carved for each page. Movable type at that time involved small wooden blocks with individual characters that could be rearranged and used repeatedly. This wood type was later replaced by metal type for greater durability.

### C. Linotype

The next important publishing innovation did not occur for four centuries. With the invention of the Linotype machine in the 1880s, publishers could set an entire line of type at once rather than only character by character. Although the process entailed melting lead, it reduced publishing costs and production time.

### D. Offset Printing and Phototypesetting

In the 1950s and 1960s, the publishing process transformed from the "hot type" process involving molten lead into a photographic "cold type" process that is faster and cleaner. At this time, several photographic printing processes were introduced that used a flat plate rather than the raised type of earlier methods. The most important new process was offset printing, which is widely used today. Offset printing was soon joined by computer-based phototypesetting, which projects images of columns of text onto photosensitive paper. The paper is developed, cut into sections, and pasted onto a sheet of heavy paper to produce a camera-ready page mechanical—a paste-up of the complete page for offset printing. Illustrations are sized photographically and added as needed.

By the early 1980s, staff overhead presented a problem for a growing segment of the publishing industry: businesses, government offices, and educational institutions that had become major publishers of brochures, books, catalogs, and other phototypeset publications. The advent of microcomputers and software development introduced DTP in 1985. This new development gave businesses and educational institutions an affordable publishing system anyone could learn to use. For some, bringing publishing to the desktop is known as the second publishing revolution.

## E. Desktop Publishing

The term desktop publishing is credited to Paul Brainerd of Aldus Corporation, the father of PageMaker. Desktop publishing actually originated in 1973, when an experimental multifunction workstation was developed at the Xerox Palo Alto Research Center (PARC). The workstation was called Alto. Because its designers had graphics applications in mind, the Alto had a high-resolution bit-mapped display screen and a mouse pointing device.

Altos were assigned to a few selected sites, including the White House, the House of Representatives, the Senate, a few universities (especially Stanford), and several undisclosed locations in the United States and Europe. This large base of users provided input as to what was right and what was wrong with the Alto and with the various software packages that had been developed for it. These packages included Bravo, Gypsy, Markup, Draw, SIL, and Laurel for text editing and formatting, creating pictures and diagrams, and providing electronic mail by means of a central file server.

None of the Alto machines were ever commercially viable on cost grounds, and there was no affordable printer. Apple Computers, inspired by the PARC systems, designed and marketed the Lisa desktop microcomputer. This unit, released by Apple in 1983, had a high-resolution screen, a mouse, a WIMP (windows, icons, mice, and pointers) interface, and high-capacity hard and floppy drives. Lisa was discontinued in 1984 when Apple produced the first Macintosh, which led to a family of microcomputers that further extended the idea of an integrated suite of applications emulating the original PARC systems. The following year Apple produced a laser printer with an output resolution of 300 dots per inch (dpi), which is generally considered to have started the DTP revolution. This printer had a plain-paper typesetter using a raster image processor (RIP), which gave a resolution approximating professionally published documents. At 300 dpi a letter-size page requires about 1MB of storage to be set up in the RIP for printing manipulations.

The fact is that the new technology has spawned a host of desktop publishers who are publishing—writing, producing, and distributing—to a readership, and the publishing process has suddenly become democratized and affordable beyond the dreams of only a few years ago.

Macintosh computers get credit for desktop publishing as we perceive it today. But today DTP is not limited to Macs. Windows-based computers are, in some respects, an ideal platform for the development of a DTP system because they are popular, powerful, and open. Using a Windows-based computer, the user must specify what kind of graphics board is available.

Until recently, the interval between the development of new technologies and their introduction into printing has been quite considerable. In the case of DTP, where some of the essential tools, such as text and graphics processing, are already providing effective publishing experiences, the interval has been extremely short. Expectations have already been raised by the early release of packages that claim to be DTP systems because they provide simple manipulations of both text and graphics on a single page. A particular selling point of such systems has been their facility to design pages of newspapers and magazines for delivery on the Internet system.

## F. PostScript

PostScript is a computer graphics language that describes pages of text and graphics and reproduces the descriptions on a printer, typesetting machine, or other output device. PostScript was developed by Adobe Systems of Palo Alto, CA, for Apple's LaserWriter printer. PostScript is a widely accepted page description language for the Macintosh and other computers, such as the IBM PC series and compatibles.

PostScript describes pages by using mathematical formulas that represent shapes rather than by specifying individual pixels in a bit-mapped graphic image (the small dots that make up an image are known as picture elements, or pixels). PostScript translates images into the tiny dots that make up text, graphics, and halftones on printed pages. PostScript encodes typefaces into outlines that a laser printer reconstructs at the proper size and then fills in to solidify the outline. This approach has two clear advantages: computer memory is conserved, and many different types of output devices can recreate the standard PostScript format at a wide range of resolutions.

PostScript's unique system of typeface definition creates characters in a wide variety of type styles and sizes. This approach can cut typesetting costs considerably, because typefaces are traditionally sold by the font (a set of characters of a single typeface, type style, and size). In other words, most typesetting machines require separate purchases for 9-point Times Roman, 12-point Times Roman, 12-point Times Roman italic, and so forth. On the other hand, a single purchase gets you the Times Roman typeface for a PostScript system in all the major type styles—plain text, italic, bold, underline, outline, shadow, small caps, superscript, and subscript—and in virtually any type size (the Apple LaserWriter, for example, can produce any size from 3 points up). You don't pay separately for each variation. It is also important to keep in mind that PostScript doesn't work only on printers. For example, you can write PostScript programs to create special graphics effects. Computers can use PostScript to describe text and graphics in a standardized common format that other computers or PostScript programmers can employ. Video displays can use PostScript to interpret page descriptions, convert the results into bit-mapped images, and display the images on a computer screen. PostScript provides a common language for various audiences and devices that precisely and concisely describes pages of information that contain text, graphics, or both (Fig. 2).

PostScript has emerged as a standard among page description languages; many hardware and software companies now make PostScript-compatible products from page layout and graphics software for the Macintosh and IBM PC to page printers and typesetting machines.

## VII. TEX AND LATEX

TeX is a typesetting language developed by Donald E. Kunth and is designed to produce a large range of documents typeset to extremely high-quality standards. For various reasons (e.g., quality, portability, stability, and availability) TeX spread very rapidly and can now be best described as a worldwide de facto standard for high quality typesetting. Its use is particularly common in specialized areas, such as technical documents of various kinds, and for multilingual requirements. The TeX system is fully programmable.

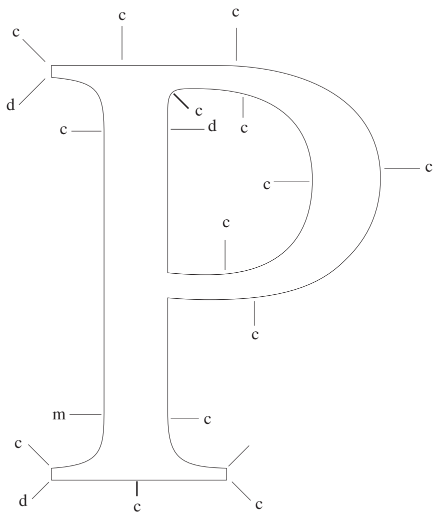


**Figure 2** PostScript outline of the letter P.

LaTeX is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents, but it can be used for almost any form of publishing. LaTeX is not a word processor. The user must write in the commands that control the layout and appearance of the text. LaTeX is based on Donald E. Kunth's TeX typesetting language. It was first developed in 1985 by Leslie Lamport, and is now being maintained and developed by the LaTeX3 project. LaTeX is available for free by anonymous ftp.

## VIII. PORTABLE DOCUMENT FORMAT

Portable Document Format (PDF) is designed to be totally cross platform. The resulting PDF files can be viewed and printed from several different platforms (Mac or PC) with the page layout and typography of the original document intact using Acrobat Reader, as well as on other systems, including OS/2 and UNIX.

## IX. ADOBE TYPE MANAGER

Adobe Type Manager (ATM) is a system software component that automatically generates high-quality screen font bitmaps from PostScript or OpenType outline font data. With ATM, you can scale your fonts without the characters appearing jagged, and you can also enable "font smoothing," which further improves the appearance of your fonts on-screen by using your computer monitor's color palette to intelligently improve the rendering of characters. Also ATM allows you to print PostScript fonts on non-PostScript printers.

## X. COMPARISON OF CONVENTIONAL AND ELECTRONIC METHODS

In DTP process, every step in the design and production, from conception to master copy, to short-run production, can be accomplished electronically from a desktop computer. Taken individually, each step in the electronic production cycle is direct and well-defined. Text can be electronically formatted in typographic detail; graphics created, or scanned; and all these pieces can be assembled in DTP software.

### A. Text Preparation

**Conventional method**—Creating handwritten copy or composing copy on a typewriter requires very little training or specialized computer knowledge, but there is little flexibility or editing capability, and it is difficult to incorporate into a design process. The typewriters of today come with microprocessors, create disk files, and include such automatic editing or authoring tools as spell checkers. They just don't require the user to be computer literate.

**Electronic method**—Text prepared in a word processor on a computer is easy to edit, makes use of automated text tools, and incorporates into a page composition scheme. Specialized computer knowledge of operating systems and file structures is required, however, the strong advantages word processors offer compel their use.

### B. Painting or Drawing

**Conventional method**—Paper and pen or pencil are the traditional media of artists. Advantages include low cost, ease of use, and many special effects that are difficult to achieve on a computer. Contrarily, artwork is not easily altered, is not very precise, and is hard to adapt to a production process.

**Electronic method**—Computer paint and draw programs with digital file output are easy to alter, include many special effects, are precise, and adapt easily to an electronic production cycle. Disadvantages include cost of equipment and lengthy rendering times. Electronic art is more frequently becoming the choice for a production process, but not all artists are welcoming the advantages of electronic art forms.

### C. Photographic Images

**Conventional method**—Camera-generated artwork is low-cost, high-quality, easy to generate, and can incorporate many special effects. But it is difficult to alter and include into an electronic production process. Traditional photography is preferred in all situations except where moderate quality is acceptable or where there is access to very high quality computer equipment.

**Electronic method**—Using scanned artwork you can alter an image and create certain special effects not offered by film. Also, digital cameras and royalty-free photographs are readily available. Disadvantages are training and highly varying quality, depending on equipment.

### D. Layout, Typesetting, and Composition

**Conventional method**—Paste-up boards, if in use, result in a high-resolution master. However, they are time-consuming and expensive to produce.

**Electronic method**—Page layout programs have the support of automation tools, offer device independence, and are low cost and of good quality. Disadvantages include complex design strategy and lengthy setup times for initial run. Desktop page layout is a better choice for composing a page, particularly for repetitive situations.

### E. Printing Process

**Conventional method**—Letterpress, offset, gravure, screen printing, and heat-transfer printing offer high quality and low cost per unit, give better color reproduction, and offer large page sizes. Disadvantages include high setup costs and time and low adaptability for mixed print runs.

**Electronic method**—Personal laser printers, inkjet printers, and imagesetters are adaptable and have short setup times. Quality is variable but can be from good (laser printers or copiers) to excellent (for imagesetters).

## XI. PUBLISHING CATEGORIES

Publishing can be divided into four distinct areas: business publishing, periodical publishing, book publishing, and personal publishing. All four areas can benefit from DTP technology.

*Business publishing* involves the printing and distribution of company materials. It includes business reports, brochures, catalogs, product documentation, forms, letterheads, and corporate communications such as internal memos, advertising and promotional materials, and other items.

*Periodical publishing* is the printing and distribution of materials on a regular basis. The materials are usually distributed to the same group of individuals or a similar group at an interval that can be daily, as in the case of newspapers; weekly, as in the case of newsletters and magazines; monthly, as in the case of magazines and journals; or even annually, as in the case with some journals and reviews.

Periodical publishing also involves a coherent content and usually similar look from issue to issue. What distinguishes book publishing from other types of publishing is that it involves the printing and distribution of a single, cohesive work. Book publishing's production time is usually longer than that of other types of publications.

*Personal publishing* is the printing and distribution of materials by individuals. The materials can include poetry, freelance writing, essays, reports, humor, school papers, greeting cards, wedding and birth announcements; party invitations, personalized calendars, etc. Personal publishing also encompasses artistic uses of publishing such as the publication of poetry. Often personal publishing overlaps with another type of publishing, as in the case of people who publish a book on their own or who send a periodical such as a newsletter to friends, members of an organization, or people with a common interest. The production of essays, reports, and other such documents is another area of personal publishing that benefits from DTP technology. The option of publishing your own reports can be an excellent method for dissemination of scientific information, especially in the publish-or-perish academic world. People in the personal category of publishing usually publish for their own benefit—financial gain and commercial success are not usually the prime motivations.

## XII. DESIGN ELEMENTS IN DESKTOP PUBLISHING

The success of many DTP materials can be attributed in large measure to the quality and effectiveness of the graphic design. These are achieved through organizing preliminary thoughts, planning, and applying the techniques outlined in this section.

Many people who develop DTP materials have little or no professional art background. To avoid producing poor graphic materials, they can consider a number of practical suggestions and guiding principles and then apply these as the need arises.

## XIII. DESIGN PRINCIPLES

Effective graphic design is based on knowing the building blocks of design and when to use them. In graphic design, everything is relative. Tools and techniques that work in one situation will not necessarily work in another. For example, it is impossible to define the exact layout for informal balance or the appropriate color for a newsletter or book cover.

In creating a visual, important design considerations are best faced by starting with a preliminary sketch of the intended visual. This is referred to as a rough layout. At the rough layout stage, little attention is paid to rendering the artistic details, but careful consideration is given to effective design.

Graphic design must be seen as a means of communication rather than mere decoration. There should be a logical reason for the way you employ every graphic tool. That tool should relate to the idea it expresses as well as to the environment in which the final product will appear. The following guidelines provide a framework for effective graphic design.

### A. Planning

Design must be planned with consideration for the intended audience. What is the basic message, and what format will be used (newspaper, bulletin, poster)? The more you define your project's purpose and environment, the better you will do. You should also consider the size and dimensions of the publication in addition to the resources, skills, techniques, materials, and facilities that you can employ to produce the publication.

It is useful to develop a grid or choose a predesigned one in template form. Shown in Fig. 3, the grid adds continuity to a publication by defining where margins, rules, columns, and other elements should be on each page. Publications may be designed by drawing thumbnail sketches. A thumbnail sketch is a rough sketch of a page design. This technique helps designers visualize the final product.

### B. Organization

The visual and verbal elements of the design should be arranged in a pattern that captures the viewer's

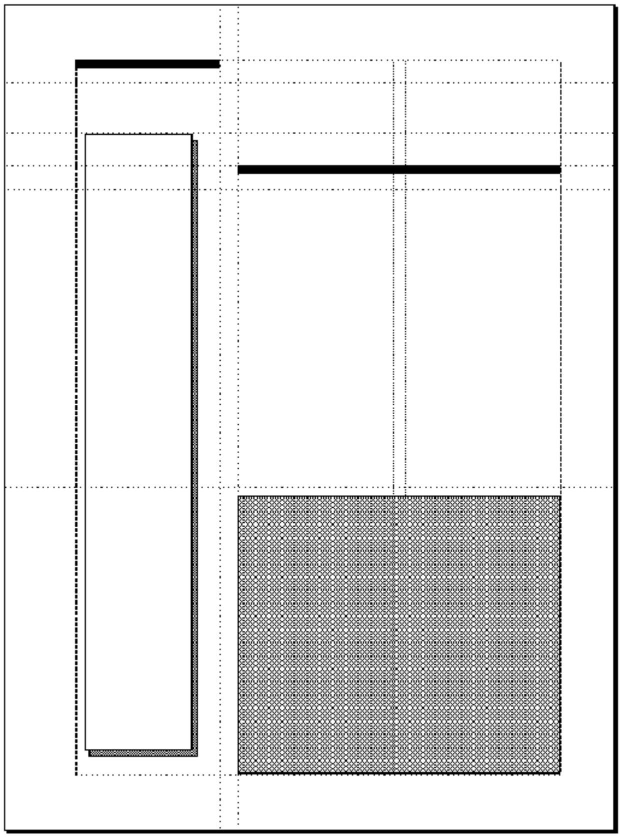


**Figure 3** A grid contributes to the continuity of a publication.

attention and directs it toward the relevant details. The manipulation of line and space are the designer's primary tools. The arrangement should be clear enough to attract and focus attention quickly. A configuration pattern is usually found in effective design. It may be established by the directional cues that are developed to guide the viewer to see details in proper sequence. Certain patterns guide the viewer's eye throughout the publication.

## C. Simplicity

Generally speaking, the fewer elements in the design, the more pleasing it is to the eye. Simplicity is the first rule of design. Drawings should be bold, simple, and contain only key details. Picture symbols should be outlined. Use simple, easy-to-read lettering systems and a minimum of different type styles in the same visual or series of visuals.

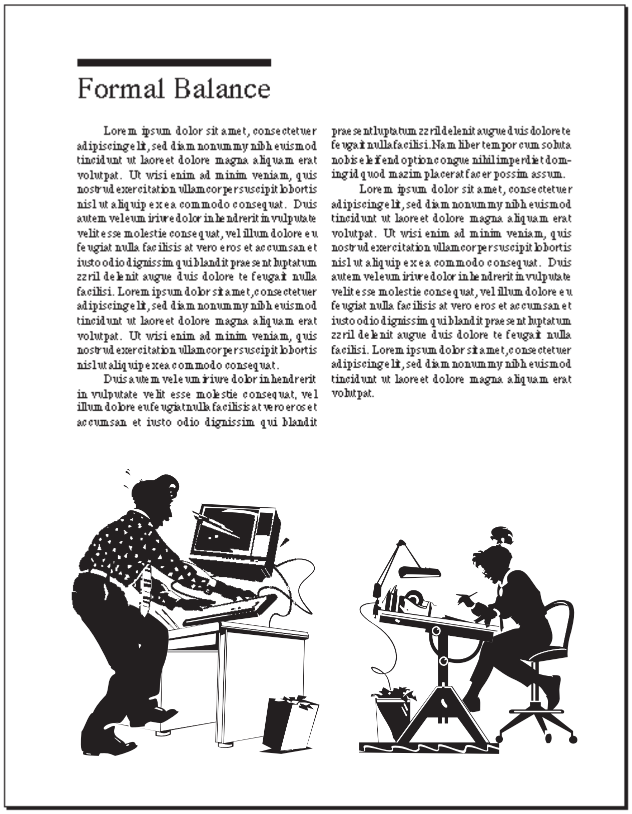
## D. Balance

A psychological sense of equilibrium or balance is achieved when the design elements in a display are equally distributed on each side of the axis—either horizontally, vertically, or both. There are two kinds of balance: formal and informal. When the design is repeated on both sides, the balance is formal or symmetrical.

*Informal balance* is asymmetrical; the elements create an equilibrium without being alike. Informal is dynamic and attention-getting, unlike *formal balance*. It requires more imagination from the designer. Informal balance is usually regarded as the more interesting choice (Figs. 4 and 5).

## E. Unity

Unity is the relationship that exists among the elements of a visual. These elements function together to provide a single dominant visual to capture the reader's attention. Unity can be achieved by overlapping ele-



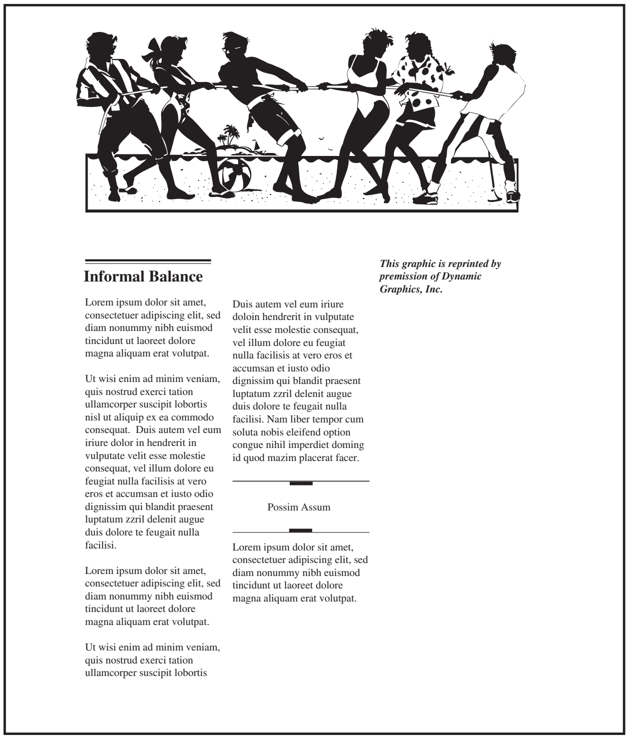
Formal Balance

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

**Figure 4** An example of formal balance. Formal balance, if used too much, becomes monotonous.



Informal Balance

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.  Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis

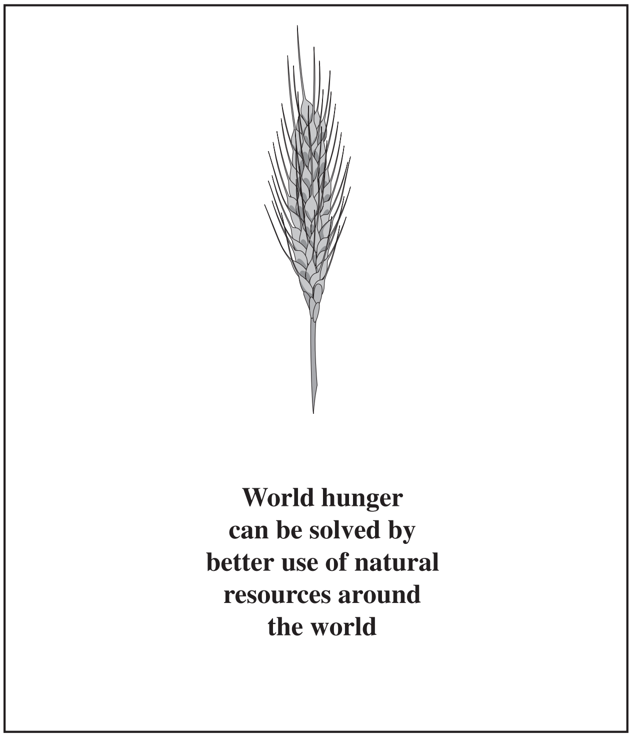
Duis autem vel eum iriure doloin hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer.

Possim Assum

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

*This graphic is reprinted by premission of Dynamic Graphics, Inc.*

**Figure 5** An example of informal balance. There are an infinite number of possible designs in informal composition.



**World hunger**
**can be solved by**
**better use of natural**
**resources around**
**the world**

**Figure 6** Using white space for emphasis.

ments, by using pointing devices such as arrows, and by employing visual tools (lines, shape, color, texture, and space). The single dominant visual helps organize the reader's eye movement throughout the publication.

## F. Emphasis

Through the use of size, relationships, perspective, lines, and such visual tools as color and space, emphasis can be given to the most important elements in the publication (Fig. 6).

## G. Contrast

Any element that is different from those surrounding it will tend to stand out. The contrast or variation may be in terms of size, shape, color, or orientation. Contrast also refers to the relative amount of space devoted to text, artwork, and white space. You can create a high-impact publication with definite light and dark areas as well as lots of white space and illustrations.

# XIV. COLOR

Color is an important adjunct to most visuals. Careful use of color plays an important role: (1) to heighten the realism of the image by depicting its actual colors; (2) to point out similarities and differences and to highlight important emphasis; and (3) to create a particular emotional response.

Contemporary research in the psychology of motivation reveals that different colors stimulate more than the visual sense. They have "taste", for example, blue is sweet, orange is edible. They have "smell"; pink, lavender, yellow, and green smell best. Colors also have psychological connotations; dark red and brown evoke masculine images of earth, wood, and leather; gold, silver, and black suggest the prestige and status associated with wealth.

When selecting colors for visual materials, attention should be given to their elements. The hue, which is the specific color (red, blue, etc.), should be considered. Another element is the value of the color, meaning how light or dark the color should appear in the visual with relation to other visual elements. The final component is the intensity or strength of the color for its impact or coordinated effect.

## A. Using Color

Color can be the most important part of presentation design. Used carefully, it can enhance your message, provide richness and depth, and put a personal stamp on your work. The most obvious reason for using color is to show things as we see them in nature—green trees, yellow bananas, and red bricks, for example. Abstractions, such as statistics, ideas, and proposals have no intrinsic colors, but color can be used to represent symbolic associations. For instance, red can suggest warning, danger, or financial loss. The following points are some reasons for using color.

1. Advertisers try to establish associations of certain colors with consumer products, sports teams, etc. You can apply the same logic by developing identifying color schemes for your presentation.
2. Use color to distinguish between like and unlike elements. To clarify a flow chart, show file names in red and program names in blue. Color can distinguish elements or classes of information.
3. Indicate the importance or progression of data by increasing the value and saturation level. Light-to-dark or gray-to-bright sequences are excellent ways to represent levels of importance. Use a chromatic or a rainbow series to show a graduated sequence.
4. You can draw attention to elements in your presentation by choosing colors that are either brighter or lighter than the rest. Suppose you want to emphasize some of the words in a list. On a dark gray, blue, or black background, use light yellow or cream as your main text color. For contrast, select full yellow (brighter) or white (lighter) as the emphasis color. Do not try to emphasize too much at once with several different colors; they may cancel each other out.
5. Be sensitive to cultural biases as well. Some people cannot accept pink as a serious color. That does not mean that pinks are not valid presentation colors. Pink is actually a serviceable color, but be aware of personal prejudices.
6. The setting, the subject matter, and the audience's previous experience can create certain expectations. Weigh all factors involved and decide whether audience expectation is reason enough to develop a color presentation.
7. Do not sacrifice readability for pleasing color. Legibility takes precedence over all else in presentation materials, and poor color choices can interfere. Colors do not perform the same way under all conditions: pure blue on dark backgrounds is extremely hard to read, but blue on white is fine. This isn't an opinion; it is a fact of human vision. Base your color choices on what works well for the audience rather than on personal taste.

## B. The Color Wheel

You are probably familiar with the color wheel showing the range of colors and their relationships to each other (Fig. 7). Primary and secondary colors alternate to form the circle. We are used to thinking of red, blue, and yellow as the primary colors and green, orange, and purple as the secondary colors. Complementary colors are ranged opposite each other on the color wheel. The three sets of complements each pair a primary with a secondary color.

In graphics, there are two methods of mixing colors. The first mixes colors of projected light on your computer screen, for example. The second method mixes colors of pigment, like the ink on a printed page. The differences between the two methods make it very difficult to match what you see on the screen with "hard copy" printed on paper or film.

Whether it is projected light or an opaque substance, a color can be described by its properties or qualities—its hue, value (or lightness), and saturation.

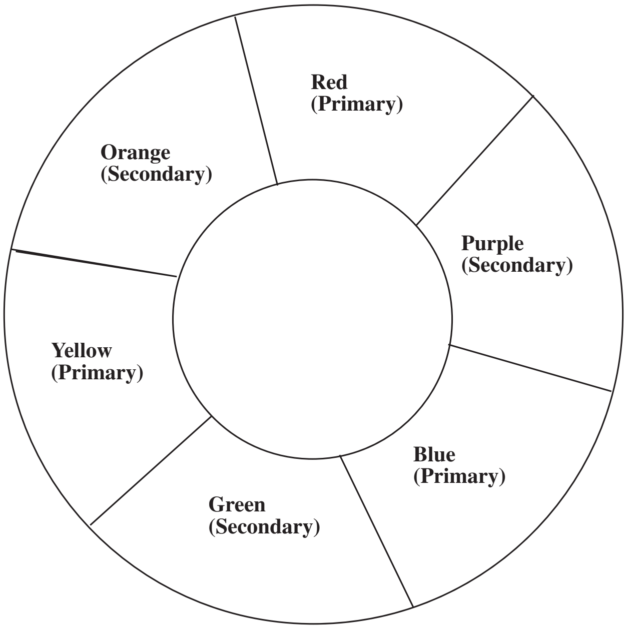


**Figure 7** Color wheel.

Each color within the spectrum or around the color wheel is a hue (e.g., red or green). Value refers to the shade or the degree to which a color approaches black or white. Saturation is the intensity of a given hue.

## C. Colors on Computer

The color you see on your computer screen is created by mixing red, green, and blue (RGB) light. As more colors are added, the image approaches white; as colors are taken away, the image approaches black. The color receptors in our eyes are sensitive to the additive primaries; that is the way we perceive the light around us.

Computer color is mixed in RGB; pigment on paper or film is mixed in CYM (cyan, yellow, and magenta). If you make a color picture on screen and output it to paper, your system must translate the colors from additive to subtractive mode in order to move from colors of light to pigment colors. The translation can cause minor (or major) discrepancies between screen colors and output colors. Be prepared for this shift, first by identifying the degree to which it occurs on your particular setup, and second by choosing flexible palettes that do not depend too much on precision in order to work well. If your output is color film, you will probably notice less shift than if you are working on paper.

## D. Graduated Color

Graduated or ramped color is a special effect that allows one color to dissolve into another with no discernible break. Software products differ in regard to graduated color features. Some let the user specify two end colors; others add a color specification for the middle. Users sometimes choose the number of steps to use in creating the blend. With other systems, this is handled automatically by the software. Color can ramp horizontally, vertically, or radially or in some other way, according to individual program features.

Graduated color is an appealing, easy way to lend depth to a picture. People are attracted to the impression of process, transition, and motion associated with graduated color. Backgrounds that ramp from top to bottom suggest a horizon and the vault of the sky. Traditionally, colors have been tricky to blend by manual methods like airbrush, but computers are ideal for this purpose. Whenever you come upon an easy way to do what used to be difficult, it can seem like magic. In the excitement, you may forget your usual good judgment and go overboard. Color graduations can fall into this category.

## E. Using Black and White

Since you will not always be working in color, it is good to be skilled at telling your story in black and white, too. Aside from a few technical considerations, the principles are much the same, whether your palette is monochrome or multicolor. Contrast, figure/ground definition and readability still top the list of design goals. Black and white means shades of gray as well. The illusion of gray is created with various black-and-white patterns. These patterns range from finely spaced dots to representational repeats of brick walls, with an infinite number of intermediate choices (Fig. 8).

In commercial printing, ink density for a tint color is controlled by sandwiching a screen between the negative and printing plates. This screen stops light where the printer does not want ink to print, and lets light through where ink is desired. When a density of 50% is desired, the screen allows light through only half the image area and stops it from hitting the other

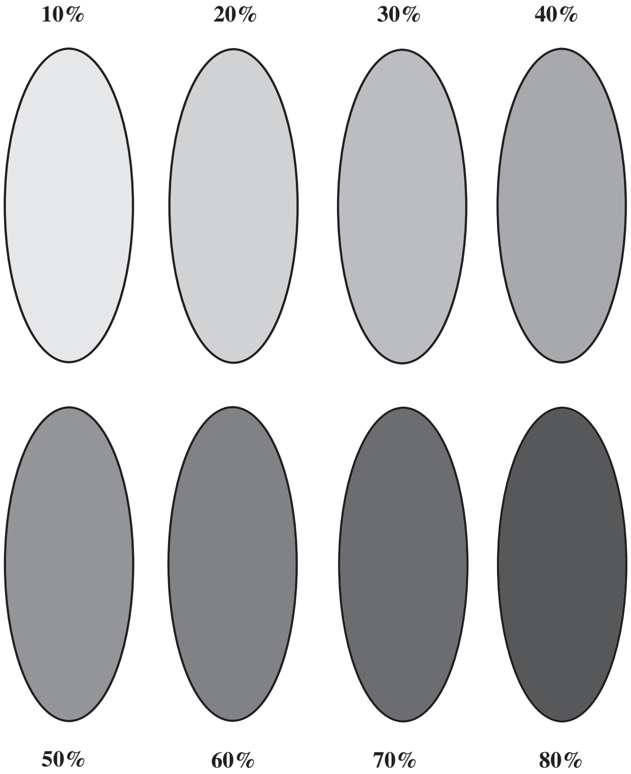


**Figure 8** An example of shades of gray.

half. The screen acts as a light stencil. Screen densities are determined by the percent of coverage; resolution is determined by the number of dpi—typically 85–150. Although this technology comes from the commercial printer's need to control ink coverage on paper, most desktop graphics programs incorporate some of these techniques for producing gray values on laser printer output.

### F. Texture

Texture is a visual element that may serve as a replacement for the sense of touch. Texture can be used to give emphasis, to create separation, or to enhance unity.

The feel for design is perhaps caught and not taught. Examine some of the graphic materials that are part of your everyday world—magazine advertisements, outdoor billboards, television titles and commercials, etc. You can find many ideas for designing your own materials by studying the arrangement of elements in such commercial displays. Imitation and practice are the best ways to develop graphic design skills.

## XV. WAYS TO IMPROVE GRAPHICS

Graphic design is creative, subjective, and personal: its functions are to inform, influence, educate, persuade, and entertain. The widespread use of laser printers and easy-to-use graphics and layout software have opened up the graphic arts to thousands of enthusiasts who otherwise might not have explored these disciplines. The advent of microcomputers and their prevalent use in education have made DTP readily available.

During its short history, DTP has changed the look of many of the things we read. Desktop publishing theoretically allows any user to become a typographer, paste-up artist, editor, writer, compositor, and graphic artist, once specialized skills practiced only by people with years of training and experience behind them. Of course, an untrained user cannot be expected to instantly acquire and use all these skills effectively. The availability of DTP must be accompanied by an attention to detail that leads to good design.

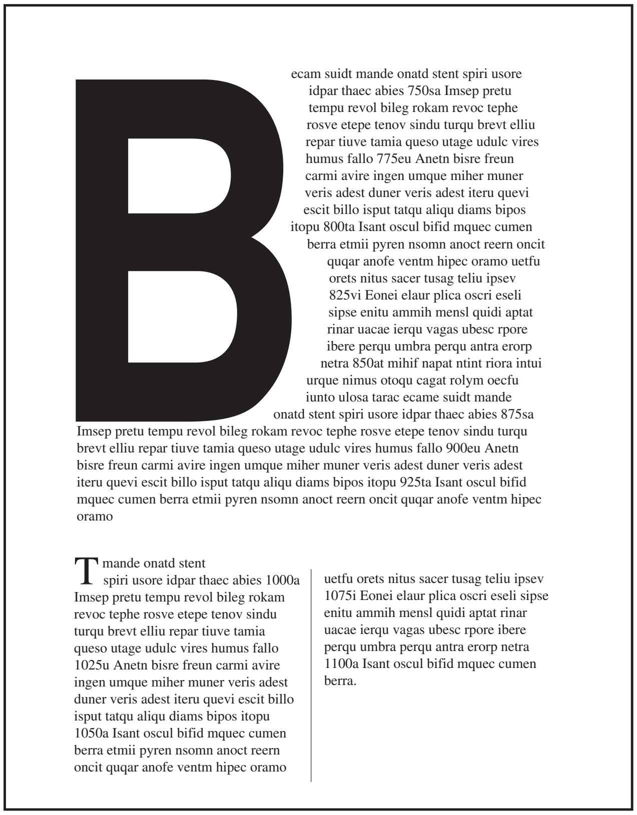
Graphics that can be used in DTP include photographs, line art (illustrations such as cartoons), digitized logos, charts, and graphs produced with your software package, and your own illustrations and diagrams produced with a paint or draw software package. Some color works such as photographs and transparencies can be introduced if the appropriate equipment is available.

Software packages exist that enable a user to become a cartoonist and illustrator as well. If you want to use cartoons and illustrations in your publication, you can use one of the available clip art packages. These contain sets of predrawn artwork, such as symbols, that can be included in your design.

Tints (screens) and boxes are a form of graphics that are an easy way to introduce variety and interest to a page. A monochrome page can be easily enhanced with the use of a 10 or 20% gray tint. Screens can be used effectively on a page that has only text and can highlight logos, headlines, and subheads or be laid over simple graphics.

### A. Get Attention with Unusual Elements

Any unusual visual elements capture the reader's attention. For example, exaggerated quotation marks draw attention to the text and an exaggerated drop cap draws the eye immediately (Fig. 9).



Becam suidt mande onatd stent spiri usore idpar thaec abies 750sa Imsep pretu tempu revol bileg rokam revoc tephe rosve etepe tenov sindu turqu brevt elliu repar tiuve tamia queso utage udulc vires humus fallo 775eu Anetn bisre freun carmi avire ingen umque miher muner veris adest duner veris adest iteru quevi escit billo isput tatqu aliqu diams bipos itopu 800ta Isant oscul bifid mquec cumen berra etmii pyren nsomn anoct reern oncit quqar anofe ventm hipec oramo uetfu orets nitus sacer tusag teliu ipsev 825vi Eonei elaur plica oscri eseli sipse enitu ammih mensl quidi aptat rinar uacae ierqu vagas ubesc rpore ibere perqu umbra perqu antra erorp netra 850at mihif napat ntint riora intui urque nimus otoqu cagat rolym oecfu iunto ulosa tarac ecame suidt mande onatd stent spiri usore idpar thaec abies 875sa Imsep pretu tempu revol bileg rokam revoc tephe rosve etepe tenov sindu turqu brevt elliu repar tiuve tamia queso utage udulc vires humus fallo 900eu Anetn bisre freun carmi avire ingen umque miher muner veris adest duner veris adest iteru quevi escit billo isput tatqu aliqu diams bipos itopu 925ta Isant oscul bifid mquec cumen berra etmii pyren nsomn anoct reern oncit quqar anofe ventm hipec oramo

Tmande onatd stent spiri usore idpar thaec abies 1000a Imsep pretu tempu revol bileg rokam revoc tephe rosve etepe tenov sindu turqu brevt elliu repar tiuve tamia queso utage udulc vires humus fallo 1025u Anetn bisre freun carmi avire ingen umque miher muner veris adest duner veris adest iteru quevi escit billo isput tatqu aliqu diams bipos itopu 1050a Isant oscul bifid mquec cumen berra etmii pyren nsomn anoct reern oncit quqar anofe ventm hipec oramo uetfu orets nitus sacer tusag teliu ipsev 1075i Eonei elaur plica oscri eseli sipse enitu ammih mensl quidi aptat rinar uacae ierqu vagas ubesc rpore ibere perqu umbra perqu antra erorp netra 1100a Isant oscul bifid mquec cumen berra.

**Figure 9** An exaggerated drop cap draws attention.

Reverse Type

**Figure 10** Reverse type.

### B. Vary Size and Use Special Effects

Normal size may be less interesting to the reader. Altered size and special effects draw attention to the visuals.

### C. Reverse Type

Reverse type is usually white letters on a black, dark, or color background. This type style is very useful for headlines or short sentences (Fig. 10).

### D. Shades of Gray

Shades of gray can contribute a great deal to the success of a visual. The purpose of the shadow is to lend a three-dimensional effect to the flat page and to draw the reader's attention to that area.

### E. Contrast

A highlight of color on the printed page draws the eye immediately. When you are limited to black and white, you must be resourceful in gaining and retaining your reader's attention. Contrast is one of the most effective ways of doing so (Fig. 11).

### F. Use Bleed Techniques

"Bleeds" refer to extra areas of ink coverage that extend beyond the trim of the page. To bleed an element means to imprint it to the edge of the paper. You can bleed rules, borders, large letter forms, photos, large solid areas, and other elements. A bleed gives the page a feeling of expansiveness; the page seems larger than it actually is, unbounded by margins.

## XVI. TYPOGRAPHY

Type is nothing more than letters, and everybody has worked with letters most of his or her life. First we see typography in books and perhaps packaging. Then advertising, brochures, parts lists, and manuals became part of our typographic experience. Today, typographic communication in the form of newsletters, annuals, reports, and proposals has been added to our typographic exposure.

As an educator, you probably have a sense for what looks good in type, what is easiest to read, and what is accomplished by the most effective graphic communication. You know when a B looks like a B and when it does not. You know that if a typewriter malfunctions and sets letters so close to each other that the letters begin to overlap, the end result will be difficult to read. Consciously or subconsciously, you have gained knowledge regarding typographic communication. Of course, being familiar with typographic communication from a reader's viewpoint does not make one a typographer. It is, however, a very important step in the right direction.

### A. Type Style

Type style is the range of shapes and thicknesses within a typeface family. Type style makes it possible to

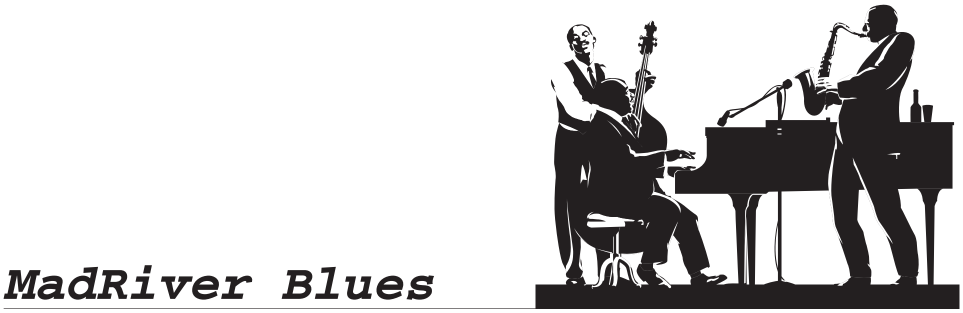


**Figure 11** The high contrast players against a white background and the bold text create a good example of the use of contrast.

create legible text in a quick and consistent manner and adds emphasis to the text as well. Letters may be changed and arranged to create an interesting and highly effective document. Using a variety of type styles often adds to the effectiveness of a publication. Words can be bold, italic, shadow, outline, underlined, etc. Type style can be modified to add contrast or emphasis to a publication.

## B. Type Weights

Weight refers to the thickness of the lines that make up letters and varies from light to heavy or even black (Fig. 12). Although a single typeface may exist in a variety of weights, no typeface currently offers every weight, and weights are not consistent from one typeface to another.

## C. Typeface

Since the invention of movable type nearly 500 years ago until recently, about 20 different typefaces were available. The advent of computers and DTP added more than thousands of different typefaces to the list, and it is still growing. A typeface is classified as serif or sans serif.

The serif, or cross-line, added at the beginning and end of a stroke probably dates from early Rome (Fig. 13). The serif was either a way to get a clean cut at the end of a chiseled stroke or an imitation of brush-written letter forms. The serif first appeared in ancient Roman monuments, where the letters were chiseled in stone. Later on, serifs appeared in the handwritten manuscripts prepared with quill pens by medieval scribes. Serif, or Roman, typefaces are useful in text because the serifs help distinguish individual letters and provide continuity for the reader's eye.

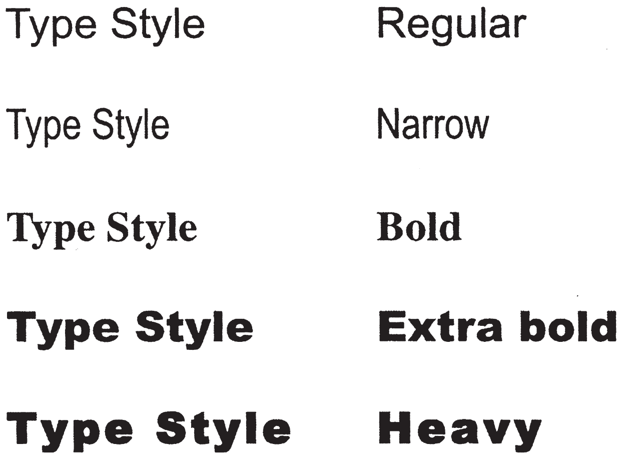


**Figure 12** Type weights.

**Figure 13** Examples of serif typefaces.

In French sans means without. Sans serif typefaces have no serifs (Fig. 14). In sans serif typefaces the strokes of the letters are usually of nearly even thickness, which tends to give them a very austere, mechanical appearance. This even, mechanical look works against sans serif faces when they are used in text applications. Their uniform strokes tend to melt together before the human eye.

Sans serif typefaces are most useful in large point sizes—in newspaper headlines, book titles, advertisements, and chapter headings—or in small bursts that are designed to contrast with surrounding serif type. Sans serif type is typically used for signs because of the simplicity of its letter forms and because it does not seem to taper at a distance. These same characteristics make sans serif type useful for highlighting amid serif type, especially in listing, directory, and catalog formats.

## D. Leading

Leading, or line spacing, is used to improve the appearance and readability of the publication. Leading (pronounced ledding) originally referred to the practice of placing small strips of lead between lines of type to make a page more readable. Although the job is no longer done with actual lead, the purpose and process are basically the same. Leading for lines of type specifies the distance from the baseline of one line to the baseline of the next line. It is usually at least the height of the font to prevent the bottoms of the letters on the top line from printing over the tops of the letters on the bottom line.

For smaller sizes of type, line spacing is often set one or two points (approximately 20%) greater than the type size to increase legibility. For example, if the

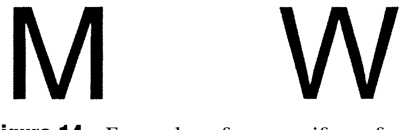


**Figure 14** Examples of sans serif typefaces.

type size is 12 points, the recommended leading is 14 points and usually is shown as 12/14 (12-point type with 14-point leading).

Leading that is too spacious makes the reader's eyes wander at the end of one line trying to find the beginning of the next line. Closed leading can be used as a design tool. Sometimes you might want to tighten leading so that descenders (the portion of a letter that extends below the baseline, as in the letter y) from one line of type touch the ascenders (the portion of a lowercase letter that rises above the main body, as in the letter b) from the line below.

## E. Word Spacing (Trackings)

Word spacing refers to the amount of space between words in a line of type. When words are closer together, more words can be included on each line. Word spacing increases or decreases the density of type. In certain situations, that also can reduce the number of hyphenated words. If you reduce word spacing too much, however, the text becomes difficult to read.

## F. Paragraph Spacing

Adding space between paragraphs enhances readability and makes each paragraph more like a self-contained unit. Paragraph spacing can be adjusted in many software packages.

## G. Text Alignment

Text alignment determines how lines of text are placed between the right and left margins. Lines can be flush left (aligned on the left and ragged on the right), flush right (aligned on the right and ragged on the left), justified (aligned on both the left and the right), or centered.

Often text is aligned flush left. This style creates an informal, contemporary, and open publication where each word is separated by equal space. Hyphenation is kept to a minimum. Western readers normally read from left to right; therefore, flush right/ragged left should be used cautiously. When flush right/ragged left is used, readers tend to slow their reading speed and spend more time identifying words by putting together individual letters.

Justified type, or flush left/flush right, because the text is forced together to fit in a line, is considered more difficult to read. Also, justified text contains more hyphenated words. Centered text is useful for short headlines. If a centered headline is more than three or four lines, however, readers have to search for the beginning of each line (Fig. 15).

Text does not always have to be presented in either justified or ragged right blocks. Often, it is desirable to "wrap" copy around the contours of a graphic.

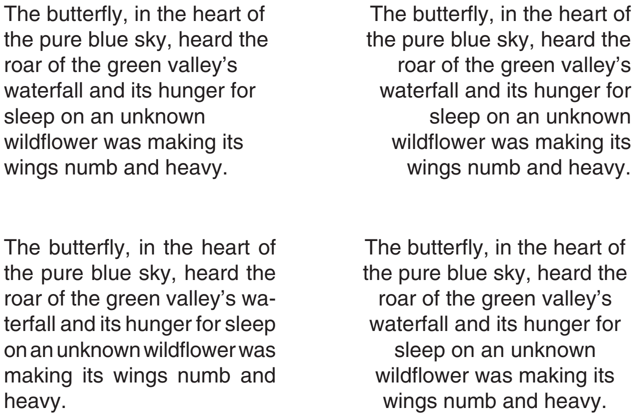
## H. Indention

An indent is the amount of space a given line or paragraph is inset from the normal margin of a paragraph or from the column guides. Software packages offer automatic and manual indent options. Indention enhances readability—especially when there is a large number of consecutive paragraphs on a page. The first line of a paragraph can have an indent or outdent. First-line indent is the most common. In this style, the first line of each paragraph is indented from the left margin (Fig. 16).

Hanging indent, or outdenting the first line, is the opposite of the first-line indent. In this style, the first line in a paragraph is at the left margin and the rest of the lines in the paragraph are indented from the left margin. Hanging indents are useful in bibliographical references and directory listings—the telephone book and a yearbook are typical examples (Fig. 17).

## I. Tabs

Tabs are used to align text at a particular location on the page, allowing great flexibility in formatting text. Most DTP packages space words and letters proportionally. Therefore, it is recommended to use tabs



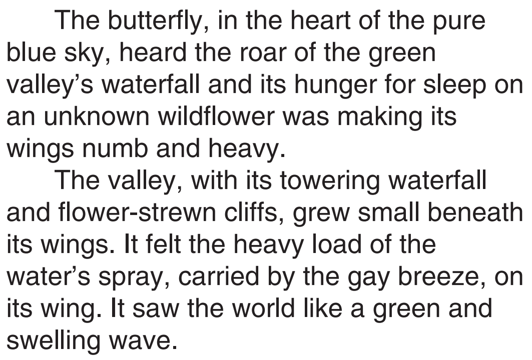
The butterfly, in the heart of
the pure blue sky, heard the
roar of the green valley's
waterfall and its hunger for
sleep on an unknown
wildflower was making its
wings numb and heavy.

The butterfly, in the heart of
the pure blue sky, heard the
roar of the green valley's
waterfall and its hunger for
sleep on an unknown
wildflower was making its
wings numb and heavy.

The butterfly, in the heart of
the pure blue sky, heard the
roar of the green valley's waterfall and its hunger for sleep
on an unknown wildflower was
making its wings numb and
heavy.

The butterfly, in the heart of
the pure blue sky, heard the
roar of the green valley's
waterfall and its hunger for
sleep on an unknown
wildflower was making its
wings numb and heavy.

**Figure 15** Flush left, flush right, justified, and centered text.



The butterfly, in the heart of the pure blue sky, heard the roar of the green valley's waterfall and its hunger for sleep on an unknown wildflower was making its wings numb and heavy.

The valley, with its towering waterfall and flower-strewn cliffs, grew small beneath its wings. It felt the heavy load of the water's spray, carried by the gay breeze, on its wing. It saw the world like a green and swelling wave.

**Figure 16** A first-line indent.

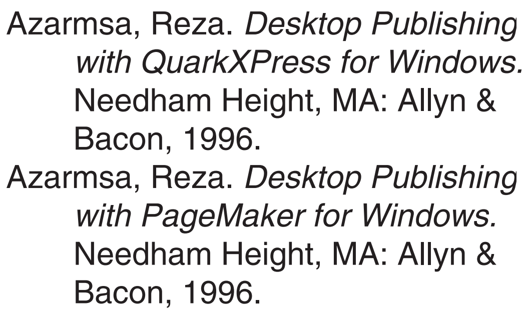
rather than spaces to align text; otherwise, the printed text will be uneven. Tabs are especially useful in creating tables and columns.

## J. Line Length

The length of a line of text, also known as the column width, can affect readability and the overall look of a publication. The line length can be as narrow as one letter or as wide as the boundaries of the paper. If a line of text is too long, a reader may become weary. If a line is too short, the text may give a choppy appearance and may be difficult to read. A good rule of thumb is to relate line length to type size: the bigger the type size, the longer the line length can be. An appropriate line length enhances the appearance of the publication and increases its readability. A good-sized line, depending upon the type of document, should be from 8 to 15 words long.

## K. Hyphenation

Hyphenation is the placing of a hyphen into a word so the word can be broken between two lines of text.



Azarmsa, Reza. *Desktop Publishing with QuarkXPress for Windows.* Needham Height, MA: Allyn & Bacon, 1996.

Azarmsa, Reza. *Desktop Publishing with PageMaker for Windows.* Needham Height, MA: Allyn & Bacon, 1996.

**Figure 17** Outdents.

Hyphenation can improve the appearance of your publication by helping you avoid particularly unequal line lengths in flush left/ragged right text and unsightly word spacing in justified text. Hyphenation is an area of trade-offs. A break in the middle of a word slows down reading. Desktop publishing programs offer different hyphenation options. You can have the program hyphenate the entire publication automatically or you can turn off the automatic hyphenation and hyphenate your document manually.

## L. Drop Capital

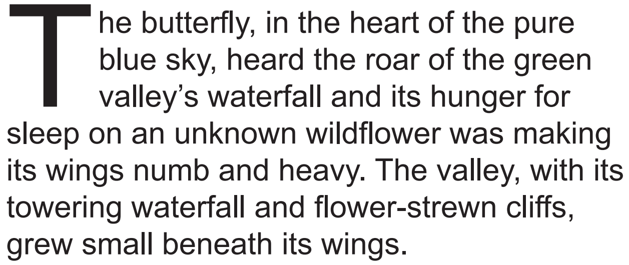
An enlarged first character at the beginning of a paragraph that extends (drops) into the following lines of text is known as a drop capital. A drop capital breaks up large blocks of text. Typically, drop capitals are used to begin a chapter or section in a book or magazine. Drop capitals provide an important visual transition between the headline and body copy. Drop capitals can have the same or different character attributes, such as font, size, color, and style, as the rest of the text in the paragraph (Fig. 18).

## M. Stick-Up Capital

An enlarged initial letter that extends above the body text is called a stick-up capital. A stick-up capital is used as a graphic element to draw attention to the beginning of a story or chapter (Fig. 19).

## N. Type Size

The type size of a letter (also called height or point size) is measured by its vertical height, in points. One point is equal to one-twelfth of a pica, which in turn is almost exactly one-sixth of an inch. Therefore, there are 72 points per inch. Point is normally abbreviated "pt." For example, 72-pt. type would measure 72 points from the top of its ascender to the bottom of its descender. This



The butterfly, in the heart of the pure blue sky, heard the roar of the green valley's waterfall and its hunger for sleep on an unknown wildflower was making its wings numb and heavy. The valley, with its towering waterfall and flower-strewn cliffs, grew small beneath its wings.

**Figure 18** An example of drop capital.

The butterfly, in the heart of the pure blue sky, heard the roar of the green valley’s waterfall and its hunger for sleep on an unknown wildflower was making its wings numb and heavy. The valley, with its towering waterfall and flower-strewn cliffs, grew small beneath its wings.

**Figure 19** An example of stick-up capital.

means that x heights, as well as ascenders and descenders, will vary in size for different fonts even though their point sizes are the same. Except for f, j, l, and t, the body of lowercase letters rises to the x height (Fig. 20).

## O. Letterspacing

Letterspacing refers to the amount of space between each letter of a line or block of type. Each letter consists not only of the letter itself but also a tiny amount of space before and after the letter called the left- and right-side bearing.

In general, it is desirable to keep the letterspacing range to a minimum, as wide spacing fluctuations give an uneven look to the page. Also, larger typefaces call for tighter letterspacing. Some DTP packages allow you to control the letterspacing.

## P. Kerning

Kerning means the adjustment of space between pairs of characters to create visually even text that is easy to read. Most of the popular DTP programs are capable of adding or subtracting space between characters in minuscule increments. Certain pairs of letters appear to be separated by too much space. According to the Adobe type catalog *Font & Function,* the table below shows the pairs that have a hard time “getting along.”

AO
AT
AV
AW
AY
Ac
Ad
Ae
Ao
Au
Av
Aw
FA
Ka
Ke
Ko
LY
Ly
OV
OW
OX
PA
TA
Ta
Te
To
Tr
Tu
Tv
Tw
Ty
VA
Va
Ve
Vo
Vu
Vy
WA
Wa
We
Wo
Wr
Wu
Wy
YA
Ya
Ye
Yo
Yu
av
aw
ay
ev
ew
ex
ey
ov
ow
ox
oy
rw
ry
va
vo
wa
we
wo
xc
xe
xo
ya
yc
yr
yo

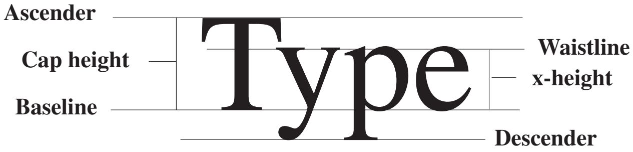


**Figure 20** Type size anatomy.

Too much word spacing breaks the line into separate elements, inhibiting reading. It also creates gaps within columns. When word spacing is greater than line spacing, reading is difficult because the eye tends to move from top to bottom instead of from left to right. Spacing between words also affects your type’s appearance. Kern word spaces in harmony with letterspaces to create evenness and balance within a line or block of type.

Once you know the basic typographic rules and have had a little firsthand experience with preparing simple typographic communication, it becomes relatively easy to produce a variety of documents.

A surprising amount of beautiful and efficient communication can be created with just your printer’s core fonts. More than 80% of all printed communication is set with just these two typefaces, and this includes lots of very creative and sophisticated graphic design work.

# XVII. WAYS TO IMPROVE TEXT

The primary function of words is to communicate instantly and effectively. A difference in type style not only helps to attract the reader’s eye but also helps to organize information in terms of importance. The following points may enhance the effectiveness of text.

## A. Avoid Too Many Typefaces

Desktop publishing manufacturers loaded their software with typefaces. In practice, this abundance can create an absolute disaster in terms of good typographic design, and overloads of typefaces should be avoided.

As a general rule, two typefaces (a headline face and a body copy face) are adequate for most page designs. A variety of textures and styles can be conveyed by your choice of leading (the space between lines of type), weight (bold, light, or thin), size, and alteration of spacing between letters.

## B. Avoid Fancy Typefaces

Although you want distinctive type, do not choose a typeface that is too elaborate. Fancy typefaces can be very difficult to read.

### C. Avoid All Caps

Uppercase type has its uses, but use of only caps slows down readers. Some typefaces cannot be set in all caps.

### D. Avoid Large Blocks of Text in Italic or Bold

Try not to set a great deal of text in italic or bold, particularly if you are using small type. Also, italic's small size in all caps tends to fit badly and is thus more difficult to read.

Use italic type sparingly for emphasis or when irony or humor is intended. It can also imply an interjected conversational tone or a quotation. The use of italic can preclude quotation marks. Italic type is often used to set captions and the titles of books and other creative works.

### E. Avoid Excessive Underlining

Underlining is a means of emphasis. However, the availability of italic and bold types makes most underlining unnecessary. More than a few underlined words causes visual clutter and confusion. Also, it takes more time for readers to separate the words from the horizontal lines.

### F. Control Widows and Orphans

A widow is a line, a word, or a syllable of text at the end of a page or column that is separate from the paragraph that it finishes, which is at the end of the previous page or column. An orphan is a line, a word, or a syllable of text at the bottom of a page or column that is separate from the paragraph it begins, which starts the next page or column.

Widows and orphans can be controlled in several ways: edit the paragraph and omit unnecessary words; substitute a shorter word for a longer one and tighten the spacing of any loosely set line. (You can do this by reducing the tracking gradually.) Also, widows and orphans can be eliminated by rejuggling the page or the column lengths. Desktop publishing software packages offer widow and orphan control, and you can specify the number of lines of widows or orphans.

In short, DTP has changed the way people produce documents. It has enhanced the individual's ability to control all or most of the process of producing printed materials.

The temptation is to add many different graphic and typographic elements in one publication. Such a mixture, though, usually defeats the purpose of attracting and keeping the reader's attention. By keeping your design simple and using graphics and type selectively, you will produce the most effective and pleasing publication.

## SEE ALSO THE FOLLOWING ARTICLES

Copyright Laws • Electronic Mail • End-User Computing Tools • Hyper-Media Databases • Internet Homepages • Multimedia • Productivity • Speech Recognition • Spreadsheets • Word Processing

## BIBLIOGRAPHY

Azarmsa, R. (1998). *Desktop publishing for educators: Using Adobe PageMaker.* Boston: Allyn & Bacon.

Bauermeister, B. (1988). *A manual of comparative typography: The PANOSE system.* New York: Van Nostrand Reinhold.

Beaumont, M. (1987). *Type: Design, color, character & use.* Cincinnati, OH: North Light Books.

Brown, A. (1989). *In print: Text and type in the age of desktop publishing.* New York: Watson-Guptill.

Collier, D., and Floyd, K. (1989). *Layouts for desktop design.* Cincinnati, OH: North Light Books.

Durbin, H. C. (1995). *Desktop publishing systems.* Easton, PA: Durbin Associates.

Green, C. (1997). *Desk top publisher's idea.* New York: Random House.

Heinich, R. (1985). *Instructional media and the new technology of instruction,* 2d ed. New York: John Wiley & Sons.

Henry, J. H. (1996). *Do's & don'ts of desktop publishing design.* Ann Arbor, MI: Promotional Perspectives.

Maxymuk, J. (1996). *Using desktop publishing to create newsletters, library guides, & web pages: A how-to-do-it manual for librarians.* New York: Neal-Schuman Publishers.

Parker, R. C. (1990). *Looking good in print,* 2nd ed. Chapel Hill, NC: Ventana Press.

Seybold, J. W. (May 1987). The desktop-publishing phenomenon, *Byte,* 12:149–165.

Shushan, R., Wright, D., and Lewis, L. (1996). *Desktop Publishing by Design,* 4th ed. Redmond, WA: Microsoft Press.

Silver, G. A. (1997). *Layout, design, & typography: For the desktop publisher.* Encino, CA: Editorial Enterprises.

Tilden, S. (1987). *Harnessing desktop publishing: How to let the new technology help you do your job better.* Pennington, NJ: Scott Tilden.

Willams, J. B., and Murr, L. E. (Spring 1987). Desktop publishing: New right brain documents, *Library Hi Tech,* 7–13.

# Developing Nations

**Elia Chepaitis**
*Fairfield University*

## GLOSSARY

**developing economies** Formerly referred to as lesser-developed countries, whose nations are characterized by a gross domestic product (GDP) under 2000, and high rates of illiteracy, infant mortality, and an inadequate material infrastructure.

**digital divide** An intellectual construct used to describe two information environments, one for the information "haves" and the other for the information "have-nots."

**emerging economies** Economies in transition from communism to a market economy, generally used for nations in the former Soviet bloc.

**hard information infrastructure** The sum of those material factors, such as telephone lines or computer networks, upon which computer and communications systems rely.

**ICTs (information and communication technologies)** Information systems that are also used for communications.

**information ethics** Moral codes of behavior characterized by integrity and fairness in the creation, dissemination, and ethical use of information.

**information poverty** An endemic problem, characterized by a dearth of quality information.

**IS (information system)** A system that creates, processes, stores, and outputs information.

**soft information infrastructure** The political, economic, and sociocultural factors which affect information quality, management, and access.

## I. INTRODUCTION

Information systems (IS) in developing, emerging, and newly industrialized economies are vital for economic prosperity and stability, and their development is one of the greatest challenges of the early 21st century. Increasingly, as the computer evolves as an invaluable communication device, IS are referred to as information and communication systems (ICTs).

ICTs deeply affect the tempo and direction of social and political change in developing and transition economies, especially where wealth and infrastructures are inadequate and unevenly distributed. Moreover, effective ICTs are necessary for survival in a global economy, a competitive precondition for strategic relationships.

Before the 1990s, before the advent of the Internet and e-business, organizations in developing economies who lacked the capacity for electronic data interchange (EDI) often were bypassed by potential external partners in favor of connected enterprises. In the 2000s, at every level of economic development, connectivity and information management are more critical for prosperity and stability than ever.

Technology is not neutral: the danger that ICTs will be more pernicious than benign in some developing areas has been researched extensively. However, in general, technologically and empirically, greater opportunity exists in the early 21st century for those previously excluded from the Information Age since the advent of the global Internet and the introduction of wireless communications.

## II. CRITERIA: IMBALANCE AND UNDERDEVELOPMENT

Developing economies, once referred to as lesser-developed economies (LDCs), are characterized by a poor infrastructure, inferior growth rates, an imbalanced economy, and extremely low personal incomes. These economies lack necessary skills and resources to escape a heavy reliance on production from agriculture or mineral resources. This imbalance is a legacy of both external aggression and domination, but also of internal factors: poor natural endowments, inadequate human resources, political instability, and social inequality. Developing economies tend to be concentrated in Africa, Latin America, and the Middle East, and also in some states and clients of the former Soviet Union.

In the 1990s, a new classification, emerging economies, was coined to describe postcommunist countries that face an arduous transition to a market economy.

Russia, Belarus, and Ukraine share endemic developmental obstacles with traditional developing economies. Since 1980, numerous nations such as India, Brazil, Estonia, South Korea, and Turkey dramatically improved their GDP and became known as newly industrialized economies (NICs). However, these NICs also feature imbalanced economies and are globally competitive only in selective sectors. Clusters of poverty within NICs and emerging economies in the former Soviet Union resemble poverty in developing economies. A discussion of information systems in underdeveloped areas often applies to clusters in all three types of economies; technological dualism is common within all three models. Conversely, within developing economies, oases of advanced ICTs resemble more prosperous nations.

Indeed, the rural poor in India (a newly industrialized country), Honduras (a developing country), and Ukraine (an emerging economy) share numerous problems with underdeveloped areas worldwide who lag in the information revolution. Conversely, wired clusters of urban, relatively affluent and cosmopolitan populations flourish within all three economic models; these information-rich oases access global information and communications and resemble advanced economies. Current data do not reflect the maldistribution of GDP, telephone lines, and personal computers within economies. However, World Bank data illustrate the huge differences in information systems availability between advanced economies (the United States) on the one hand, and NICs, merging economies, and developing economies on the other hand (Table I).

The correlation among GDP per capita, the number of telephone lines, and personal computer ownership is evident.

## III. THE CRITICALITY OF INFORMATION AND COMMUNICATION TECHNOLOGY FOR DEVELOPMENT

At every economic level, but especially for developing economies, information and communications are critical, both for internal economic development and also for global competitiveness. Internally, information can supplement and expand scarce resources such as capital, know-how, labor, inventory, basic materials, and management skills. Also, unlike these resources, information is not depleted when it is utilized but increases in both quality and quantity. Moreover, information can supplement factors of production simultaneously, not one at a time, with significant synergies emerging. For example, if information resources are used to minimize inventory, less capital, labor, and warehouses are required; also, more current and complete knowledge of consumer preferences, supply, and distribution is acquired. In resource-poor environments, such optimization is critical not only for the expansion of wealth, but also for the formulation of successful business strategies in volatile and turbulent environments.

More dramatically, computers have emerged as essential communication devices in the past decade, compensating for egregious deficits in traditional telecommunications infrastructures. The maturation of wireless communications holds enormous promise for impoverished areas that have been unable to afford land-based telephone systems. Extended information and communication systems facilitate joint ventures with partners in advanced economies, improve customer relations internally, and enable developing economies to participate in global markets with flexibility and speed. They also can facilitate inexpensive village-to-village communication and expand scarce services in fields such as medicine, banking, insurance, and education.

**Table I** The Correlation among GDP/Capita, Telephone Lines, and Personal Computers: A Wide Range of IS Development from Tanzania to the United States[a]

| Country | GDP/capita | Telephone lines | Personal computers |
|---|---|---|---|
| Algeria | 1540 | 51.9 | 5.8 |
| Bolivia | 990 | 61.7 | 12.3 |
| Columbia | 2170 | 160.0 | 33.7 |
| Egypt | 1380 | 75.0 | 12.0 |
| Guatemala | 1680 | 55.0 | 9.9 |
| Indonesia | 580 | 29.0 | 9.1 |
| Jordan | 1620 | 61.8 | 13.9 |
| Morocco | 1190 | 52.6 | 10.8 |
| Nigeria | 250 | 3.9 | 6.4 |
| Pakistan | 510 | 18.4 | 4.3 |
| Romania | 1510 | 167.0 | 26.8 |
| Russian Federation | 1750 | 210.0 | 37.4 |
| South Africa | 3160 | 125.0 | 54.7 |
| Tanzania | 260 | 4.5 | 2.4 |
| Uzbekistan | 640 | 66.0 | N/A |
| United States | 31920 | 664.0 | 510.5 |

[a] World Bank Data Profiles of GDP, Telephone Lines, and Personal Computers in 1999. "Indicators Database," *World Bank Group,* July 2001.

However, the gap between wired and nonwired communities threatens to become increasingly significant as ICTs mature and improve. The movements toward increased bandwidth and toward the fusion of information technologies in advanced economies may deepen the digital divide, the gap between the information "haves" and "have-nots." Although advances such as wireless communications convey "democratic" and broad benefits, the rate of adoption is sporadic and sluggish in developing areas and is not keeping pace with changes in global markets.

Indeed, in 2002, although IS problems have changed, the gap in ICT development relative to advanced economies resembles the gap in computerization 20 years earlier (Table II).

## IV. CHANGING MARKET CONDITIONS: AN ACUTE PROBLEM FOR IS DEVELOPMENT

Developing countries face a dilemma: an adequate telecommunications infrastructure is becoming

**Table II** Changing Problems in IS in Developing Economies

| 1980 | 2002 |
|---|---|
| High cost of hardware | Political and cultural resistance |
| High level of education | Lack of external partnerships |
| Lack of effective software | Information quality |
| Development and use | Internet access/content regulations |
| Poor electrical/telecommunications/ education infrastructure | Financial market imperfections: lack of credit; fraud; mistrust |
| Lack of EDI capability commercial law | Inadequate property |
| Ethics and corruption | Language<br>Urban–rural split |

increasingly necessary to attract global business partners (including customers); yet capital and partners are necessary to construct that infrastructure. No longer can developing economies rely on cheap labor resources to attract direct and indirect investment, or to ensure price competitive exports. Labor costs as a percentage of the costs of production have been steadily declining as a result of a confluence of factors, and typically represent 5 to 15% of the costs of production, depending on the product. (High tech products typically are tilted toward 5%, the lower end of the range.) Labor markets are shifting dramatically and nations that depend on low wage rates for competitive advantage are in peril, in large part as a legacy of the total quality movement (TQM). TQM emphasized superior design, continuous product improvement, capital-intensive equipment and processes, and less direct labor with less inspection. Enterprises are shifting strategies away from island hopping across low-wage markets with shoddy workmanship, unreliable infrastructures, and low quality cultures toward high skilled labor and quality cultures.

To utilize ICTs for economic competitiveness, developing economies need capital, partners, and information resources as soon as possible. The movement toward superior production technology and processes is inexorable; ICTs are vital for quality improvements, integrated processes, a skilled workforce, and cost containment. Producers in developing economies find that affluent clusters in the domestic market are neither loyal, local, nor captive. The middle to upper class urban householders are not only on the privileged side of the digital divide, but also positioned to take advantage of e-commerce, improved and affordable transportation, and global services.

## V. CHALLENGES IN SYSTEMS DESIGN: INFORMATION INFRASTRUCTURES

Systems designers must consider that information systems in developing and emerging economies are affected by many of the same macro- and microeconomic forces which shape the economy as a whole: capital shortages, a lack of skilled labor, inefficient distribution channels, poor resource endowments, and remote locations or inhospitable terrain or climate. Information technology in many of these areas is more important for communications than for computation, but salient barriers to cost-effective information ICTs in developing economies are confined to the telecommunications infrastructure. In addition to the technology infrastructure, the local information infrastructure affects ICT effectiveness. The information infrastructure is the context within which information and communications are managed. Information systems reside and develop in discrete environments which, although affected by global markets, feature distinctive, nonglobal components: spatial, material, cultural, institutional, organizational, political, and economic elements. These components must be considered integral and organic to any discussion of international information infrastructures.

In both developed and emerging economies, ICT effectiveness is affected by two information infrastructures: hard and soft. The hard information infrastructure includes factors such as connectivity, electrical supply, available bandwidth, wireless resources, and the availability of IT vendors, maintenance, and supplies (Table III).

The collection and manipulation of data, as well as the dissemination of information and knowledge, also depend on a soft information infrastructure: the legal, political, economic, and social information environment that impacts ICTs (Table IV). This "soft" information infrastructure is largely composed of nontechnical elements: spatial, cultural, organizational, political, and economic variables which are highly germane to the tempo and direction of economic development.

For example, information quality may be variable in emerging economies. Information quality is affected by cultural, economic, and political variables such as social trust, informal relationships, and commercial law, respectively (Tables V–VII). These variables may be intrinsic to a gray economy, may compensate for unreliable distribution or credit mechanisms, or may simply be traditional but atavistic.

Social characteristics that affect an information environment include habits of trust and integrity, language and alphabets, occupational mobility, educational patterns, ethics and equity, and ways of knowing (Table VII). Across information cultures, for examples, trust has a seminal impact not only on information quality and access, but also on a range of factors—from the viability of e-business and web page

**Table III** **The Hard Information Infrastructure**

| |
|---|
| Communications |
| Wireless |
| Ground wire |
| Platforms: hardware and software |
| Electricity and other utilities |
| Transportation |

**Table IV** The Soft Infrastructure

Political and legal variables

Economic and financial variables

Sociocultural variables

design, to the degree of covertness in security. Moreover, trust has an egregious impact on information ethics in cultures with well-established habits of secrecy and deception, and thus can cripple integrated and extended ICTs. Within each list of soft factors, some potential impacts are presented in Tables V–VII. These environmental elements may not only be ubiquitous, but also be dynamic and critical in ICT development: they impact the accepted division of labor, wealth creation, and market development significantly. Also, political and legal factors are integral to the information infrastructure: authority and legitimacy, the definition of rights and obligations, and citizen/subject compliance affect information quality, integrity, and access.

In the 21st century, information system developers face a dynamic set of challenges: to understand, exploit, promote, and work around elements of hard and soft information infrastructures to optimize economic development. Since the information infrastructure is not only a skeletal series of linkages but also an organic membrane in which communications pass and evolve, the internal information infrastructure itself changes and resists change. Especially in emerging economies, the soft and organic information infrastructure is the underbelly of successful ICT development. One of the most significant issues in ICT development in developing economies is the regulation of the Internet.

## VI. THE REGULATORY ENVIRONMENT: POLITICAL AND CULTURAL CONTROL

Government priorities shifted in past decades as information systems became affordable and powerful, and as local computer retailers and maintenance industries proliferated. Also, popular culture and global

**Table V** The Soft Information Infrastructure Social and Cultural Variables: Some Possible Impacts *(in italics)*

Information poverty: *data integrity and completeness*

Soft information: *reticence, oral traditions, fraud, lack of knowledge worker skills*

Existing elites: *proprietary or hoarded data, monopolies*

Indirect planning: *tempo of decision making, external variables weighted*

Traditional information content and format

Bargaining discourse: *areas of extreme limited disclosure*

Social capital, including trust language: *possible mistrust of online data*

Ways of knowing

Storytelling: *information format neural nets, fuzzy logic*

Message style: *interface and output format*

Nonverbal signals: *agreements or malleable positions not textual*

Timing of exchanges

Literacy and knowledge access

Occupational mobility

Language and alphabets/characters: *dialect*

Logic

Views of time

Team-building patterns: *incremental problem-solving*

Value on consensus: *layers of duplicated and verified effort*

Ethics: *piracy desirable for infant economy*

Equity: *every worker reassigned if displaced*

Codes of conduct: *tolerance of cheating*

**Table VI** **The Soft Information Infrastructure Economic and Financial Variables: Some Possible Impacts *(in italics)***

Financial institutions and practices: *data integrity and availability*

Accounting standards and practices: *choice of models; information availability*

Logistics and communication: *procesing speed*

Fiscal and monetary stability: *range of real and nominal variability*

Surplus and division of labor: *disguised underemployment*

Labor productivity: *output per unit variable*

GDP and distribution of wealth: *marketing projections skewed*

Wealth creation mechanisms: *gray economy affects GDP reliability*

Labor organization: *hours worked, benefits, holidays*

communications popularized the collateral benefits of ICTs, and more democratic governments succumbed to pressure for relatively unrestricted Internet access.

However, throughout the world, the Internet poses a dilemma: the value of connectivity must be balanced against a range of problems such as fraud and pornography, problems which encourage legal and political controls. Regulations proliferate in some developing economies, where often the government is the sole Internet service provider (ISP), and prohibitions against online political discussions are brutally enforced. Occasionally, the regulatory environment is onerous, and compliance with cultural control and political censorship can curtail ICTs aid to market development (Tables VIII and IX).

In traditional international trade and investment, developing economies often adopted regulations for a variety of purposes: quotas and other on-tariff barriers, luxury taxes on pricey imports, tariffs, protection for infant industry, special labor benefits, tolerance for intellectual property thefts, and special taxation of foreign enterprises.

Restrictions on online commerce often protect authoritarian governments, fundamentalism, or sociocultural elites, rather than infant industry, labor, or distressed sectors of the economy. Some governments are hostile to the Internet, and political controls may restrict ICT in critical developmental areas (Table IX). The current regulatory environment often affects partnerships which depend on e-commerce, multiple ISPs, and intellectual property. Restrictive policies reflect the priorities, tensions, traditions, and tempo of economic development.

Government priorities have shifted in the past decades as information systems became affordable and powerful, and as local computer retailers and maintenance industries proliferated. Also, popular culture and global communications popularized the collateral benefits of ICTs, and most democratic gov-

**Table VII** **The Soft Information Infrastructure Political and Legal Variables: Some Possible Impacts *(in italics)***

Authority and legitimacy: *varying degrees of enforcement/interpretation*

Property and other commercial law: *ability to raise and protect capital*

Well-defined rights and obligations

Regulation of the Internet: *persistent digital divide*

Successful appeal mechanisms

Functional bureaucracy

Citizen or subject loyalty/compliance: *reliability of tax, GDPO data, nonconfiscatory taxation, tariffs, and nontariff policy; confiscation, expropriation; nationalization*

Conflict resolution mechanisms: *degree of disclosure, methods of closure*

Government aid: *national priorities; fields necessary for demonstration of need*

Supervision of electronic communities: *privacy legislation; security of intellectual property*

**Table VIII** Internet Regulations: A Contrast between Common Legal Issues in All Economies and Legal, Political, and Cultural Regulation Areas in Developing Economics

| Internet regulations | |
|---|---|
| **Common global legal issues** | **Additional issues in developing economics** |
| Fraud | Political opinions and criticism |
| Privacy | Regulated data collection |
| Theft of intellectual property | Regulated computer ownership |
| Pornography | Secularism in fundamentalist societies |
| Hacking | Government ownership of ISPs |
| Cyberterrorism | External news reports |
| Sabotage and electronic crime | Constraints on intellectual property law |
| Prostitution rings | Western culture |
| Gambling offshore | Hardware, software, network costs |
| Taxation | Maintenance |
| Anti-trust activity | Information access |

**Table IX** Political Resistance to Cyberspace[a]

| | |
|---|---|
| Belarus | Single, government-owned ISP (Belpak) |
| Burma | State monopoly on Internet access; computer ownership must be reported to the government |
| Tajikistan | Single, government-owned ISP (Telecom) |
| Turkmenistan | More restricted than Tajikistan |
| Usbekistan and Azerbaijan | Private ISPs controlled by telecommunications ministry |
| Kazakhstan and Kyrgyzstan | Private ISPs but exorbitant government fees for usage and connection |
| China | Users monitored and government-registered |
| Cuba | Government-controlled Internet |
| Iran | Censorship and blocked sites; content restrictions apply to discussions on sexuality, religion, United States, Israel, and selected medical and scientific data (anatomy) |
| Iraq | No direct access to the Internet; official servers located in Jordan; few citizens own computers |
| Libya | Internet access impossible |
| North Korea | Internet access impossible; government servers located in Japan |
| Saudi Arabia | Science and Technology Center screens information offensive to Islam values |
| Sierra Leone | Opposition and online press persecuted |
| Sudan | Single, government-owned ISP (Sudanet) |
| Syria | Private Internet access illegal |
| Tunisia | Two government-owned ISPs |
| Turkey | February 2001 content restrictions |
| Vietnam | Two government-owned ISPs |

[a] From Reporters without Borders, "The Twenty Enemies of the Internet," August 9, 1999, press release: "Molly Moore, Turkey Cracks Down on Youths Using Internet," *Hartford Courant,* February 4, 2001, A10.

ernments succumbed to pressure for unrestricted Internet access.

## VII. THE DIGITAL DIVIDE: INTERNATIONAL PROJECTS AND ACTION PLANS

Since the 1970s, before computers evolved into potent communication devices, international resolutions, conferences, and manifestos called for a new international information order for the poorest nations. As the agencies of the United Nations turned their attention from peacekeeping toward development, information was perceived as a key to affluence and independence. Resentment toward multinationals' perceived monopoly on information resources fueled a movement by nonaligned "Third World" nations to embrace development programs subscribed to by more than 90 members of the United Nations by the 1970s.

A 1976 UNESCO commission on communications recommended balanced and equal access to information, and specifically noted that "diverse solutions to information and communication problems are required because social, political, cultural, and economic problems differ from one country to another and within a country, from one group to another." The attention of international agencies shifted from television and telephones to computers and the Internet in the following 20 years, and the issue of government-controlled media emerged as salient and divisive.

Regional and national projects have been influenced by the European Union's resolutions on privacy, by the World Trade Organization's studies of mandatory data control standards, and by transnational efforts to combat undesirable Internet content and costs. More than ever, the fusion of information technologies and the emergence of wireless communications have placed ICTs in the forefront of three developmental problems: the need for capital, the need for technical assistance, and the growing digital divide.

## VIII. ICTS: TOOLS FOR PROJECT COORDINATION AND RESEARCH DISSEMINATION

Numerous information systems journals, international aid organizations, and national commissions are currently dedicated to the democratization of computing and communication systems. An influential online journal, *Eldis,* is sponsored by Sussex's Institute of Developmental Studies. *Eldis* published a World Economic Report in 2001 which explores the impact of ICTs on work in developing areas (Table X). This report focuses on the disparity between positive impacts of e-business in advanced economies, and the devastating affects on labor in developing economies. Online research and reports on economic development initiatives by scores of aid organizations, such as the World Bank, USAID, and OECD, as well as multiagency reviews are available.

On a regional basis, within Africa alone, dozens of projects are coordinated and expanded through ICTs, such as Project Africa (centered in Senegal), the Leland Project (USAID), the Pan African Development Information System (PADIS) Project, and Projet Afrinet (French). Within Latin America, almost every country hosts a Web site which publicizes the goals and strategies of ICT adoption. Global audiences can reach Argentina, Bolivia, or Chile through UNESCO to read

**Table X** **World Economic Report 2001: An Excerpt of Issues for Developing Economies**[a]

| |
|---|
| Life at work in the information economy |
| Irreversibility and speed |
| A widening digital divide |
| How will markets be affected |
| How will work organizations be affected |
| Education matters most of all |
| How will the quality of life and work be affected |
| Social factors and social choices for addressing negative consequences |

[a] *Eldis,* World Employment Report 2001, International Labor Organization (ILO), 2001.

the latest statutes or development initiatives for ICTs. UNESCO also maintains an Internet Public Policy Network, G8 Global Information society Pilot Projects, OECD infrastructure guidelines, and a current ASEAN ICT Framework Agreements. The growth of virtual research and development centers for ICT development holds significant promise for swifter and more comprehensive responses to developmental challenges.

To move into the information economy, developing countries must set priorities and learn from the mistakes of the early leaders. International aid organizations and forward-looking political leaders typically support the following goals: universal information access, decision support systems, access to international information highways, vibrant private sector leadership, and global access to data within developing economies. However, the lack of capital, the question of intellectual property, the liberalization of national communications and public broadcasting services, and the development of human resources are salient problems. For example, not only have human resources been depleted by brain drain out of the country before the Internet era, but skilled workers can now work for foreign rather than national enterprises online, without emigrating.

The most persistent digital divides may endure within developing economies, if current patterns of economic development mirror uneven patterns of growth and investment, and if political patronage and largesse is a guide. Just as the gap between developing and advanced economies has widened in the past decade, so the gap between wired and nonwired populations will probably increase within the poorest nations, similar to the patterns we see in NICs like India today.

Patterns of economic development are uneven in developing and emerging economies, and, barring seismic political and cultural changes, the digital divide will deepen along these lines, destabilizing the economy and constraining balanced long-term growth. The seismic fault lines of the digital divide will probably produce egregious inequity where there are rural, mountainous, or interior regions with poor or nonexistent utilities; where dialects or minority languages are spoken; where minority religions, castes, tribes, or female populations are predominant; where investments in education are lacking or minimal; and where political elites have no vested interest in popular support (Table XI). The digital divide is pernicious, not only to producing an informed population and to linking remote populations to global markets, but to spreading the services of pharmacists, educators, physicians, agronomists, managers, and technicians.

In advanced economies, the information highway has opened opportunities to small nations, small businesses, innovative shops, and minority chat groups. In developing areas, the potential for the democratic expansion of e-commerce depends on substantial external aid, extensive economic integration, and responsible political leadership.

**Table XI** The Digital Divide: Developing Economies

| Resources needed | Most critical areas |
|---|---|
| Capacity building | Fault lines across and between sectors |
| Technical assistance | Deprivation along telephone systems |
| Capital | Topography matters |
| Political leadership | Political nonelites excluded<br>Agricultural–rural divisions<br>Littoral–inland splits<br>Industry-specific contrasts<br>Education levels<br>Language: the problem of dialect |

## IX. ETHICS: BEYOND PIRACY AND PRIVACY

Corruption, inequity, and immoral behavior in economic development migrate to information systems. In the 1990s, substantial intellectual and cultural resources were used to identify and define ethical problems for in international business, to achieve a rough consensus on priorities, and to design solutions. Unfortunately, these designers were often working with arm-chair philosophers and social engineers who did not focus on unethical behaviors within developing areas. However, the will to address moral behavior was established both globally and locally.

In advanced economies, information ethics emerged as a logical and vital extension of computer ethics. Before this extension, computer ethics focused on software piracy and other crimes against property such as the theft of computer time for personal use. Social and economic issues surfaced, such as egregious failure to develop ergonomic and user-friendly information systems. In the 2000s, riveting problems demand attention: security, unemployment, and equitable data access for a wide range of shareholders. With the expansion of e-commerce and the need for economic justice, not only ethicists and IS professionals, but also international organizations and global consumer advocates seek effective professional and moral codes of behavior.

In developing economies, the thrust and the centrality of information ethics differ from those societies with mature markets and global information systems.

Without the ethical development and use of information, information poverty cannot be eradicated. In addition, the stakes are critical: first, establish stability, popular trust, and a consensus on the new economic order; second, focus on the expansion of wealth, opportunity, and equity. From Honduras to Russia, computer and information ethics are preconditions for both stable market mechanisms and the maturation of information systems. The lack of information integrity and access contributes to low or stagnant growth and cripples orderly transitions to an information economy. Immoral ICT behaviors will become immoral behaviors via ICTs without intervention, crippling economic development from Mexico to China. Mismanaged, hoarded, and distorted information results in missed opportunities and valuable partnerships, inefficient distribution and supply channels, fraud, unproductive hidden assets abroad, egregious personal aggrandizement, tax avoidance, and unfair advantage—luxuries which developing economies cannot afford.

The search for an ethical consensus draws upon experience, development theory, and multicultural resources, and represents a marked shift away from utopianism and toward action-based pragmatism and individual accountability. The stakes are enormous; the evolution of information ethics affects not only the viability of ICTs, but the digital divide within the most needy societies. The success of open, integrated, and extended systems depends upon the availability of information technology as well as equitable and honest information-handling behaviors and other moral considerations. Information systems cannot succeed in environments with endemic corruption, information-hoarding elites, illegality, and other moral considerations. The significance of historical experience, consequence-oriented traditions, and culture are germane but must not be overstated. Although ethical problems in information management arise from local economic, political, legal, and cultural factors, research on information ethics reveals a craving for moral consensus at every stage of economic development. At all levels of society, a debate on ethics is taking place, conflicting ethical norms have surfaced, and strategies to deal with corruption sit at the top of political and economic agendas. A critical mass of evidence on unethical behavior has accumulated, and the short- and longterm repercussions are self-evident and instructive.

Action-based ethics feature three major reference points: a reevaluation of existing practices and values in developing economies, a critical analysis of the relationships among ethics and a market economy both internally and abroad, and a search across multiple cultures and within layers of cultures for the identification, definition, and solution to ethical problems.

Although intercultural transfers of moral norms, or the "missionary position," cannot be transferred from advanced to developing economies, developing economies are dynamic, instructive workshops and laboratories positioned at the leading edge of ethical inquiry in information ethics and other moral problems in economic development. The 1990s were a time of severe trial and soul-searching. Substantial intellectual and cultural resources are now available to identify and define ethical problems with rigor, to achieve a rough consensus on priorities, and to begin to design solutions in a nation replete with arm-chair philosophers and social engineers. Most importantly, the will to address ethical questions and habits of investigation into novel and multiple resources have been established.

## X. CONCLUSION

An overview of research priorities and projects in the past 20 years shows significant shifts in development priorities and information systems (Table XII).

At present, information systems are mixed blessings for developing countries. On the one hand, ICTs hold the promise of assisting balanced and accelerated economic development. On the other hand, ICTs are deepening the digital divide, erasing the modest comparative advantages which developing areas have held in cheap labor markets or other natural endowments.

Furthermore, the success of ICTs depends upon economic and political reform and the success of painful cultural disruptions. Economic developers cannot succeed without broad infrastructure improvements, or without external supplies of capital and technical assistance. Start-up costs, partnerships, increased competition, and the globalization of domestic markets are daunting challenges.

The development of information systems in developing economies is both critical and problematic. The resource shortages, inequities, and cultural obstacles which impede balance and wealth creation also influence the effectiveness of ICTs. The limits of protectionism, tradition-based solutions, and cultural congruence are clear. A timely movement toward open, extended, and integrated ICTs is mandated. The international, regional, and national programs for ICT development require unprecedented investments but promise to reap massive, long overdue benefits. The campaign to spread the advantages of the informa-

**Table XII** IS Research Areas

| 1980s | 2000s |
|---|---|
| EDI | e-commerce |
| National computer industries (Brazil) | India's software industry |
| Impact on rural/minority/female population | Technological dualism |
| Brain drain | Wireless |
| Networks | Ethics and equity |
| TQM | The digital divide |
| Globalization | "Soft" factors |
| Joint ventures | Information quality |
| Multinational investment | Open, integrated, and extended systems |
| Systems design and maintenance | International assistance |

tion revolution to the poorest areas of the world is one of the salient challenges of this generation.

## SEE ALSO THE FOLLOWING ARTICLES

Digital Divide, The • Digital Goods: An Economic Perspective • Economic Impacts of Information Technology • Electronic Commerce • Ethical Issues • Future of Information Systems • Global Information Systems • Globalization • Globalization and Information Management Strategy • Internet, Overview • People, Information Systems Impact on

## BIBLIOGRAPHY

Amoako, K.Y. (1996). Africa's information society initiative: An action framework to build Africa's information and communication infrastructure. *Information and Communication for Development Conference Proceedings, Midrand, South Africa.*

Avgerou, C., and Walsham, G. (2000). *Information technology in context.* Aldershot: Ashgate.

Castells, M. (2000). *The rise of the network society,* Vol. I. New York: Blackwells.

Chepaitis, E. (1990). Cultural constraints in the transference of information technology to third world countries, in *International Science and Technology: Philosophy, Theory and Policy* (Mekki Mtewa, Ed.). New York: St. Martin's.

Chepaitis, E. (1993). After the command economy: Russia's information culture and its impact on information resource management. *Journal of Global Information Management,* Vol. I, no. 2.

Chepaitis, E. (2001). The criticality of information ethics in emerging economies: Beyond privacy and piracy. *Journal of Information Ethics.*

Chepaitis, E. (2002). E-commerce and the information environment in an emerging economy: Russia at the turn of the century, in *Global Information Technology and Electronic Commerce: Issues for the New Millennium.* (Prashant Palvia *et al.* Eds.), 53–73. Marietta, GA: Ivy League Publishing.

Davenport, T. (1997). *Information ecology.* New York: Oxford Univ. Press.

Fukuyama, F. (1995). *Trust: The social virtues and the creation of prosperity.* New York: Free Press.

Hedley, R. A. (2000). The information age: Apartheid, cultural imperialism, or global village? in *Social Dimensions of Information Technology: Issues for the New Millennium.* (G. David Garson, Ed.), 278–290. Hershey, PA: Idea Group Publishing.

Iansiti, M. (1998). *Technology integration: Making choices in a dynamic world.* Cambridge, MA: Harvard Univ. Press.

Kannon, P.K., Change, A.-M., and Whinston, A. B. Electronic communities in e-business: Their roles and issues. *Information Systems Frontiers,* Vol. 1, No. 4, 415–426.

Tallon, P. P., and Kraemer, K. L. (2000). Information technology and economic development: Ireland's coming of age with lessons for developing countries. *Journal of Global Information Technology Management,* Vol. 3, No. 2, 4–23.

Wild, J. L., Wild, K. L., and Han, J. C.Y. (2000). *International business: An integrated approach: E-business edition.* New York: Prentice Hall.

World Bank Data Profiles. (July, 2001). *World Bank Group.*

# Digital Divide, The

**Randal D. Pinkett**
*Building Community Technology (BCT) Partners, Inc.*

## GLOSSARY

**community-based organization (CBO)** Private, nonprofit organizations that are representative of segments of communities.

**community building** An approach to community revitalization that is focused on strengthening the capacity of residents, associations, and organizations to work, individually and collectively, to foster and sustain positive neighborhood change.

**community content** The availability of material that is relevant and interesting to some target audience (e.g., low-income residents) to encourage and motivate the use of technology.

**community network** Community-based electronic network services, provided at little or no cost to users.

**community technology** Community-based initiatives that use technology to support and meet the goals of a community.

**community technology center (CTC)** Publicly accessible facilities that provide computer and Internet access as well as technical instruction and support.

**community telecenter** Similar to a community technology center, but community telecenter is a term more commonly used in remote or rural areas outside of the United States.

**digital divide** A phrase commonly used to describe the gap between those who benefit from new technologies and those who do not.

**Free-Net** Loosely organized, community-based, volunteer-managed electronic network services. They provide local and global information sharing and discussion at no charge to the Free-Net user or patron.

**ICT** Information and communications technology including computers and the Internet.

**National Information Infrastructure (NII)** An interconnected network of computers, databases, handheld devices, and consumer electronics.

**nongovernmental organization (NGO)** Similar to CBOs, whereas NGO is a term more commonly used outside of the United States.

## I. THE DIGITAL DIVIDE

The digital divide is a phrase commonly used to describe the gap between those who benefit from new technologies and those who do not. The phrase was first popularized by the National Telecommunications and Information Administration (NTIA) in the U.S. Department of Commerce in their 1995 report "Falling through the Net: A Survey of the Have Nots in Rural and Urban America." Thereafter, the NTIA released three additional reports: "Falling through the Net. II. New Data on the Digital Divide" in 1998, "Falling through the Net. III. Defining the Digital Divide" in 1999, and "Falling through the Net. IV. Toward Digital Inclusion" in 2000. In their most recent report, the NTIA wrote:

> A digital divide remains or has expanded slightly in some cases, even while Internet access and computer ownership are rising rapidly for almost all groups. For example, the August 2000 data show that noticeable divides still exist between those with different levels of income and education, different racial and ethnic groups, old and young, single and dual-parent families, and

> those with and without disabilities. . . . Until everyone has access to new technology tools, we must continue to take steps to expand access to these information resources.

Excerpts from the 1999 NTIA report include:

1. *Income*—Households with incomes of $75,000 and higher were more than *9 times* as likely to have a computer at home (see Fig. 1), and more than *20 times* as likely to have access to the Internet than those with incomes of $5000 or less (see Fig. 2).
2. *Education*—The percentage-point difference between those with a college education or better, when compared to those with an elementary school education, was as high as 63% for computer penetration (see Fig. 3), and 45% for Internet penetration (see Fig. 4).
3. *Race*—Black and Hispanic households were approximately one-half as likely as households of Asian/Pacific Islander descent, as well as White households, to have a home computer (see Fig. 5), and approximately one-third as likely as households of Asian/Pacific Islander descent, and roughly two-fifths as likely as White households, to have home Internet access (see Fig. 6).
4. *Geography*—Americans living in rural areas lagged behind those in urban areas and the central city, regardless of income level. For example, at the lowest level of income ($5000 and below), those in urban areas and the central city were almost one and a half times as likely to have a home computer (see Fig. 7) and more than twice as likely to have Internet access than those in rural areas (see Fig. 8).
5. *Income and race*—For households earning between $35,000 and $74,999, 40.2% of Blacks and 36.8% of Hispanics owned a computer, compared to 55.1% of Whites (see Fig. 9), while for households earning between $15,000 and $34,999, 7.9% of Blacks and 7.6% of Hispanics had Internet access, compared to 17% of Whites (see Fig. 10). A similar pattern emerged in each income category.

Corroborating evidence can be drawn from a number of related studies. In 1995, RAND's Center for Information Revolution Analyses (CIRA) published the results of a 2-year study entitled, "Universal Access to E-Mail: Feasibility and Societal Implications." Consistent with the NTIA reports, they found "large differences in both household computer access and use of network services across income categories. . . large differences in household computer access by educational attainment . . . [and] rather large and persistent differences across race/ethnicity in both household computer access and network services usage."

In 1997, Bellcore released the results of a national public opinion survey entitled "Motivations for and Barriers to Internet Usage: Results of a National Pub-

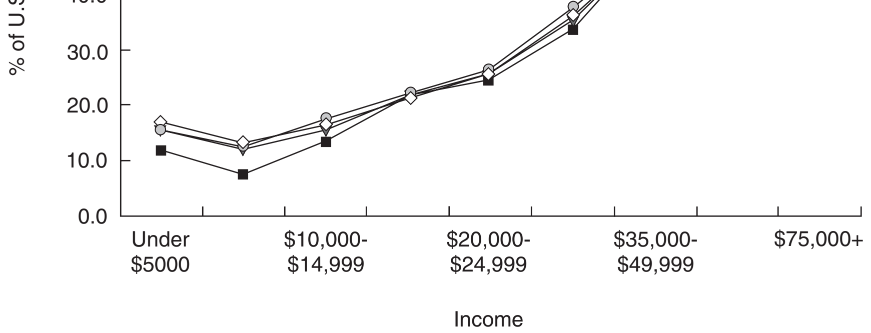


**Figure 1** The 1998 computer penetration rates by income. Data from the U.S. Department of Commerce.

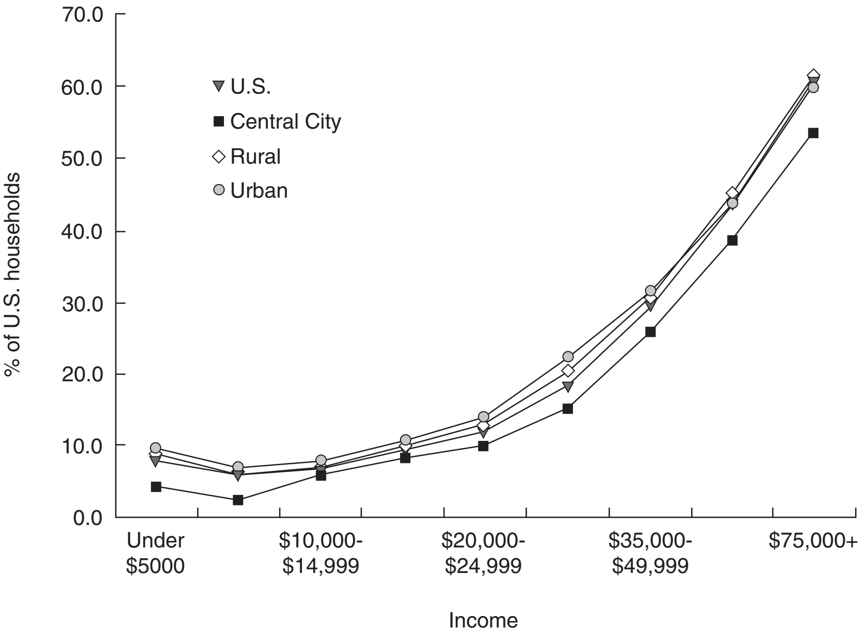


**Figure 2** The 1998 Internet penetration rates by income. Data from the U.S. Department of Commerce.

lic Opinion Survey." In similar fashion to the NTIA, they found that a disproportionately high 58% of those with a household income below $25,000 reported a lack of awareness of the Internet.

The Spring 1997 CommerceNet/Nielsen Internet Demographic Study (IDS), conducted in December 1996/January 1997 by Nielsen Media Research, also confirmed the NTIA's observations. This study was the first to collect data on patterns of use with computers and communications technologies as a function of income and race. In 1998, using the IDS data, Vanderbilt University released a report, "The Impact

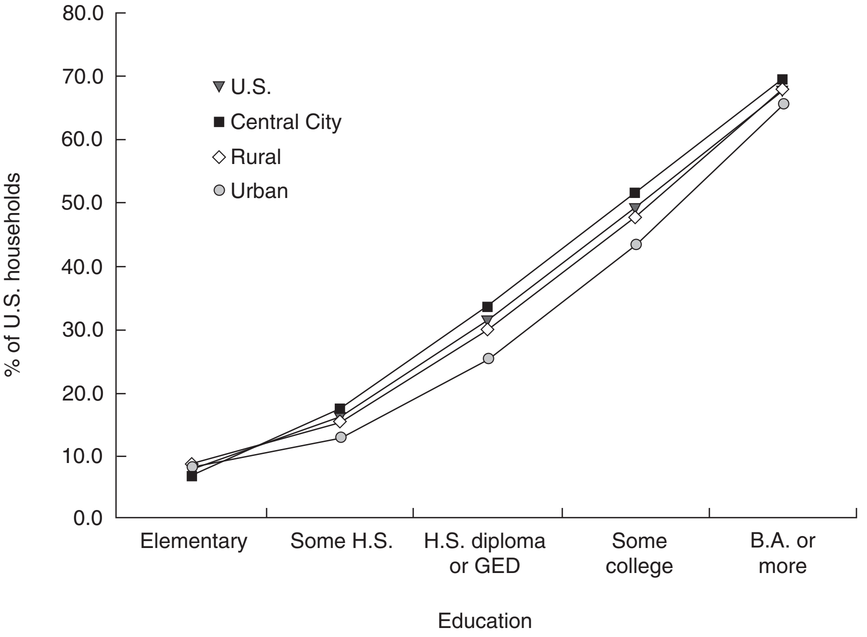


**Figure 3** The 1998 computer penetration rates by education. Data from the U.S. Department of Commerce.

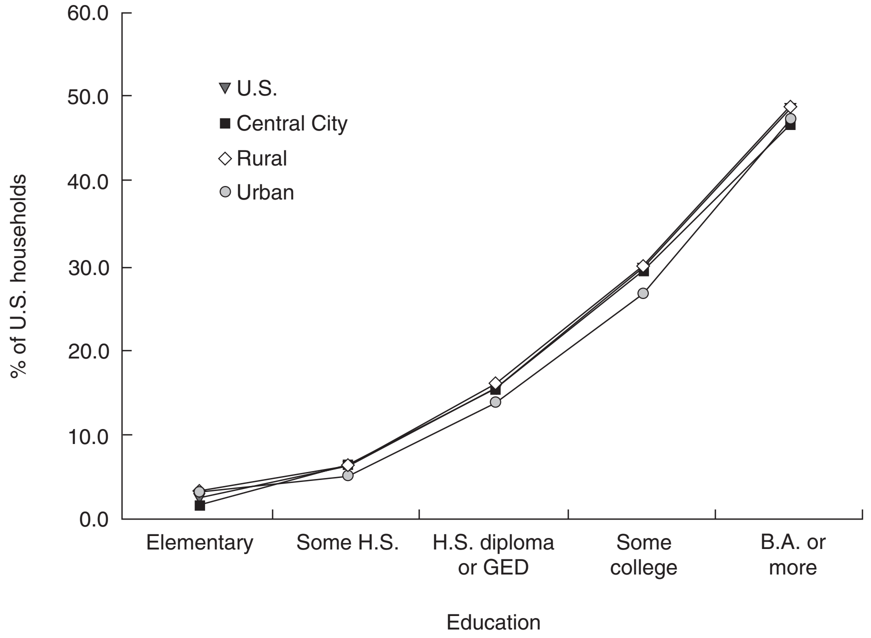


**Figure 4** The 1998 Internet penetration rates by education. Data from the U.S. Department of Commerce.

of Race on Computer Access and Use," that examined differences in PC access and Web use between African-Americans and Whites. The most interesting finding from their study was that income did not explain race differences in home computer ownership. While 73% of White students owned a home computer, only 33% of African-American students owned one, a difference that persisted when a statistical adjustment was made for the students' reported household income. They concluded that "in terms of students' use of the Web,

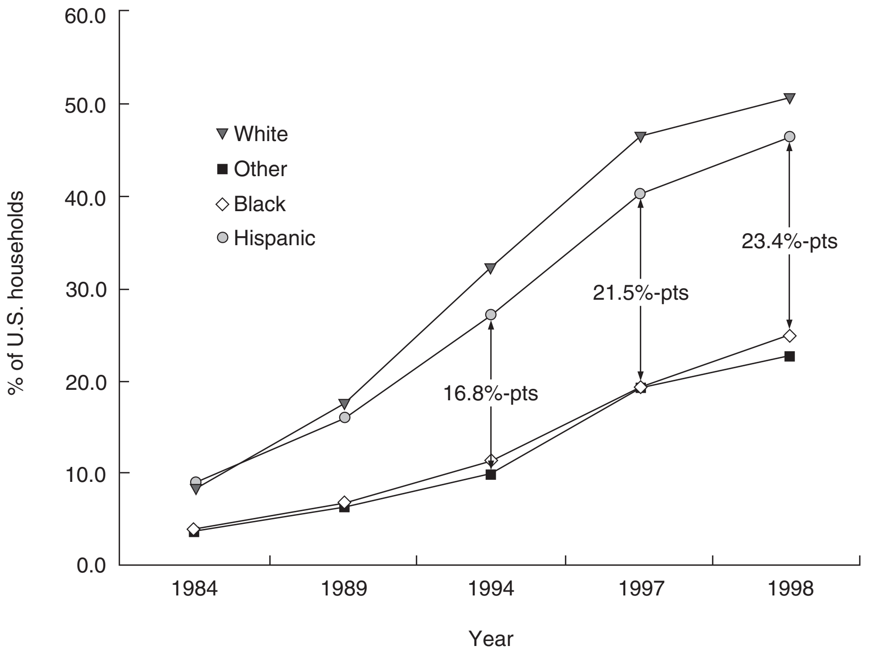


**Figure 5** The 1998 computer penetration rates by race. Data from the U.S. Department of Commerce.

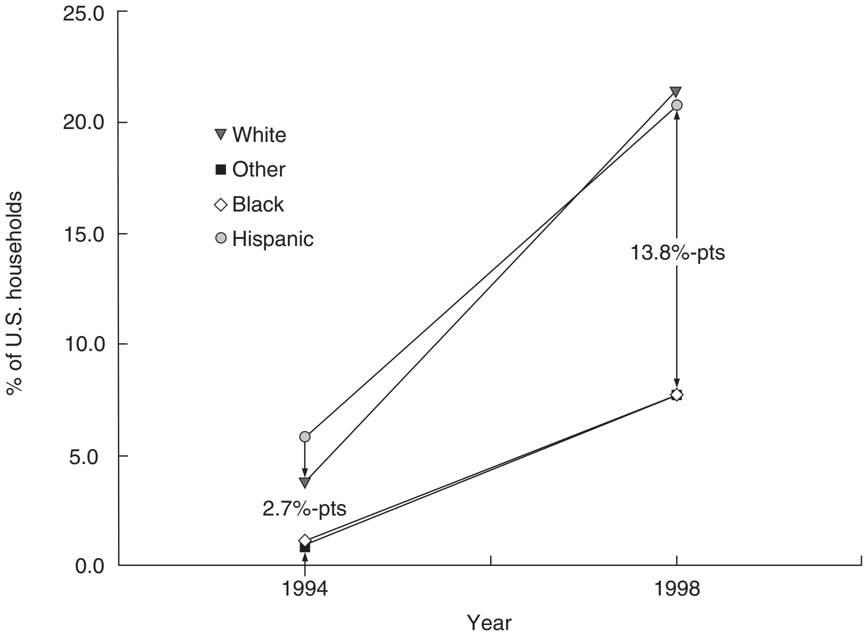


**Figure 6** The 1998 Internet penetration rates by race. Data from the U.S. Department of Commerce.

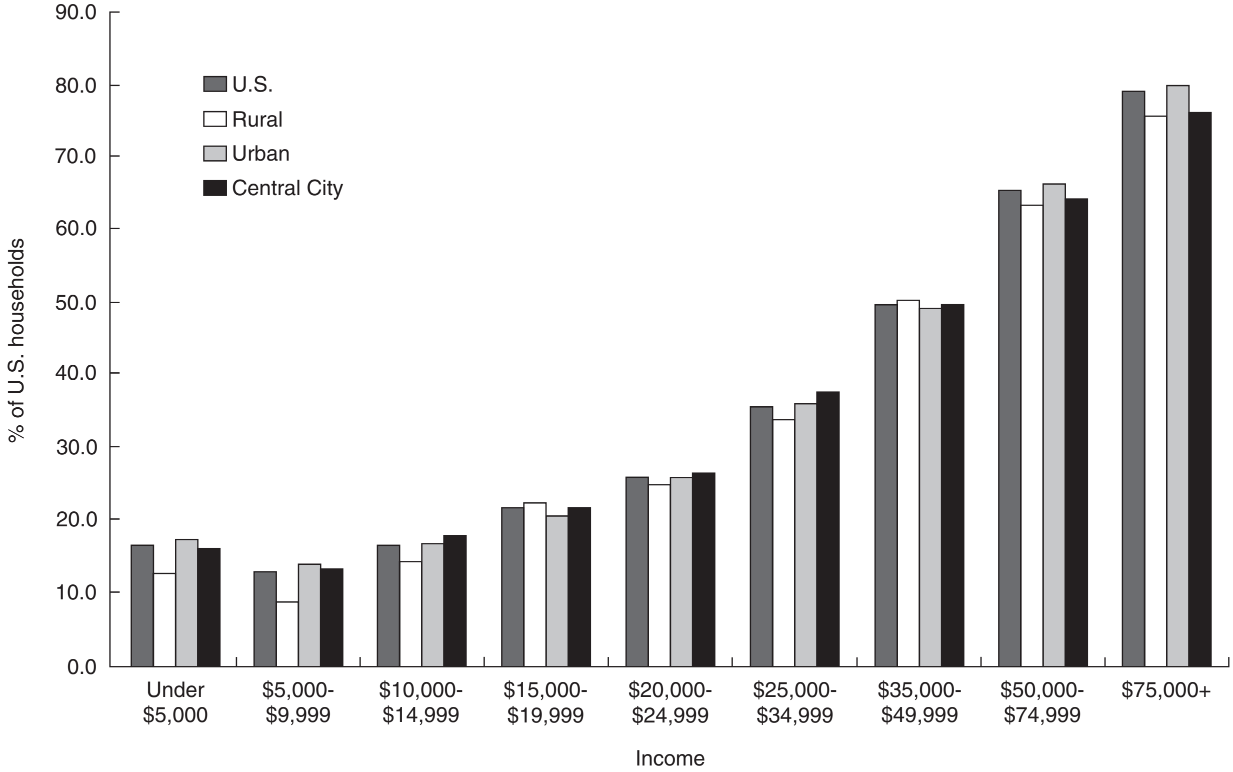


**Figure 7** The 1998 computer penetration rates by income by geographic location. Data from the U.S. Department of Commerce.

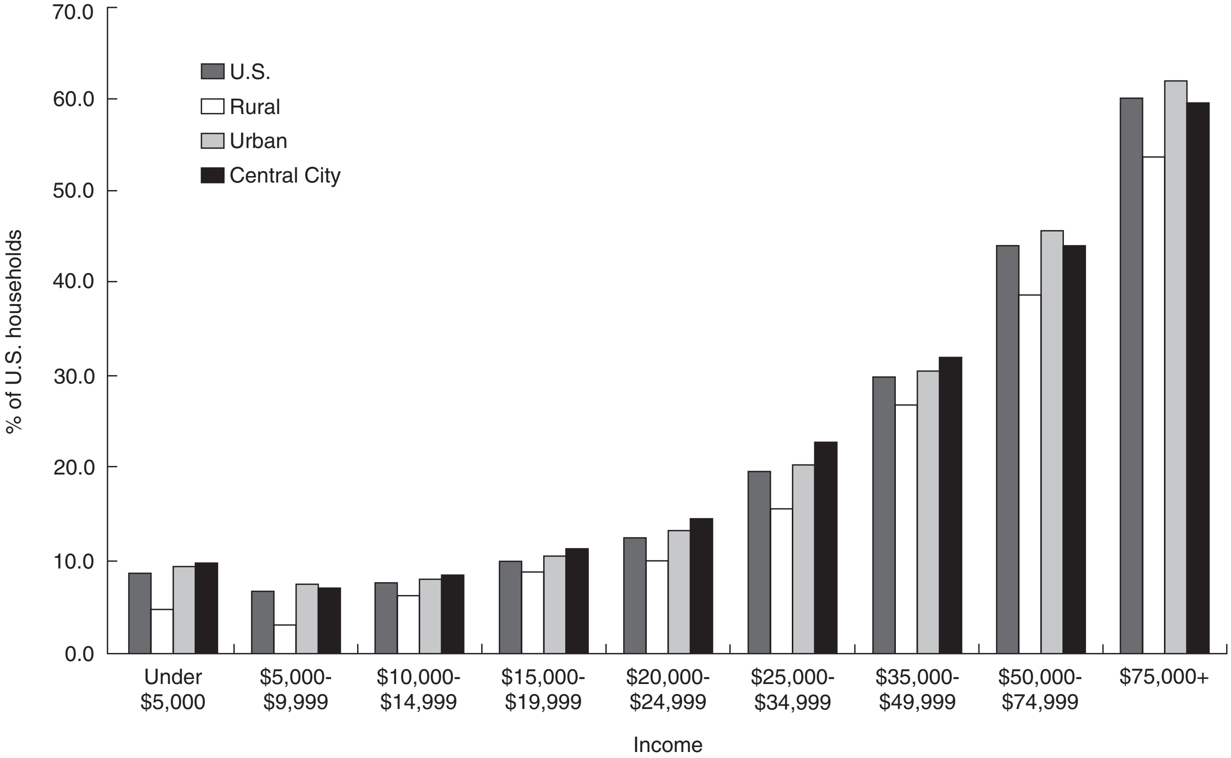


**Figure 8** The 1998 Internet penetration rates by income by geographic location. Data from the U.S. Department of Commerce.

particularly when students do not have a home computer, race matters." In 2000, Stanford University released a preliminary report, "Internet and Society," which focused on the digital divide, but with a noticeably different explanation for the problem. Focusing on the Internet solely (to the exclusion of computer ownership), they concluded that education and age were the most important factors facilitating or inhibiting access. According to their analysis, "a college education boosts rates of Internet access by well over 40 percentage points compared to the least educated group, while people over 65 show a more than 40 percentage point drop in their rates of Internet access compared to those under 25." Regardless of the explanatory construct, all of these statistics suggest that the digital divide is a very real phenomenon.

In addition to the aforementioned studies, the digital divide has been framed along a number of other dimensions, including inequities in computer and Internet access, use, and curriculum design between urban and suburban schools, or predominantly minority and predominantly White schools; the lack of participation of underrepresented groups in computer-related and information technology-related fields and businesses, and the dearth of online content, material, and applications, geared to the needs and interests of low-income and underserved Americans.

## II. DIGITAL DIVIDE POLICY

The United States policy related to the digital divide originates within the context of *universal service* which, at its inception, referred to universal access to "plain old telephone service" (POTS). This was later expanded to include information and communications technology (ICT). In 1913, the Kingsbury Commitment placed the national telephony network of American Telephone and Telegraph (AT&T) under control of the federal common carrier. In exchange, AT&T received further protection from competition while agreeing to provide universal access to wiring and related infrastructure. The Communications Act of 1934 extended this accord and ensured affordable telephone service for all citizens by regulating pricing, subsidizing the cost of physical infrastructure to re-

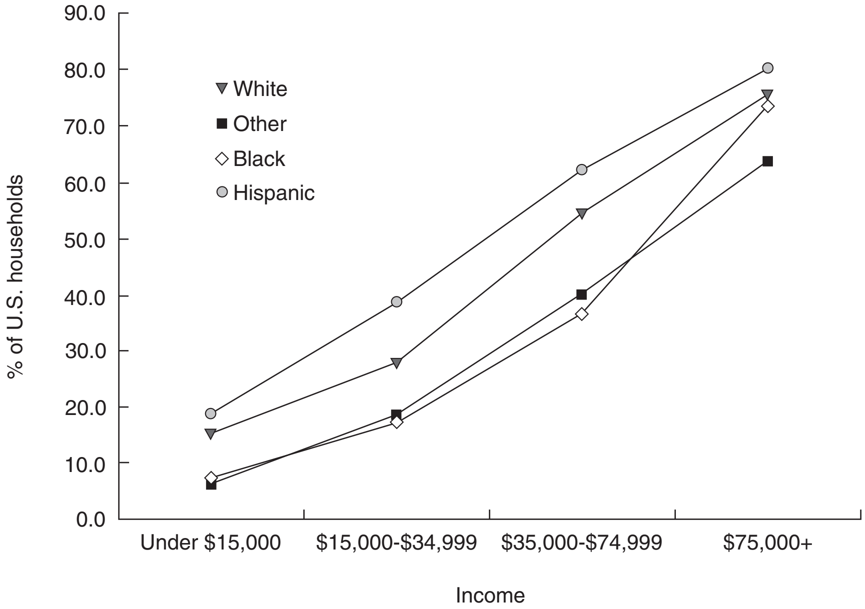


**Figure 9** The 1998 computer penetration rates by income by race. Data from the U.S. Department of Commerce.

mote areas, and establishing the Federal Communications Commission (FCC). In the Communications Act of 1934, the role of the FCC was described as follows:

> For the purpose of regulating interstate and foreign commerce in communication by wire and radio so as to make available, so far as possible to all the people of the United States, a rapid, efficient, nation-wide, and world-wide wire and radio communications service with adequate facilities at reasonable charges.

The FCC would then play the central role in ensuring that national telephone service was both available and affordable, whereas it would be almost 60 years before telecommunications policy was revisited in the U.S. In 1993, the Clinton Administration released a report,

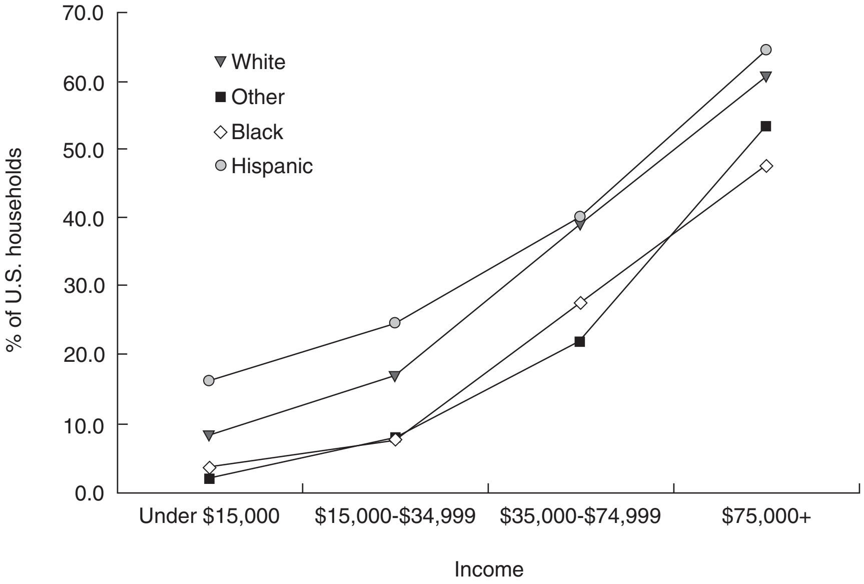


**Figure 10** The 1998 Internet penetration rates by income by race. Data from the U.S. Department of Commerce.

"National Information Infrastructure: Agenda for Action," which identified strategies for expanding the National Information Infrastructure (NII)—an interconnected network of computers, databases, handheld devices, and consumer electronics. The report addressed the following areas:

1. Promoting private sector investment, through appropriate tax and regulatory policies
2. Extending the universal service concept to ensure that information resources were available at affordable prices
3. Acting as a catalyst to promote technological innovation and new applications
4. Promoting seamless, interactive, user-driven operation of the NII
5. Ensuring information security and network reliability
6. Improving management of the radio frequency spectrum
7. Protecting intellectual property rights
8. Coordinating with other levels of government and with other nations
9. Providing access to government information and improving government procurement

Meanwhile, in 1994, the NTIA in the U.S. Department of Commerce began gathering data related to computer and Internet access, by adding a series of questions to the U.S. Census Bureau's "Current Population Survey." This resulted in the first of the four "Falling through the Net" reports that highlighted the digital divide—as defined in the report as the gap "between those who have access to and use computers and the Internet." Soon after, Congress passed the Telecommunications Act of 1996 with the aim of deregulating the telecommunications industry, thus promoting competition, lowering the overall cost of services to consumers, and expanding the notion of universal service through the inclusion of additional mechanisms that supported its provision. The Telecommunications Act of 1996 also established one of several major policy initiatives aimed at closing the digital divide—the Universal Service Fund, also known as the "E-Rate" Program.

The Universal Service Fund was administered by the Universal Service Administration Company (USAC), a private, not-for-profit corporation responsible for providing every state and territory in the United States with access to affordable telecommunications services. All of the country's communities—including remote communities—such as rural areas, low-income neighborhoods, rural health care providers, public and private schools, and public libraries, were eligible to seek support from the Universal Service Fund. To accomplish this task, USAC administered four programs: the High Cost Program, the Low Income Program, the Rural Health Care Program, and the Schools and Libraries Program. Each of these programs provided affordable access to modern telecommunications services for consumers, rural health care facilities, schools, and libraries, regardless of geographic location or socioeconomic status, via 20 to 90% discounts on telecommunications services, Internet access, and internal connections. In addition to USAC, a few of the other major policy initiatives of the Clinton Administration included the following:

1. *U.S. Department of Commerce, Technology Opportunities Program*—The Technology Opportunities Program promoted the widespread availability and use of advanced telecommunications technologies in the public and nonprofit sectors. As part of the NTIA, TOP gave grants for model projects demonstrating innovative uses of network technology. TOP evaluated and actively shared the lessons learned from these projects to ensure the benefits were broadly distributed across the country, especially in rural and underserved communities.
2. *U.S. Department of Education, Community Technology Center (CTC) Program*—The purpose of the Community Technology Centers program was to promote the development of model programs that demonstrated the educational effectiveness of technology in urban and rural areas and economically distressed communities. These community technology centers provided access to information technology and related learning services to children and adults.
3. *U.S. Department of Housing and Urban Development, Neighborhood Networks Program*—The Neighborhood Networks initiative encouraged the establishment of resource and community technology centers in privately owned apartment buildings that received support from HUD to serve low-income people. The centers offered computer training and educational programs, job placement, and a diverse array of other support. The goal of Neighborhood Networks was to foster economic opportunity and encourage life-long learning.
4. *National Science Foundation, Connections to the Internet Program*—The Connections to the Internet program encouraged U.S. research and education institutions and facilities to connect to

the Internet and to establish high performance connections to support selected meritorious applications. This included three connection categories: (1) connections for K-12 institutions, libraries, and museums that utilized innovative technologies for Internet access; (2) new connections for higher education institutions; and (3) connections for research and education institutions and facilities that had meritorious applications with special network requirements (such as high bandwidth and/or bounded latency) that could not readily be met through commodity network providers.

5. *U.S. Department of Agriculture, Construction and Installation of Broadband Telecommunications Services in Rural America*—The Rural Utilities Service (RUS) was a loan program and the availability of loan funds under this program to finance the construction and installation of broadband telecommunications services in rural America. The purpose of the program was to encourage telecommunications carriers to provide broadband service to rural consumers where such service did not currently exist. This program provided loan funds, on an expedited basis, to communities up to 20,000 inhabitants to ensure rural consumers enjoyed the same quality and range of telecommunications services that were available in urban and suburban communities.
6. *U.S. General Services Administration, Computers for Learning Program*—The Computers for Learning (CFL) program transferred excess Federal computer equipment to schools and educational nonprofit organizations, giving special consideration to those with the greatest needs. Federal agencies used the CFL web site to connect the registered needs of schools and educational nonprofit organizations with available government computer equipment.

## III. CLOSING THE DIGITAL DIVIDE

Three primary models have emerged for closing the digital divide. These efforts fall under the heading of *community technology,* or community-based initiatives that use technology to support and meet the goals of a community. The first model is *community networks,* or community-based electronic network services, provided at little or no cost to users. The second model is *community technology centers* (CTCs), or publicly accessible facilities that provide computer and Internet access as well as technical instruction and support. The third model is *community content,* or the availability of material that is relevant and interesting to some target audience (e.g., low-income residents) to encourage and motivate the use of technology.

These approaches can be classified according to what they provide: hardware, software, training, infrastructure, online access, or content. They can also be classified according to the groups they target: individuals, schools, youth, community organizations, and the general public, or specific groups such as a neighborhood, racial or ethnic minorities, the homeless, and the elderly. Each model is described in greater detail below.

### A. Community Networks

*Community networks* are community-based electronic network services, provided at little or no cost to users. In essence, community networks establish a new technological infrastructure that augments and restructures the existing social infrastructure of the community.

Most community networks began as part of the Free-Net movement during the mid-1980s. According to the Victoria Free-Net Association, Free-Nets are “loosely organized, community-based, volunteer-managed electronic network services. They provide local and global information sharing and discussion at no charge to the Free-Net user or patron.” This includes discussion forums or real-time chat dealing with various social, cultural, and political topics such as upcoming activities and events, ethnic interests, or local elections, as well as informal bartering, classifieds, surveys and polls, and more. The Cleveland Free-Net, founded in 1986 by Dr. Tom Grundner, was the first community network. It grew out of the “St. Silicon’s Hospital and Information Dispensary,” an electronic bulletin board system (BBS) for health care that evolved from an earlier bulletin board system, the Chicago BBS.

In 1989, Grundner founded The National Public Telecomputing Network (NPTN) which, according to the Victoria Free-Net Association, “evolved as the public lobbying group, national organizing committee, and public policy representative for U.S.-based Free-Nets and [contributed] to the planning of world-wide Free-Nets.” NPTN grew to support as many as 163 affiliates in 42 states and 10 countries. However, in the face of rapidly declining commercial prices for Internet connectivity, and a steady increase in the demands to maintain high-quality information services, NPTN (and many of its affiliates) filed for bankruptcy in 1996. While a number of Free-Nets still exist today, many of the community networking initiatives that are presently active have incorporated some aspects of the remaining models for community technology—community technology centers and community content.

The aforementioned study at RAND's CIRA involved the evaluation of the following five community networks: (1) Public Electronic Network, Santa Monica, CA; (2) Seattle Community Network, Seattle, WA; (3) Playing to Win Network, New York, NY; (4) LatinoNet, San Francisco, CA; and (5) Blacksburg Electronic Village (BEV), Blacksburg, VA. Their findings included increased participation in discussion and decision making among those who were politically or economically disadvantaged, in addition to the following results:

- *Improved access to information*—Computers and the Internet allowed "individuals and groups to tap directly into vast amounts and types of information from on-line databases and from organizations that advertise or offer their products and services online."
- *Restructuring of nonprofit and community-based organizations*—Computers and the Internet assisted these organizations in operating more effectively.
- *Delivery of government services and political participation*—Computers and the Internet promoted a more efficient dissemination of local and federal information and services and encouraged public awareness of, and participation in, government processes.

Examples of other community networks include Big Sky Telegraph, Dillon, Montana; National Capital Free-Net, Ottawa, Ontario; Buffalo Free-Net, Buffalo, New York; and PrairieNet, Urbana-Champaign, Illinois.

## B. Community Technology Centers

*Community technology centers (CTCs),* or community computing centers, are publicly accessible facilities that provide computer and Internet access, as well as technical instruction and support. According to a 1999 study by the University of Illinois at Urbana-Champaign, CTCs are an attractive model for a number of reasons. First, they are cost-effective when compared to placing computers in the home. Second, responsibility for maintaining computer resources is assumed by an external agent. Third, knowledgeable staff members are present to offer technical support and training. Fourth and finally, peers and other community members are present, creating a pleasant social atmosphere. Consequently, CTCs are, by far, the most widely employed strategy to-date for community technology initiatives.

For more than two decades, significant public and private funds have been invested in the development of CTCs nationwide, including the Intel Computer Clubhouse Network, the Community Technology Centers' Network (CTCNet), PowerUp, the U.S. Department of Education CTC program, the U.S. Department of Housing and Urban Development Neighborhood Networks (NN) program, and more. CTCs have been the focus of numerous studies relating to computer and Internet access and use, and the factors influencing their effectiveness have been well-researched and documented, including determining space; selecting hardware, software, and connectivity; scheduling and outreach; budgeting; funding, and more.

The Community Technology Centers' Network (CTCNet), housed at Educational Development Center, Inc. (EDC), in Newton, Massachusetts, is a national membership organization that promotes and nurtures nonprofit, community-based efforts to provide computer access and learning opportunities to the general public and to disadvantaged populations. CTCNet is a network of more than 350 affiliate CTCs including multiservice agencies, community networks, adult literacy programs, job training and entrepreneurship programs, public housing facilities, YMCAs, public libraries, schools, cable television access centers, and after-school programs.

CTCNet has conducted two evaluations examining the impact of computers and the Internet on individuals and families. Their first evaluation, "Community Technology Centers: Impact on Individual Participants and Their Communities," was a qualitative study that involved semistructured interviews with 131 participants at five intensive sites: (1) Brooklyn Public Library Program, Brooklyn, NY; (2) Somerville Community Computing Center, Somerville, MA; (3) Old North End Community Technology Center (ONE), Burlington, VT; (4) New Beginnings Learning Center, Pittsburgh, PA; and (5) Plugged In, East Palo Alto, CA. The results of the study included:

- *Increased job skills and access to employment opportunities*—Individuals were able to access information and resources about job search and employment opportunities (14%), improve job skills including computer and literacy skills (38%), and consider new, higher-wage, career options that involved the use of technology (27%).
- *Education and improved outlook on learning*—Individuals gained access to lifelong learning

opportunities such as computer literacy and mathematics programs (15%), changed their goals for learning and educational attainment (e.g., decided to pursue a GED or more) (27%), and improved their outlook and perspective on learning (e.g., using the computer they "learned that they can learn") (27%).

- *Technological literacy as a means to achieve individual goals*—Individuals obtained greater computer awareness and new computer skills that increased their comfort with technology as a tool for accomplishing their goals (91%).
- *New skills and knowledge*—Individuals improved their reading and writing (37%), mathematics skills, and interest in science (8%).
- *Personal efficacy and affective outcomes*—Individuals achieved greater personal autonomy (18%) and feelings of pride and competence as a result of success with computers (e.g., decided to stay off drugs) (23%).
- *Use of time and resources*—Individuals found productive uses for their time (15%) which resulted in positive outcomes such as reduced reliance on public assistance (4%).
- *Increased civic participation*—Individuals identified new avenues for voicing their opinions on a range of social and political issues (5%), gained access to community, municipal, and government services and resources, and demonstrated greater interest in and engagement with current events (5%).

CTCNet's second evaluation, "Impact of CTCNet Affiliates: Findings from a National Survey of Users of Community Technology Centers," was a quantitative study that surveyed 817 people from 44 sites. This evaluation corroborated the findings from the 1997 study and also found that more than one-third of users with employment goals, half of users with educational goals, and more than half of users with goals of self-confidence and overcoming fear reported achieving their goals.

Examples of other CTCs include Computer Clubhouse, Boston, Massachusetts; Austin Learning Academy, Austin, Texas; PUENTE Learning Centers, Los Angeles, California; New Beginnings Learning Center, Pittsburgh, Pennsylvania; and West Side Community Computing Center, Cleveland, Ohio.

## C. Community Content

*Community content* refers to the generation and availability of local material that is relevant and interesting to a specific target audience (e.g., low-income residents) to encourage and motivate the use of technology. Community content can be broadly classified along two dimensions: *information* vs *communication,* and *active* vs *passive.* The information vs communication dimension highlights the Internet's ability to both deliver information and facilitate communication. Interestingly, a 1997 study at Carnegie Mellon University found that people use the Internet more for communication and social activities than they do for information purposes. A simple example of the difference between these two forms of community content is the difference between reading and writing about community-related matters (information), and discussing and dialoguing about community-related matters (communications).

Information-based community content takes the form of databases and documents that can be accessed online such as a directory of social service agencies, a listing of recommended websites, or a calendar of activities and events. Communications-centered community content takes the form of interactive, synchronous tools such as chat rooms and instant messaging, or asynchronous tools such as listservs (e-mail lists) and discussion forums. Here, the distinguishing factor when contrasted with other forms of content is that the nature of the information or communication exchange is solely focused on, or of use to, members of the community.

The active vs passive dimension, at one extreme, positions community members as the active producers of community content, while at the other extreme, it positions community members as the passive recipients of community content, with varying degrees of each designation found at each point along the continuum. A simple example that highlights the distinction between these two orientations toward community content is the difference between browsing a community website (passive) and building a community website (active).

A passive disposition is static, unidirectional, and sometimes described as "one-to-many" because content is generated by a third party (one) and delivered to the community (many). It typically manifests itself in the form of centralized, one-way repositories of information that can be accessed by community members such as an information clearinghouse of city or municipal services; an online entertainment guide that lists movies, shows, live performances, and restaurants; or a web portal of local news, weather, sports, etc. Here, the distinguishing factor is that although very little, if any, content is produced by the community, it is still intended for the community. An active disposition is dynamic, multidirectional, and often described as "many-to-many," because content is generated by the community (many) for the community (many). It typically manifests itself in the form of multiple-way, interactive communication and information exchange between end-users such as an online, neighborhood-based, barter and exchange network, an e-mail listserv for the purpose of online organizing and advocacy, or a community-generated, web-based, geographic information system (GIS) that maps local resources and assets. Here, the distinguishing factor is that most, if not all, content is produced by the community.

Community content was the focus of a report authored by the Children's Partnership, a children's advocacy, nonprofit organization, entitled "Online Content for Low-Income and Underserved Americans." In the report, community content was defined according to the following five categories:

- Information that is more widely available
- Information that can be customized by the user
- Information that flows from many to many
- Information that allows for interaction among users
- Information that enables users to become producers of information.

Based on an evaluation that included discussion with user groups, interviews with center and community network directors, interviews with other experts, and analysis of the web, they concluded the following with respect to low-income and underserved populations and existing Internet content: (1) a lack of local information, (2) literacy barriers, (3) language barriers, and (4) a lack of cultural diversity. This suggests that while the web has emerged as a valuable resource for mainstream users, additional effort must be made to make the Internet more attractive to local community residents and ethnic and cultural groups, as well as more accessible to users with limited literacy, users with disabilities, and nonnative English speakers. Community content is an emerging strategy for community technology initiatives, and additional work will be required to overcome these barriers.

Notable community content sites include CTCNet (http://www.ctcnet.org), the Digital Divide Network (http://www.digitaldividenetwork.org), the America Connects Consortium (http://www.americaconnects.net), the Children's Partnership's Content Bank (http://www.contentbank.org), and the Community Connector (http://databases.si.umich.edu/cfdocs/community/index.cfm).

## IV. THE INTERNATIONAL DIGITAL DIVIDE

Although the digital divide was initially recognized as an issue within the domestic United States, the phrase was very quickly adopted to encompass the disparities in information and communications technology (ICT) access and use across the globe. The following is an overview of the digital divide in selected countries and continents.

### A. Africa

Internet access arrived in Africa circa 1998, which clearly explains why most African countries have experienced relatively low levels of computer and Internet penetration when compared to other countries in their region. In 1999, the United Nations Development Program (UNDP) released the "World Report on Human Development," which identified Africa as having 0.1% of the hosts on the Internet and the U.S. as having 26.3%. In 2000, Nua estimated that there were 407.1 million people online, of which 3.11 million were in Africa (less than 1%) from among its 738 million people. That same year, the International Telecommunications Union (ITU) reported that while Africa represented 12.8% of the world's population, it had just 2% of wired telephones, 1.5% of wireless telephones, and 1.5% of personal computers.

Efforts to close the digital divide and expand ICT diffusion in Africa were bolstered in April 1995, when the Lisbon Center for the Study of Africa (CEA), ITU, United Nations Educational, Scientific and Cultural Organization (UNESCO), International Development Research Center (IDRC), and Bellanet International held the "African Regional Symposium on Telematics for Development" in Addis Abeba. One month later, at the annual meeting of the CEA, ministers from various African countries gathered and passed Resolution 795, "Building the African Speedway for Information," which led to the implementation of a national telecommunications infrastructure for the purpose of planning and decision making, and the establishment of a committee comprised of ICT experts charged with preparing Africa for the digital age. One year later, the CEA Ministers passed Resolution 812, which established an even broader initiative known as the "African Information Society Initiative (AISI)." AISI, the recognized plan for fostering an environment within Africa that could support broad deployment of ICT throughout the continent, was based on the following nine points:

1. Develop nation plans for building information and communication infrastructure
2. Eliminate legal and regulatory barriers to the use of information and communications technologies
3. Establish and enable an environment to foster the free flow and development of information and communications technologies in society
4. Develop policies and implement plans for using information and communications technologies
5. Introduce information and communications applications in the areas of highest impact on socioeconomic development at the national level
6. Facilitate the establishment of locally based, low-cost, and widely accessible Internet services and information content
7. Prepare and implement plans to develop human resources in information and communications technologies
8. Adopt policies and strategies to increase access to information and communications facilities with priorities in servicing rural areas, grassroots society, and other disenfranchised groups, particularly women and youth
9. Create and raise awareness of the potential benefits of African information and communications infrastructure

In 1998, following the ITU Africa Telecomm Conference, African Telecomm Ministers established the “African Connection” and the African Telecomms Union (ATU) to further promote strategies to bring ICT to the continent on a large-scale basis. This was followed by a series of related conferences and forums focused on building the information and communications infrastructure of Africa.

Alongside the efforts of AISI and ATU, certain African countries have made noticeable progress toward strengthening the availability and use of ICT within their regions, mostly notably South Africa. According to Nua, in 2000 the largest share of Internet users in Africa was located in Southern Africa, of which South Africa alone represented roughly three-quarters of the continent’s total users, followed by Zimbabwe and Botswana. During the changeover years to the post-apartheid South Africa, the Center for Developing Information and Telecommunications Policy was established in coordination with the transitioning government of the African National Congress (ANC). After the ANC came into complete power in 1994, a process was undertaken to understand and subsequently address the disparity in ICT in South Africa, which included the Information Society and Development Conference (ISAD) in May 1996. This led to the Telecommunications Act of 1996, which established an independent regulatory agency for ICT in South Africa and created the Universal Service Agency (USA), a public agency responsible for overseeing ICT-related initiatives throughout South Africa. Since then, a number of initiatives have been undertaken including training programs by the Department of Labor and efforts to bring computers into schools by 2005 sponsored by the Department of Education and led by nongovernmental organizations (NGOs) such as SchoolNet and the Multi-Purpose Community Centers Program. South Africa has also witnessed the emergence of various Internet Cafes (commercial access sites) and USA telecenters (public access sites) in addition to phoneshops, libraries equipped with computers and Internet access, and other programs that provide ICT training and support.

## B. Australia

In 2000, the Australian Bureau of Statistics (ABS) released figures concerning the digital divide in Australia, which determined that 40% of Australian households were connected to the Internet by the end of the year, while 50% of Australian adults had accessed the Internet during the past year. Their research also identified the following factors related to computer and Internet access: household income, showing that households with an annual income above $50,000 were almost twice as likely to have a computer (77%) and Internet (57%) at home than those below this figure (37 and 21%, respectively); region, showing that households located in metropolitan areas (i.e., the east coast of Australia) were more likely to have computer and Internet access (59 and 40%, respectively) when compared to nonmetropolitan areas (52 and 32%, respectively); family composition, showing that households with children under 18 were more likely to access computers and the Internet (48%) when compared to their counterparts without children under 18 (32%); age, showing that adults between the ages of 18 and 24 were the most likely to have a computer and Internet access at home (88%) and adults above the age of 55 were the least likely to do so; employment status, showing that employed adults were more than twice as likely to have a computer and Internet access (82%) than unemployed adults (38%); gender, showing that women (47%) were less likely than males (53%) to use computers and the Internet; and education, showing that computer and Internet access for adults with a bachelor's degree (64%) was more than twice the level of access among adults with a secondary school education (28%).

Subsequently, in June 2001, the National Office of Information Economy (NOIE) in the Commonwealth Government of Australia released a report entitled "Current State of Play," which examined the digital divide in Australia from an even broader perspective. Their report corroborated the findings from the ABS and focused on three areas: (1) readiness, or access to ICT as indicated by levels of penetration and infrastructure; (2) intensity, or the nature, frequency, and scope of ICT use; and (3) impacts, or the benefits of ICT to citizens, businesses, and communities. This information was then used to inform a variety of national, regional, and local initiatives aimed at closing the digital divide.

Nationally, NOIE worked with the Department of Communications, Information Technology and the Arts, through the Networking the Nation (NTN) project, which has provided online access primarily in nonmetropolitan areas, created a multicultural web-based portal with community information and content, established a national directory of organizations providing subsidized access and free computers, offered training programs aimed at women and the elderly, and more. Regionally and locally, NTN has funded a number of programs such as AccessAbility, an online resource to raise awareness concerning disabled users of the Internet; Farmwide, which provides assistance to residents who have to place long-distance telephone calls to access the Internet; Building Additional Rural Networks (BARN), which supports network infrastructure and innovative technology development in rural areas, and more. Finally, to specifically address the needs of indigenous people, NOIE has instituted several programs such as the Remote Islands and Communities Fund, which provides assistance for the ICT needs of people in remote islands and communities; the Open Learning Projects to Assist Indigenous Australians, which offers funds for educational packages designed to serve indigenous students and electronic networks to link indigenous postgraduate students with other academic institutions; and the Connecting Indigenous Community Links, which provides public Internet access sites in seven indigenous communities.

## C. Canada

Canada has been tracking statistics related to the digital divide through the "General Social Survey of Internet Use," and other similar surveys. The 2000 survey revealed that 53% of Canadians age 15 years and older had used the Internet in the last 12 months. The breakdown of Internet use in the last 12 months along various social, geographic, and demographic lines was 56% of men and 50% of women; 90% of people aged 15 to 19 and 13% of people aged 65 to 69; 30% of people with an annual income below $20,000 and 81% of people with an annual income above $80,000; 79% of people with a university education and 13% of people with less than a high school diploma; 55% of people in urban areas and 45% of people in rural areas; and as high as 61% of people in Alberta and British Columbia and as low as 44% of people in Newfoundland and New Brunswick.

Beginning in the early 1990s, Canada witnessed a concerted effort to foster greater levels of civic participation among its citizens by leveraging the affordances of ICT. In 1992, the Victoria Free-Net and National Capital Free-Net were founded, and modeled after the Cleveland Free-Net in the United States. These early Free-Nets were the predecessors for a number of community networking initiatives in Canada that aimed to engage citizens in the public sphere. In 1994, Telecommunities Canada was created as an umbrella and advocacy organization for community networks, and between 1995 and 1998, the number of Free-Nets in Canadian cities increased from approximately 24 to 60. However, much like the community networking movement in the United States, a number of these Free-Nets experienced financial difficulties that caused them to shift their focus or completely disband. Similarly, the Coalition for Public Information (CPI), a federally incorporated, nonprofit organization, was founded in 1993, as a coalition of organizations, public interest groups, and individuals with a mandate to foster universal access to affordable, useable information and communications services and technology. With tremendous activity around ICT advocacy and policy between 1993 and 1999, CPI has since become dormant in its efforts to take action related to these issues.

To further promote ICT access and use throughout the country, the Canadian government has played a very active role through its "Connecting Canadians" initiative. These efforts involved a variety of government-sponsored programs and activities aimed at making Canada the most connected country in the world. The initiative was based on the following six pillars:

- *Canada Online*—This included the creation of up to 10,000 public Internet access sites in rural, remote, and urban communities. It also involved

connecting Canada's public schools and libraries to the Internet and upgrading Canada's network infrastructure.
- *Smart Communities*—These initiatives were focused on promoting the use of ICT for community development such as better delivery of heath care, improved education and training, and enhanced business and entrepreneurial opportunities.
- *Canadian Content Online*—A series of activities were organized to digitize and make widely available information and content related to Canadian people, culture, and history. This also included new application and software development to support information and content delivery.
- *Electronic Commerce*—The Canadian Government worked with specific provinces, territories, businesses, and other community members to implement their Electronic Commerce Strategy, released in 1998, which identified strategies for promoting greater availability and use of electronic commerce as an integral component of Canada's economic landscape.
- *Canadian Governments Online*—A comprehensive effort to deliver government services electronically as a means to provide more efficient and effective access to information and better respond to the needs of Canadian citizens.
- *Connecting Canada to the World*—As an overarching endeavor, the Connected Canadians initiative aimed to establish Canada as a recognized leader in the digital age, so as to bolster the country's attractiveness to foreign investors and global businesses toward strengthening the overall economy.

## D. China

In China, the digital divide is a phrase that was first referenced in the Okinawa Charter in Global Information Society and referred to the new difference among groups and polarization among the rich and the poor because of the different level of information and communication technologies among different countries, areas, industries, enterprises, and groups in the process of global digitalization. According to the China Internet Network Information Center, by the end of 2000, telephone penetration in China had reached 398 million households (24.4%)—281 million wired and 117 million wireless—computer penetration had reached 30 million people (2.5%), and Internet penetration had reached 18 million people (1.5%), with noticeable differences between the more affluent areas of eastern China, such as Beijing, Shanghai, and Guangdong, and the less developed provinces of central and western China.

On September 25, 2000, the Chinese government took a first and comprehensive step toward telecommunications policy by enacting the Telecommunications Statutes. This was done to encourage increased investment in telecommunications infrastructure and promote greater competition in the telecommunications industry, toward lowering prices for these services. This included associated steps to break up the then-monopoly China Telecom into three separate companies for wired communications (China Telecom), wireless communications (China Mobile), and paging/satellite communications (transferred to Unicom). Furthermore, the Chinese Ministry of Information Industry (MII) began regulating access to the Internet while the Ministries of Public and State Security (MSS) began monitoring its use. This included Internet Information Services Regulations that banned the dissemination of information that could potentially subvert the government or endanger national security.

A number of initiatives were undertaken in China to close the digital divide. The Digital Alliance was formed as a result of the 2001 High Level Annual Conference of Digital Economy and Digital Ecology in Beijing. It included representatives from the telecommunications industry, media industry, and academia, with the aim of eliminating the gap between the developed countries and China, the eastern and western regions of China, and different industries, social, and demographic groups. This was accomplished by providing technological resources, assistance, consultation, and application development for enterprises and community technology centers throughout China.

A number of cities in China also implemented strategies to deliver information and content electronically. For example, Shanghai completed an information exchange network, international trade electronic data interchange (EDI) network, community service network, social insurance network, and electronic commerce network. Similarly, Beijing undertook a comprehensive online project that spanned the areas of e-commerce, e-government, social insurance and community service, and science, technology, and education. Finally, in 1999, the Chinese government released the "Framework of National E-Commerce Development" and the International E-Commerce Center of China under the Ministry of Foreign Trade announced the Western Information Service Project, both with the objective of promoting more widespread deployment and use of electronic commerce throughout the region.

## E. Europe

In Europe, efforts to close the digital divide have been very closely aligned with efforts to establish an "information society"—a new economy fueled by ICT—among the member countries of the European Union. Toward this end, the European Commission released the results of a multinational survey called "Measuring Information Society 2000," that examined ownership and use of digital technologies, interest and intent to purchase digital technologies, use of the Internet, and Internet connection speed, among others indicators, for each of the member countries of the European Union.

According to the report, looking across the entire European Union, desktop computer ownership stands at 35%, with measurable differences between men (39%) and women (32%), people age 15 to 24 (46%) and 55 and up (16%), people at lower and upper levels of educational attainment (16% for people age 15 at the completion of formal schooling and 53% for people age 20 and up at the completion of formal schooling), as well as people at the lower and upper quartile with respect to income (16 and 61%, respectively). Similarly, Internet access stood at 18%, with measurable differences between men (21%) and women (16%), people age 15 to 24 (23%) and 55 and up (8%), people at lower and upper levels of educational attainment (6% for people age 15 at the completion of formal schooling and 63% for people age 20 and up at the completion of formal schooling), as well as people at the lower and upper quartile with respect to income (8 and 37%, respectively).

Not surprisingly, the digital divide manifests itself to a greater or lesser extent among the various countries located in the region. For desktop computer ownership and Internet access, the countries with the highest levels of home penetration were the Netherlands (66 and 46%, respectively), Denmark (59 and 45%, respectively), Sweden (56 and 48%, respectively), Luxembourg (45 and 27%, respectively) and Finland (45 and 28%, respectively), whereas the countries with the lowest levels of penetration were Spain (34 and 10%, respectively), France (29 and 15%, respectively), Ireland (28 and 17%, respectively), Portugal (20 and 8%, respectively) and Greece (15 and 6%, respectively).

European efforts to establish an information society began in 1993, when the Brussel's European Council convened a group of experts to draft the "Bangemann Report," which identified the initial steps toward its implementation. This was followed by the creation of the Information Society Project Office (ISPO, later renamed the Information Society Promotion Office) in 1994, an entity responsible for coordinating the activities of relevant public and private organizations. Subsequently, in 1995, the G7 Ministerial Conference on the Information Society was held in Brussels where eight core principles were identified for fostering wider access and use of ICT throughout Europe. Another major milestone was achieved in 1997 at the Ministerial Conference on "Global Information Networks," where ministers of 29 European countries met to identify strategies for reducing barriers, increasing cooperation, and devising mutually supportive national and international agendas that would advance the development of telecommunications infrastructure and services. Finally, in December 1999, the eEurope Initiative was announced along with a comprehensive plan, "eEurope: An Information Society for All," which delineated specific steps toward the realization of an information society among the member countries of the European Union. The key objectives of eEurope were:

- Bringing every citizen, home, and school and every business and administration into the digital age and online
- Creating a digitally literate Europe, supported by an entrepreneurial culture ready to finance and develop new ideas
- Ensuring the whole process is socially inclusive, builds consumer trust, and strengthens social cohesion

This was accomplished by focusing on 10 priority areas, which included: (1) European youth in the digital age, (2) cheaper Internet access, (3) accelerating e-commerce, (4) fast Internet for researchers and students, (5) smart cards for secure electronic access, (6) risk capital for high-tech small-to-medium sized enterprises (SMEs), (7) eParticipation for the disabled, (8) healthcare online, (9) intelligent transport, and (10) government online.

## F. Latin America

According to Gartner Inc.'s Dataquest, access to telecommunications infrastructure such as telephone lines, and narrowband and broadband Internet connections in Latin America still lags behind that of other developing areas. Their 2000 report "What Will It Take to Bridge the Digital Divide in Latin America?" found that while 80% of United States residents had a telephone connection, Latin America ranged from a high of 24.5% in Chile to a low of 7.9% in Peru. With respect to Internet access, while there were more than 6 million broadband connections in the United States, Latin America included only 4 countries with significant penetration including Brazil (53,000 connections), Argentina (38,000 connections), Chile (22,000 connections) and Mexico (20,000 connections). Similarly, according to "Nua Internet Surveys 2001," while the world's share of Internet users stood at 40% between the United States and Canada, Latin America represented only 4% of this total.

However, despite these statistics it is also clear that Latin America represents a region that is adopting ICT at a fairly aggressive rate. For example, according to the Internet Software Consortium, Latin America experienced the largest increase in Internet hosts in 1999 (136%), compared to North America (74%), Asia (61%), Europe (30%), and Africa (18%). Similarly, Latin American is leading the way with respect to online content. For example, according to Funredes, in 2000, 10.5% of the world population were native English speakers, in addition to 6.3% who were native Spanish speakers, and 3.2% who were native Portuguese speakers. Latin countries made the greatest strides toward increasing the number of Internet sites being offered in their native language with Portuguese and Spanish oriented sites leading the way with 162% and 92% increases between 1998 and 2000, versus a 20% decrease for English oriented sites during the same period.

In addition to establishing access and providing relevant content, there are two related issues that are central to closing the digital divide in Latin America. First is the need to reduce the relatively high rate of illiteracy throughout the region, which prevents entire segments of the population from even accessing information online. This is particularly problematic in countries such as Brazil, El Salvador, Guatemala, Honduras, Nicaragua, and Haiti. Despite recent gains to offer more content in the native languages of these countries, this issue is only exacerbated by the fact that English is still the dominant language in cyberspace. Second is the need to further promote competition in the ICT industry despite successful efforts to privatize nearly all of the major telecommunications firms in each country. According to CEPAL's Division of Production, Productivity and Management, "the primary focus of privatization policies in the telecommunication sector of Latin America might—with the notable exception of Brazil—have not been an increase in competition, but rather to maximize foreign direct investment and establish access to the international financial markets (Argentina) or to defend an important national operator (Mexico)." According to the ITU, in the years following privatization, investment in fixed-line telephone lines actually fell in many countries but by the end of 2000 reached approximately 80 million in Latin America and the Caribbean, up from 60 million in 1998. Similarly, with relatively greater levels of competition in the wireless industry, the trend is now in the direction of a more competitive and dynamic marketplace. For example, in 2000 Pyramid Research estimated the percentage of mobile Internet access at 0%, whereas other forms of access were estimated at 93.4% (dial-up subscription), 2.8% (dial-up free), 0.8% (cable-modem), 0.2% (DSL), 0.9% (ISDN), and 1.9% (leased lines). In 2002, Pyramid Research projects mobile Internet access to increase to 14.7%, whereas the other forms of access are projected to change to 46.5% (dial-up subscription), 34.3% (dial-up free), 2.3% (cable-modem), 2.6% (DSL), 1.4% (ISDN), and 1.3% (leased lines), respectively. In summary, CEPAL characterizes the Latin American telecommunications industry as having low competition in the fixed line segment, medium competition in the mobile segment, and high competition in the Internet segment.

## V. THE DIGITAL DIVIDE DEBATE

Two issues have emerged in the debate surrounding the digital divide since the phrase was first introduced in the mid-1990s. The first issue is related to the framing of the problem. While several studies have measured the digital divide in terms of *access* to ICT, a number of leading research and policy organizations have recommended that the digital divide should instead be measured against the *outcomes* that ICT can be used as a tool to address, such as improvements in the quality-of-life for community members. In 2001, the Morino Institute, a nonprofit organization that explores the opportunities and risks of the Internet and the New Economy to advance social change, released a report, "From Access to Outcomes: Raising the Aspirations for Technology Investments in Low-Income Communities." They wrote,

> To date, most initiatives aimed at closing the digital divide have focused on providing low-income communities with greater access to computers, Internet connections, and other technologies. Yet technology

> is not an end in itself. The real opportunity is to lift our sights beyond the goal of expanding access to technology and focus on applying technology to achieve the outcomes we seek—that is, tangible and meaningful improvements in the standards of living of families that are now struggling to rise from the bottom rungs of our economy.

The argument here is that access to technology alone, without appropriate content and support, as well as a vision of its transformative power, cannot only lead to limited uses, but shortsighted ones as well. This suggests that framing the digital divide in terms of access obscures and oversimplifies its root causes—social and economic inequities—whereas framing the digital divide in terms of outcomes shifts the focus to more appropriate indicators of social and economic equality.

The second issue is related to the use of ICT to improve the lives of individuals, families, and communities. Many have argued that the digital divide is simply a modern day reflection of historical social and economic divides that have plagued society for years. Over the past decade, the community technology movement has gathered momentum toward closing the gap with programs targeted at access, training, content, and more. However, over the past century there has been a parallel effort to revitalize distressed communities often referred to as the community building movement—one that has wrestled with complementary issues in its' efforts to alleviate poverty by instituting programs aimed at education, health care, employment, economic development, and the like.

The community technology movement, primarily in the form of community technology centers (CTCs), and the community building movement, primarily in the form of community-based organizations (CBOs), have historically existed in separate, rather than holistic spheres of practice. In 2001, PolicyLink, a national nonprofit, research, communications, capacity building, and advocacy organization, released a report, "Bridging the Organizational Divide: Toward a Comprehensive Approach to the Digital Divide." In this report they coined this disconnect as the "organizational divide" and wrote, "As we develop policies and programs to bridge the Digital Divide we must ensure that these are linked to broader strategies for social change in two ways. First, we must allow the wisdom and experience of existing community infrastructure to inform our work. Second, we must focus our efforts on emerging technologies as a tool to strengthen and support the community infrastructure." Demonstration projects such as the Camfield Estates-MIT Creating Community Connections Project in Roxbury, Massachusetts, were conducted around this same time to serve as models for how community technology and community building could work in concert. The argument here is that leaders in both of these fields must devise strategies to connect these two movements toward unleashing their collective transformative power. This suggests that from a certain perspective the digital divide should actually be envisioned as a digital opportunity.

## SEE ALSO THE FOLLOWING ARTICLES

Developing Nations • Digital Goods: An Economic Perspective • Economic Impacts of Information Technology • Electronic Commerce • Ethical Issues • Future of Information Systems • Global Information Systems • Globalization • Internet, Overview • National and Regional Economic Impacts of Silicon Valley • People, Information Systems on • Telecommuting

## BIBLIOGRAPHY

Beamish, A. (1999). Approaches to community computing: Bringing technology to low-income groups, in *High technology in low-income communities: Prospects for the positive use of information technology* (D. Schön, B. Sanyal, and W. J. Mitchell, Eds.), 349–368. Cambridge, MA: MIT Press.

Bishop, A P., Tidline, T. J., Shoemaker, S., and Salela, P. (1999). *Public libraries and networked information services in low-income communities.* Urbana-Champaign, IL: Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign.

Cohill, A. M., and Kavanaugh, A. L. (1997). *Community networks: Lessons from Blacksburg, Virginia.* Blacksburg, VA: Artech House Telecommunications Library.

Contractor, N., and Bishop, A. P. (1999). *Reconfiguring community networks: The case of PrairieKNOW.* Urbana-Champaign, IL: Department of Speech Communication, University of Illinois at Urbana-Champaign.

Hooper, P. (1998). *They have their own thoughts: Children's learning of computational ideas from a cultural constructionist perspective,* unpublished Ph.D. dissertation. Cambridge, MA: MIT Media Laboratory.

Morino, M. (1994). *Assessment and evolution of community networking.* Paper presented at Ties That Bind, Apple Computer, Cupertino, CA.

National Telecommunication and Information Administration (1995). *Falling through the net: A survey of the "have nots" in rural and urban America.* Full Report, July. Available at http://www.ntia.doc.gov/ntiahome/digitaldivide/.

National Telecommunication and Information Administration. (1998). *Falling through the net. II. New data on the digital divide.* Full Report, July. Available at http://www.ntia.doc.gov/ntiahome/digitaldivide/.

National Telecommunication and Information Administration. (1999). *Falling through the net III: Defining the digital divide.* Full Report, July. Available at http://www.ntia.doc.gov/ntiahome/digitaldivide.

National Telecommunication and Information Administration. (2000). *Falling through the net. IV. Toward digital inclusion.* Full Report, October. (Available at http://www.ntia.doc.gov/ntiahome/digitaldivide/.

O'Bryant, R. (2001). Establishing neighborhood technology centers in low-income communities: A crossroads for social science and computer information technology. In Townsend, A. *Projections: The MIT Student Journal of Planning—Making places through information technology (2) 2,* 112–127.

Pinkett, R. D. (2000). *Bridging the digital divide: sociocultural constructionism and an asset-based approach to community technology and community building.* Paper presented at the 81st Annual Meeting of the American Educational Research Association (AERA), New Orleans, LA, April 24–28. Available at http://www.media.mit.edu/~rpinkett/papers/aera2000.pdf

Resnick, M., Rusk, N., and Cooke, S. (1998). The computer clubhouse: Technological fluency in the inner city. in *High technology in low-income communities: Prospects for the positive use of information technology* (D. Schön, B. Sanyal, and W. J. Mitchell, Eds.) 263–286. Cambridge, MA: MIT Press.

Schön, D. A., Sanyal, B., and Mitchell, W. J. (Eds.). (1999). *High technology and low-income communities: Prospects for the positive use of advanced information technology.* Cambridge, MA: MIT Press.

Shaw, A. C. (1995). *Social constructionism and the inner city: Designing environments for social development and urban renewal,* unpublished Ph.D. dissertation. Cambridge, MA: MIT Media Laboratory.

Turner, N. E., and Pinkett, R. D. (2000). An asset-based approach to community technology and community building, in *Proceedings of Shaping the Network Society: The Future of the Public Sphere in Cyberspace, Directions and Implications of Advanced Computing Symposium 2000 (DIAC-2000), Seattle, WA, May 20–23.* Available at http://www.media.mit.edu/~rpinkett/papers/diac2000.pdf.

# Digital Goods: An Economic Perspective

**Claudia Loebbecke**
*University of Cologne*

## GLOSSARY

**copyright** One of several legal constructs introduced to ensure that inventors of intellectual property receive compensation for the use of their creations. Copyrights include the right to make and distribute copies; copyright owners have the right to control public display or performance and to protect their work from alteration.

**digital goods** Goods that can be fully expressed in bits so that the complete commercial business cycle can be executed based on an electronic infrastructure such as the Internet.

**on-line delivered content (ODC)** Data, information, and knowledge tradable on the Internet or through other on-line means. Examples include digital online periodicals, magazines, music, education, searchable databases, advice, and expertise. ODC can be offered without a link to physical media. ODC explicitly excludes executable software.

**watermarking** Hiding of data within digital content. A technique for adding data that can be used to identify the owners of various rights, to record permissions granted, and to note which rights may be attached to a particular copy or transmission of a work.

## I. INTRODUCTION: CONCISE SUBJECT DEFINITION

Digital goods are goods that can be fully expressed in bits so that the complete commercial business cycle can be executed based on an electronic infrastructure such as the Internet. This article first positions digital goods at the core of the digital economy. It points to the main economic characteristics of digital goods as well as to criteria for differentiating among different kinds of digital goods. In more detail, the article then covers five specific areas relevant to digital goods: (1) legal and technical protection, (2) pricing, (3) bundling and unbundling, (4) peculiarities of online delivered content, and (5) economics of digital content provision on the Web.

## II. DIGITAL GOODS: CORE OF THE DIGITAL ECONOMY

A major characteristic of the digital economy is its shift to the intangible. Terms with similar connotation include *intangible economy, internet economy, virtual economy,* or *information society.*

The creation and manipulation of dematerialized content has become a major source of economic value affecting many sectors and activities. It profoundly transforms economic relationships and interactions, the way firms and markets are organized and how transactions are carried out. The digital economy is not limited to the Internet. Analog technologies such as radio and TV are also to be considered integral parts of the digital economy because these technologies are getting used to an increasing degree, and further media integration is foreseeable in the near future.

To some extent, the digital economy runs squarely against the conventional logic of economics. Digital goods are not limited by physical constraints and are not limited to traditional economic characteristics, such as "durable," "lumpy," "unique," and "scarce." Instead, digital goods can simultaneously be durable and ephemeral, lumpy and infinitely divisible, unique and ubiquitous, scarce and abundant. The business of purely digital goods is different from conventional electronic business areas, which focus on trading or preparing to trade physical goods or hybrids between physical and digital goods. Trading of digital goods demands new business models and processes.

Classical economic theory does not usually address the issue of digital goods as tradable goods. The value of digital goods, especially information, is traditionally seen as derived exclusively from reducing uncertainty. In the digital economy, however, digital goods—information/content—are simultaneously production assets and goods. This article focuses primarily on digital goods in their capacity as goods to be sold.

From a supplier's perspective, the growing importance of digital goods as intangible assets and the resulting complexity can be seen in the differences between book value and stock market values. These differences can partly be explained by the crucial role attributed to brands, content, publishing rights, and intellectual capital, which may emerge via, be embedded in, or be stimulated by digital goods.

Increasing discussions about information markets—mainly driven by information sciences—are targeting the peculiar case of markets for specialized digital goods such as knowledge or specialized information. Research on information brokers, structuring, retrieval, pricing of on-line databases, etc. represents an important point of reference. Despite all the work during the past 50 years, the problem of classifying, storing, and retrieving digital goods remains a major problem regardless of media type. Multimedia searching by document content is a technology that has reached the initial demonstration phase, but is still in its infancy. There is, essentially, no mature method of storage for retrieval of text, images, and sounds (including speech and music) other than through the use of words, normally in the keyword format. The challenge is to obtain the essential digital content rapidly and attractively. The implied challenge is to modernize the style of presentation to make the key digital goods accessible with as little effort and time as possible. Major efforts are required to establish the right mix of media to convey a particular type of digital good.

## III. DEFINITION, PROPERTIES, AND DIFFERENTIATION CRITERIA

Simply speaking, digital goods are goods that can be expressed in bits and bytes. Table I shows selected kinds of digital goods with some illustrations. Due the variety of terms used, some comments are necessary. A commonly quoted analysis provided by Shapiro and Varian focuses on information that the authors define as everything that is digitalized, i.e. can be shown as a sequence of bits. As an example, for information goods they mention books, magazines, films, music, stock market prices, sporting events, and web sites. Hence, the term *information* as applied by Shapiro and Varian covers basically the same concepts as our term *digital goods* stated above. In addition to such a variety of information goods, we also include software and interactive services such as chat rooms under the term *digital goods.*

In the following, we first take a broader perspective to help us understand and characterize the phenomenon of digital goods, including content and soft-

**Table I** Kinds of Digital Goods

| Kinds of digital goods | Illustrations |
|---|---|
| Searchable databases | ➲ Restaurant guides, phone books |
| Dynamic information | ➲ Financial quotes, news |
| On-line magazines and newspapers | ➲ International, national, regional; general and special interest publications |
| Reports and documents | ➲ Easy multiplication and indexing |
| Multimedia objects | ➲ Music, video files, texts, and photos |
| Information services | ➲ Offerings by travel agencies, ticket agencies, stock brokerages |
| Software | ➲ Off-the-shelf products, customized products |
| Interactive services | ➲ On-line forums, chat rooms, telephone calls, games |

ware. We offer a discussion of special properties of digital goods in this broad sense. After that we offer some criteria for further differentiation among the different kinds of digital goods. We then focus on the equally common, even if narrower concepts, of digital content in general and on-line delivered content more specifically.

In the legal literature, the concept of *digital assets* is more common than that of digital goods. This concept then tends to embrace everything that has value and is available in digital form, or could be valuable if it were changed into its digital form. The term *digital goods* is often used interchangeably with the term *intangible goods*. However, in the literature the term *intangibility* refers to two rather different concepts. Levitt suggests that the terms *goods* and *services* be replaced by *tangibles* and *intangibles* and hence observes that intangible products are highly people intensive in their production and delivery mode. This does not really match with a more recent interpretation of *intangibility,* which is suitable also for digital goods, aiming at immaterial goods (not services), often expressible in bits and bytes.

Digital goods are affected by electronic business from their inception all the way to their use by an end user, facilitating extensive transformations and product innovations. They take advantage of the digitization of the market and of distribution mechanisms. Hence, they are particularly suited for electronic markets. The whole set of business processes can be handled digitally, thus minimizing not only transaction costs, but also order fulfillment cycle time. However, as Shapiro and Varian state ". . . the so-called new economy is still subject to the old laws of economics."

## A. Economic Properties of Digital Goods

Digital goods possess some basic properties that differentiate them from physical goods. Digital goods are *indestructible* or *nonsubtractive,* meaning that they are not subject to wearing out from usage, which can often occur in the case of physical products. They are easily *transmutable;* manipulation of digital products is easier than that of physical products. Last, digital goods are easily and cheaply *reproducible.*

Digital goods are characterized by high fixed costs (first copy costs), dominated by sunk costs, and by low variable and marginal costs. This constellation typically leads to vast economies of scale. In practice, digital goods can be copied at almost no cost and can be transmitted with minimum delay to almost everywhere. This copying of digital products at almost no marginal costs, the ease of transformation in the production process, and the interactivity of the products are reforming the production of digital goods compared to the production economics of physical goods.

Costs for content creation (programming) for high-end multimedia digital titles are often high. As a consequence, companies that create digital goods have an interest in reusing the same content as many times as possible and in as many media as possible without having to pay "first copy" costs again. We speak about a systematic disconnection of production and usage, which naturally has an impact on distribution. It leads to the idea of *windowing:* Pay for content creation once, then reuse it for free. To secure profitability, the producer of digital goods must be able to recoup at least its costs at the first showing of the product.

While a free exchange of information (digital goods) is a crucial prerequisite for innovation, the incentives for innovation and investments are diminished by the difficulties of claiming property rights for digital products. The digitization of content creates a considerable degree of freedom for the provision and the transformation of content.

Consumers are partly involved in the production of information, i.e. the choice of content, mode of display, transformation, etc., and therefore evolve into *prosumers.* For some authors the role of *prosumer* is restricted to the simultaneity of production and consumption, and the nonstorability of services, extended by processes of simple self-service.

The disconnection of production and usage leads to the so-called "value paradox": Only when products are well known and highly in demand are they attributed a high value and the possibility of generating revenues. That is why comparatively unknown providers of digital goods distribute their products to the widest possible public for free (e.g., artists or freeware coders). At the same time, customers are only willing to pay for "scarce" products. Different from physical products, scarcity in digital goods does not come naturally. Instead it has to be reinforced by limited editions and individualization of the copies (e.g., through watermarking) or other restrictive measures.

A frequently applied distinction of products is that made between *search goods* and *experience goods.* This distinction is built on customers' chances to judge the value of a product. The quality of search goods can be determined without actually using them. With experience goods, knowledge about quality is learned from experiencing the product, i.e. from using the good. Search features of a product can be evaluated prior to its usage (e.g., price), but experience features can be evaluated only after usage (e.g., taste). There is also the additional category of "credence goods" where, even after usage, consumers cannot judge the value properly, because they are lacking some necessary skills (e.g., clinical diagnostics). These three terms offer a continuum of judgment, starting from search products—which can be assessed easily—to experience products and finally credence products.

In many cases, digital goods tend to be experience goods or even credence goods. To overcome the implied difficulties for advertising and sales (why should one buy information that one has already experienced or tried?), many digital goods are sold based on strong brands or teasers. For instance, without having experienced an article in a newspaper, the brand of the newspaper leads to the expected sales. Further, teasers such as abstracts or chapters serve as triggers for book and magazine sales.

Digital goods, especially information and content products, are often classified as *public goods.* Public goods share two main characteristics: nonrivalry and nonexclusiveness in usage. Nonrivalry is a product feature normally given in the case of digital goods due to the low costs and ease of reproduction. Nonexclusiveness is a feature of the legal system. In legal systems emphasizing private property, technical and legal means are in place to prevent unwanted joint usage. An automobile is protected by a lock (technical solution) or the threat of police punishment (legal solution) to prevent its use by unauthorized persons. In the digital world, copyrights grant creators of digital products certain rights, which—at least supposedly—can be enforced via technical or legal means. Therefore, digital products cannot be generally termed public goods, even when it is technically difficult to prevent unauthorized persons from using digital products. In addition, the basic laws and constitutions of most countries grant citizens access to "relevant news." So such news in digital form offered, for example, on the Internet could be characterized as public goods. The specific article written about the news, however, could be copyright protected and hence not be a public good.

## B. Criteria Used to Differentiate among Digital Goods

Digital goods represent a variety of economic goods, which require different business processes and economic models. To distinguish within the group of digital goods, we use the following criteria: transfer mode, timeliness, usage frequency, usage mode, external effects, and customizability. With them we are partially following the discussion offered by Choi, Stahl, and Whinston.

Concerning the *transfer mode,* we distinguish between delivered and interactive goods. Delivered goods are transferred to the user as a whole or in pieces, i.e. by daily updates, etc. Interactive goods or services require a synchronous interaction with the user. Examples are remote diagnostics, videoconferences, and interactive computer games. Some careful observation is necessary: Many services on the Internet today are called "interactive," although in reality they are "supply on demand." For instance, when watching "interactive" television, the user merely downloads pieces over time. Neither is a search engine fully interactive, because searches are only orders for personalized delivery. Most digital goods are based on delivery as the transfer mode. Only a realtime application with the need for consecutive questions and answers implies interactivity. Interactive goods are by definition tailored to the specific user, making problems of resale and copyright irrelevant.

The criterion of *timeliness* covers the constancy and dependence of the value of digital goods over time. Products like news, weather forecasts, or stock prices normally lose value as time goes by. The timeliness of any product correlates with the intended usage. For instance, when planning an excursion, weather data are only valuable ahead of time. On the other hand, for scientists studying the accuracy of weather, forecasts deliver value only after the predicted day.

The third criterion is *usage frequency.* Some goods are intended for single use. They lose their customer value after or through use. For instance, the query on a search engine has no recurring value. Other products are designed for multiple uses; examples include software and games. The perceived total value of digital goods designed for multiple uses may well accumulate with the number of uses. One can observe different patterns of marginal utility functions over time. Computer games tend to become boring after a while, leading to negative marginal utility. Software applications on the other hand often render learning effects, leading to increasing marginal utility.

Regarding the *usage mode,* we can distinguish between fixed and executable goods. Fixed documents allow handling and manipulation in different ways and by different means than executable goods. With executable goods such as software, suppliers define the form by which the good can be used. Furthermore, the transformation of fixed documents into executable software increases the possibilities of control by the supplier. For example, suppliers could distinguish among read-only access, sort-and-print access, and a deluxe package that allows the user to make changes to the data pool and to define any possible data queries. Thus, differentiated products to be sold at varying prices can be created out of a common data pool.

Another differentiation criterion within digital goods is the *external effects* associated with products. Products with positive external effects raise the value for customers with increasing numbers of users. For instance, the more participants who agree on a common standard, the more potential partners for exchange exist. In the same way, multi-user Dungeons computer games deliver more opportunities through a larger number of participants. But with restricted capacities, too many participants can cause traffic jams or obstructions, turning positive effects into negative ones. Negative external effects imply a higher value for users resulting from a lower number or restricted number of other participants. This is especially applicable for exclusive information providing competitive advantages, such as internal corporate information used as the basis for speculation on the stock exchange.

*Customizability* reflects the extent to which goods can be customized to specific customer needs. An electronic newspaper has a high degree of customizability in that an average customer is able to design a personal version through combinations of articles. But the articles themselves—being equal for all customers—show low customizability. Consequently, the level of analysis has to be specified (the entire, personalized newspaper or the standard article) in order to be able to judge the customizability of digital goods.

## IV. ISSUES OF LEGAL AND TECHNICAL PROTECTION

Legal questions, usually interpreted in the sense of legal protection of the value of digital products, are of high interest. The development and the application of legal rules have to take into account the properties of digital products and the corresponding technical possibilities and constraints.

Content creators and owners need to protect their property. Traditional content media (such as paper documents, analog recordings, celluloid film, canvas paintings, and marble sculpture) yield degraded content when copied or require expensive and specialized equipment to produce high-quality copies. The technical burden on traditional content creators (such as book authors) for protecting their material has been small. For digital goods, there is no obvious limit to the value that can be added by creating and providing access to digital content. High-quality copies (in fact, identical copies) of digital content are easy to produce.

In this context, legal issues are partially dealt with by application of already existing legal institutions (civic law, criminal law, and international law) and partially covered by rather new legal constructs (concerning contracts or media).

### A. Copyrights

Copyrights are one of several legal constructs relevant to any business producing digital goods. However, they are not new in the world of digital goods. They have been introduced to ensure that inventors of intellectual property receive compensation for the use of their creations. In the international context, copyrights are granted on the basis of the Treaty of Bern, the Treaty of Rome, and the Trade-Related Aspects of Intellectual Property Rights (TRIPS) Agreement. Copyrights have different components. The most notable component is the right to make and distribute copies. In addition, copyright owners have the right to control public display or performance and to protect their work from alteration. Further, content owners hold the rights over derivative works, that is, the creation of modified versions of the original.

As a means to protect intellectual capital, copyrights have gained special importance in the context of digital goods. Because digital goods are easy and cheap to duplicate, copyright protection is essential for ensuring the above-mentioned compensation for product inventors and creators. If creators cannot get paid, what would ensure the continued creation of digital goods? Therefore not only the creators but also digital intermediaries and distributors have high economic incentives to see to it that copyrights are respected and remunerated. However, traditional copyright laws have not been designed for handling digitized goods. Nationally and internationally updated rule sets are under development.

### B. Watermarking

Watermarking represents a technical solution fostering the implementation of the above-mentioned copyrights. Digital watermarks are designed to add value to legitimate users of the protected content and to prevent piracy. In addition, digital watermarks can be utilized for market research.

Following Acken, digital watermarking can be defined as the hiding of data within the digital content.

It provides a technological way to add data that can be used to identify the owners of various rights, to record permissions granted, and to note which rights may be attached to a particular copy or transmission of a work. Digital watermarks are invisible when the content is viewed. They indicate an original, but do not control somebody's access to it. This means that—similar to an original signature—watermarks do not prevent photocopying, which might be needed for fair use.

Digital watermarks can add value for different legitimate uses while increasing barriers to pirates. The benefits depend on the particular digital good and the associated and differentiated needs, burdens, and benefits for their creators, distributors, and recipients. For many business applications, there is great value in being able to reconstruct relevant events. In accounting, the resulting timeline is called an audit trail. For digital content, digital watermarks can be used to indicate recipients or modifiers without the administrative burden of keeping the associated information and links separate from the digital content itself.

Digital watermarking needs to support scalability to be able to match the different value requirements of digital good. Some digital goods, for example, a film classic like *High Noon,* carry high, long-term value. Other digital goods, such as yesterday's stock quotes, have only limited value. The longer lasting the value of digital goods is, the more time pirates have to break the protection methods. Therefore, a scalable system is required that renews itself over time.

## V. SELECTED PRICING ISSUES

Production costs cannot be used as a guideline for pricing because there is no link between input and output. Mass consumption does not require mass production. Economies of scale are determined by consumption, not by production. Economies of scale in digital goods production are limited; economies of scale in digital goods distribution can be significant due to a combination of the high fixed costs of creating the necessary infrastructure and the low variable costs of using it. Economies of scale in distribution are accentuated by consumption characteristics: Consumers tend to use the supplier with the largest variety although they only take advantage of less than 5% of the choices available.

Due to the issues that derive from the abovementioned characteristics of digital goods, neither cost-based pricing nor competition-based pricing are reasonable pricing strategies. Marginal costs are zero or near zero. So by applying cost-based or competition-based pricing mechanisms, sales prices would tend to zero or near zero. But prices near zero make it impossible for producers to get back their high fixed cost. So the only reasonable strategy for pricing information goods is to set the price according to the value the customer places on it. Because consumer valuations are different, it is also important to differentiate prices. Different approaches can be used as the basis for price differentiation. Probably, the two most popular ones are grouping and versioning.

*Grouping* refers to the distinction of prices among different customer groups for the same product. Typical examples from the nondigital world are reduced prices for students or elderly people. The problem for grouping when selling digital goods over the Internet lies in the difficulty of proving people's identity and "characteristics." How do we check, for example, whether a student number from an unknown university is correct, how do we find out where the potential customer is actually located. Technical verification procedures are on the market, but rarely applicable at reasonable effort and cost.

*Versioning* refers to price differentiation based on slightly different product characteristics. Different product versions are sold at different prices. Versioning is already familiar to us from nondigital information goods; consider the pricing of hardcover versus paperback books. For digital goods, Shapiro and Varian suggest numerous ways to create different versions (see Table II).

A consumer's willingness to pay is often influenced by the consumption or nonconsumption of others. Accordingly, it is not an adequate approach for assessing the value of digital goods, given the ease of replication/sharing and associated externalities. Furthermore, the pricing of digital goods raises the fundamental issue of inherent volatility of valuation when the value of digital goods is highly time sensitive. For instance, stock market information may be worth millions in the morning and have little value in the afternoon.

Finally, offering digital goods over an extended period of time may lead to the establishment of *electronic communities.* Electronic communities are likely to create value in five different ways: usage fees, content fees, transactions (commissions), advertising, and synergies with other parts of the business. Translating these income opportunities to the more narrowly defined area of digital goods, *usage fees* could be in the form of fixed subscriptions, paying per page, or paying per time period independent of the quality of the

**Table II** Approaches to Versioning of Digital Goods

| Basis for versions of digital goods | Illustrations |
|---|---|
| Delay | ➲ Books, FedEx |
| User interface | ➲ Search capability |
| Convenience | ➲ More or less restricted time or place of service availability |
| Image resolution | ➲ Higher resolution depending on storage format, etc. |
| Feature and functions | ➲ Quicken vs. Quicken Deluxe, which includes a mortgage calculator. |
| Flexibility of use | ➲ Allowing users to store, duplicate, or print information |
| Speed of operation | ➲ Time to download or to execute programs |
| Capability | ➲ Number of words for dictionary/voice recognition |

content. *Content fees* would most likely be based on fixed amounts per page, but should tackle the issue of valuing the content (quality/relevance). *Commissions and advertising income* are triggered by attractive digital goods on display. Strictly speaking, however, the subsequent income would not stem from the digital goods, but from attracting customers to a page regardless of its content or from offering some empty space for third-party advertising in addition to the actual digital goods offered.

The range of pricing schemes for digital goods is becoming broader and more sophisticated. Pricing models may imply giving actual goods away for free and then charging for complementary services, updates, etc. They are developed for bundles of digital products as well as for single units. Economists are developing theoretical solutions to these problem areas. However, some of the mechanisms developed demand an enormous amount of data, thus questioning the trade-off between allocation efficiency and operational cost-effectiveness.

## VI. UNBUNDLING AND BUNDLING

We see at least two different trends in the digital age: (1) the trend toward unbundling and disintermediation because of the absence of former economies of scale in printing and distribution of content, and (2) the trend toward bundling as a tool to shift consumer rents to the producers.

Traditionally, many digital goods have been bundled solely to save on these costs:

- *Transaction and distribution costs:* the cost of distributing a bundle of goods and administering the related transactions, such as arranging for payment
- *Binding costs:* the cost of binding the component goods together for distribution as a bundle, such as formatting changes necessary to include news stories from wire services in a newspaper bundle
- *Menu costs:* the cost of administering multiple prices. If a mixed bundling strategy is pursued, where the available components are offered in different combinations, then a set of $n$ goods may require as many as $2^n$ prices (one for each subset of one or more goods).

Yet these costs are much lower on the Internet than they used to be for physical goods. Thus software and other types of content may be increasingly disaggregated and metered, as on-demand software applets or as individual news stories and stock quotes. Such a phenomenon is described as *unbundling*.

Unbundling also goes along with the separation of digital goods from the delivery media. Traditionally the pricing of content has been based on the delivery medium—mostly measured in convenience—rather than on actual quality. For instance, the price of a book depends heavily on its printing quality and the number of pages, while the price for an excellent book is almost the same as for a poor one. Electronic trading in digital goods technically allows unbundling. The Internet is precipitating a dramatic reduction in the marginal costs of production and distribution for digital goods, while micropayment technologies are reducing the transaction costs for their commercial exchange. Content can be priced separately from the medium allowing for price differentiation based on the estimated value of the content. Unbundling, however, also raises problems as administration becomes more complex.

On the other hand, the low marginal costs as well as the low transaction costs of digital goods also lead to other ways for the packaging of digital goods through strategies such as site licensing, subscriptions, rentals. These aggregation schemes can be thought of as *bundling* of digital goods along some dimension. For instance, aggregation can take place across products, as when software programs are bundled for sale in a software suite or when access to various content of an on-line service is provided for a fixed fee. Aggregation can also take place across consumers, as with the provision of a site license to multiple users for a fixed fee, or over time, as with subscriptions.

Following Bakos and Brynjolfsson, aggregation or bundling is a powerful strategy to improve profits when marginal production costs are low and consumers are homogeneous because of the changing shape of the demand curve. The economic logic of bundling is based on different consumers' valuations for bundled and unbundled goods.

The larger the number of goods bundled, the greater the typical reduction in the variance. Because uncertainty about consumer valuations hinders effective pricing and efficient transactions, this predictive value of bundling can be valuable. For example, consumer valuations for an on-line sports scoreboard, a news service or a daily horoscope will vary. A monopolist selling these goods separately will typically maximize profits by charging a price for each good that excludes some consumers with low valuations for that good and forgoes significant revenues from some consumers with high valuation. Alternatively, the seller could offer all the information goods as a bundle. Under a very general set of conditions, the law of large numbers guarantees that the distribution of valuations for the bundle has proportionately fewer extreme values. Such a reduction in buyer diversity typically helps sellers extract higher profits from all consumers.

The law of large numbers makes it much easier to predict consumers' valuations for a bundle of goods than their valuations for the individual goods when sold separately. Thus, the bundling strategy takes advantage of the law of large numbers to average out unusually high and low valuations, and can therefore result in a demand curve that is more elastic near the mean valuation of the population and more inelastic away from the mean.

When different market segments of consumers differ systematically in their valuations for goods, simple bundling will no longer be optimal. However, by offering a menu of different bundles aimed at each market segment, bundling makes traditional price discrimination strategies more powerful by reducing the role of unpredictable idiosyncratic components of valuations.

In summary, bundled goods typically have a probability distribution with a lower variance per good compared to the separated goods. Hence, bundling can help to improve seller's profits. One can show that bundling could improve seller's profits when consumer preferences are negatively correlated.

## VII. ON-LINE DELIVERED CONTENT (ODC)

Loebbecke introduces the concept of on-line delivered content as a special kind of digital goods. ODC deserves further attention because the concept includes mainly those forms of digital goods that have gained attention in the Internet age.

### A. Concept, Examples, and Characteristics

On-line delivered content is data, information, and knowledge that can be traded on the Internet or through other on-line means. Examples include digital on-line periodicals, magazines, music, education, searchable databases, advice, and expertise. The decisive characteristic of ODC is its ability to be offered independently of physical media by selling it through a communication network. Whether ODC is then transferred to a computer memory (e.g., with a printout or by burning a CD) or not is irrelevant for the classification of the ODC. Streaming content like a digital video transfer and the transfer of data that can be looked at later off-line are both equally valid ODC forms. ODC focuses on the content of digital products. For that reason, software products including computer games are not covered by the ODC concept. Different from common concepts of digital goods, the term ODC, as defined and applied here, is limited to stand-alone products consisting solely of content/information. Hence, the term ODC implies that *only* the content is the object of a transaction; no physical product is shifted among suppliers, customers, or other players. When trading ODC, the complete commercial cycle—offer, negotiation, order, delivery, payment—is conducted via a network such as the Internet. Figure 1 illustrates this definition of ODC.

The ODC concept can be illustrated by three examples:

1. *Music*. ODC refers to music that can be downloaded from the Web. Afterwards, if desired, it can be stored on a CD-ROM. ODC does *not* include the ordering of a CD-ROM to be

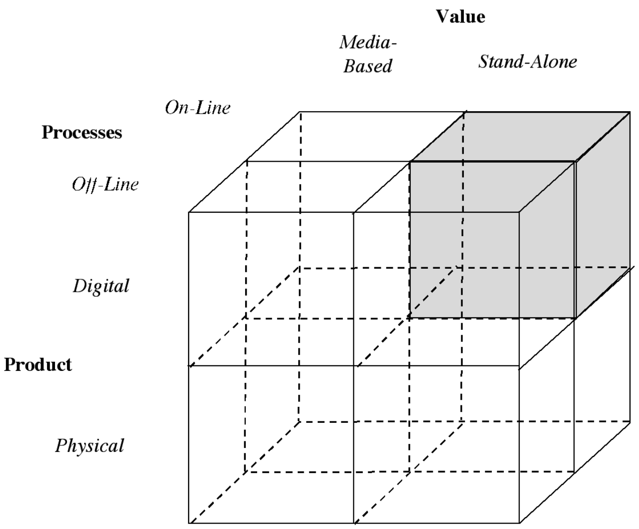


**Figure 1** Conceptualization of ODC. [Adapted from Choi, S., Stahl, D., and Whinston, A. (1997). *The economics of electronic commerce.* Indianapolis, IN: Macmillan Technical Publishing.]

delivered to one's home, since ODC—by definition—refers *only* to the content and excludes the need for any physical medium.

2. *Databases.* Databases are offered by on-line bookstores and various kinds of content are offered on web pages maintained by TV stations. The information/content contained in those web sites is a form of ODC, even if it is usually not traded separately. Possibilities for commercializing such content could be pay per view, pay per page, or pay per time concepts. By trying to sell such content (instead of offering it for free and counting on positive impact on other product lines such as books or TV programs) suppliers would rely on the actual value that potential customers associate with it.
3. *Tickets.* Tickets on planes, trains, or to concerts actually represent a counterexample. Certainly, all paper-based products, like posters, calendars, and all sorts of tickets, could be converted into or replaced by digital counterparts. Further, one can imagine ordering and receiving tickets for trains, planes, or concerts on-line. In the near future, technology will allow individuals to print tickets (administered wherever) just as travel agencies or event agencies do today. However, for consumers this is not the full delivery cycle. They do not pay for the piece of paper called a ticket, they pay for being moved from point A to point B or for attending a concert/stage performance. Those services of "being moved" or "concert performance" are the actual values bought, and they will never be delivered via any technical infrastructure (at least not within the limits of current imagination). Therefore, a ticket, even if bought and—with regard to the piece of paper—delivered over the Web does not represent unbundled, stand-alone value of content. It does not belong to ODC as understood in this article. (For simplicity reasons, this illustration leaves out the possibility of reselling a ticket and thus giving it a monetary function.)

In addition to the issues inherent in trading physical goods on the Web, trading ODC on the Internet raises concerns such as version control, authentication of the product, control over intellectual property rights (IPRs), and the development of profitable intra- and interorganizational business models.

Most forms of ODC belong to the group of experience goods (see above), for which the quality of the content is learned only from using/consuming the good. However, treating ODC as an experience good, i.e. letting potential clients "experience" ODC implies giving the actual content away for free (i.e., not trading it) and, in all likelihood, counting on receiving revenue via some synergy mechanisms. Once potential customers have experienced ODC, they have no more reason to buy it. ODC suppliers will try to solve this dilemma by shifting ODC as much as possible into the category of search goods. Possible steps for this are establishing strong brand reputation for Web sites or publishers or offering abstracts, sample chapters, or reviews as triggers to buy the whole product.

As a consequence of the characteristics of digital goods such as indestructibility, transmutability, and reproducibility, the exclusivity of ODC may be difficult to durably maintain. Sharing may be simultaneous or sequential; in any case it affects the allocation of property rights. While a seller of a physical good loses his or her property right, a seller of ODC may continue to hold it. Even illegally sharing ODC often causes positive network externalities, which may even exceed the cost of sharing if caught. Once ODC has positive network externalities, control over reproduction and sharing is the primary objective of copyright protection.

Related to the issue of externalities is the issue of value generation. Often there is no direct link between a transaction and the generation of value. Furthermore, ODC value can hardly be measured in monetary terms only. For instance, the appreciation of free TV could be measured in time budgets allocated; and appreciation of academic papers (increasingly often provided as ODC) may be measured in number of quotes. Indirect value creation and the related problem of ODC value measurement lead to the problem of adequately pricing ODC, as discussed later in this chapter.

While the conventional logic of economics is concerned with scarcity, the dematerialization logic inherent in ODC is concerned with abundance. Abundance and resulting ODC overload (the huge variety of ODC available to almost everybody) confront consumers with a dilemma. They want to take advantage of the increased choice of ODC, and at the same time, they seek to minimize the costs of searching. To respond to the first objective, new modes of consumption have emerged: zapping, browsing, or surfing. These are characterized by short attention span, latency, high frequency of switching, and capriciousness. The distinction between consumption and nonconsumption becomes difficult, rendering pricing problems even more intractable. The expanded choice of content makes consumer choice more difficult, thus continuously raising the cost of acquiring information about the content. To minimize these costs, the choice is increasingly determined by criteria other than product characteristics, e.g. brand familiarity or fashion. Low transaction costs lead to excessive volume of transactions that generate noise rather than useful content. Abundance of products and services stimulates the development of activities whose purpose is to monitor, evaluate, and explain their characteristics and performance.

## B. Trading in ODC

While the offering of free ODC has become extremely popular in the Internet area, only a few companies have started trading. To trade in ODC, several roles have to be fulfilled. The value chain depicted in Fig. 2 has been outlined by the European Commission for the electronic publishing business. It differentiates between two layers. The content-related layer addresses content creation, content packaging, and market making. The infrastructure-related layer comprises transportation, delivery support, and end-user interfaces.

The framework suggests the following strategic roles to be played (Fig. 3). Online Networks manage a full electronic marketplace, Community Organizers focus on interest-centered target groups, Interactive Studios create content with new levels of functionality, Content Rights Agencies manage rights and match content to market needs, and, finally, Platform Providers create end-to-end, easy-to-use technical platforms for authors, publishers, and end users. Rather recent concepts suggest that such activities be organized as value networks instead of value chains. The strategic roles to be fulfilled do not significantly change, regardless of conceptualization in a chain or in a web.

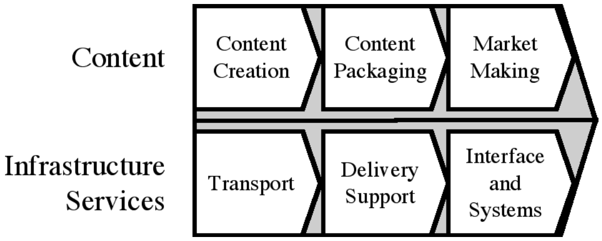


**Figure 2** Electronic publishing value chain. [From European Commission (1996). *Electronic publishing—Strategic developments for the European publishing industry towards the year 2000.* Brussels.]

Syndication is also of particular interest as a business model in the context of ODC trading. Syndication involves the sale of the same good to many intermediaries, who then integrate the good with others and redistribute the whole. First, syndication can only work with information goods since they can be duplicated and consumed by infinite numbers of people without becoming exhausted. Second, syndication requires stand-alone, modular products that may function well as a part of a whole. Third, syndication requires multiple points of distribution. The millions of existing web sites theoretically offer many different points of distribution.

In such an environment, trading in ODC can be used to supply innovative content, especially differently packaged, more targeted information. It combines communication with content, leading to higher quality and thus added value to customers. Further-

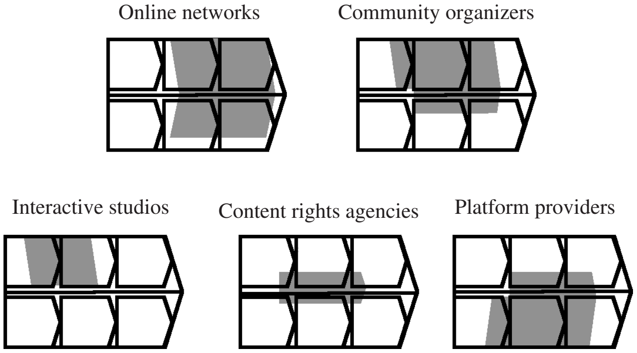


**Figure 3** Strategic roles in electronic publishing. [From European Commission (1996). *Electronic publishing—Strategic developments for the European publishing industry towards the year 2000.* Brussels.]

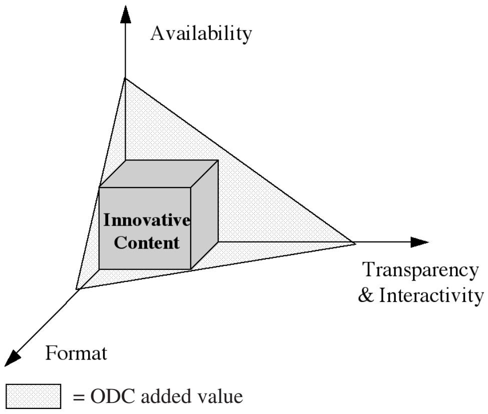


**Figure 4** Dimensions of ODC added value.

more, ODC customers are more in control of how much and what kind of content they want to obtain. When substituting print products by ODC, customers will request additional value such as availability (newest information, access to data from any location), format (multimedia such as video clips and sound), transparency and interactivity (user-friendly downloading, search functions, etc.), and innovative content (Fig. 4).

In summary, ODC refers to digital goods that are manufactured, delivered, supported, and consumed via the Internet or similar networks. Typical examples of OCD are music, information, and expert knowledge. For these types of goods, as for almost all kinds of digital goods, traditional economic models based on scarcity and uniqueness leading to a market based on demand and supply do not apply. Once created, ODC is extremely easy and cheap to replicate, distribution costs are almost zero, and most other transaction costs except perhaps marketing and sales barely exist.

## VIII. ECONOMICS OF DIGITAL CONTENT PROVISION ON THE WEB

We distinguish four possibilities for profiting from providing digital content on the Web: (1) increasing the number of units sold, (2) increasing the margin per unit sold, (3) selling digital content as stand-alone product, and (4) generating advertising income from web pages. In the first two cases, the digital good is a free enhancement of the main, nondigitizable product offered (cars, coffee, computers), which cannot be delivered via the Internet. In the third setting, the product offered consists of information and thus can be transmitted digitally via the Internet (magazines, music, etc.). For such a good, the term on-line delivered content was introduced in the previous section. In the fourth setting, the focus is not on the actual goods, but on the space for sale around the content on the Web.

1. *Increased number of units sold.* Internet-based marketing and public relations aim at increasing awareness about a company and its product and service range. As with traditional marketing, this is costless for consumers; profit is made when the marketing costs are compensated by additional sales. Currently the largest potential in Internet-based marketing is seen in attracting new customers worldwide and in establishing distant, long-term customer relationships. In most instances it is difficult to discover how many additional units are sold because of a web presence. Further, some of these may be substitutes for traditional sales (internal channel cannibalization).

   As long as overall worldwide or regional sales do not increase, but almost every bookstore, computer dealer, etc. is present on the Web (with rather different offers), it is not obvious how they all could increase their total turnover. It seems to be more like a football league: Every team gathers strength during the summer but by the end of the following season there are few winners, and there will always be some losers.

   There is no doubt, however, that Internet-based turnover is predicted to grow during the next few years. But with more efficient business processes and price transparency leading to decreasing margins there is not too much reason to foresee an increase in total (traditional and Internet-based) turnover and especially profits.
2. *Increased margin per unit sold.* Larger margins per unit could theoretically be achieved by lower costs (efficiency) or by charging higher prices per unit. Lower costs may be achieved by using the Web for various processes such as internal communications, receiving orders and payments, or providing customer service (process/business reengineering). Customers could, for instance, download information from the company's web site and special requests could be answered via (automatic) e-mail. From a more in-depth perspective, most efficiency gains will result from decreased working capital achieved by introducing electronic commerce, e.g. Internet-based activities. Higher prices charged per unit need to be based on value added for customers. This means that a particular book, computer, or type of coffee that is advertised and sold via the Internet is more expensive than if it were sold via

traditional marketing media and sales channels. This notion is the reverse of the more popular idea of selling cheaper via the Internet due to economies of scale, improved transparency, and fewer players in the value chain. If, however, the Internet sale of a digital good provides no added value, then competition may well squeeze prices down to the level of the marginal cost of the goods.

3. *Digital content sold as stand-alone product.* This is the ODC situation, which was discussed at length in the previous section.
4. *Advertising income generated from web pages.* The market for advertising space on the web is booming. Only those companies whose contents attract a certain number of site visitors can sell additional space to others who then place their ads. While this opportunity for profit is gaining importance, it is mainly suitable for those large companies whose sites are well known and visited, e.g. TV stations, newspapers, and magazines, etc. It does not appear to be a feasible source of income for the millions of small and medium sized enterprises (SMEs) that also offer content on the Web.

Large company infrastructures to market specific products are no longer required either for content provision on the Web or for the actual sale of digital goods. This causes an enormous growth of digital products and service offerings. However, small content providers still mainly count on positive, but indirect contributions of their Internet activities to their overall cost–benefit structure. For SMEs to continuously provide digital content on the Web, shifts in financial flows along intercorporate value chains are required. Table III outlines two scenarios regarding potential sources of income for digital content providers and the related shifts in intercorporate value chains. To clarify the terminology of Table III, Internet providers "transport" content from content providers to customers. They are comparable to common carriers expecting payment for this intermediary service. If they manage to enhance their service line beyond transmission, e.g., with value-added services, this should allow them to charge consumers for more than just the transmission fee.

*Scenario 1:* Digital content providers receive payment for their content directly from the consumers who not only have to pay the Internet providers but also the content providers for the information they access. Competition for customers among content providers would begin to develop; hence, the quality of information is likely to improve. The situation for Internet providers would mostly stay the same, unless—due to the higher Internet consumption price for users—the overall Internet traffic would decrease drastically.

*Scenario 2:* Digital content providers receive payment from Internet providers who forward part of their income to the content providers. Internet providers can only win in this scenario if the low price of content and service in comparison to the previous scenario would lead to a drastic increase in overall Internet traffic. The situation for consumers would remain mainly the same.

In summary, electronic media enable organizations to deliver products and services more cost-effectively and efficiently. In cases where the Internet is supposed to support the traditional business (e.g., book sales), the increasingly sophisticated services offered go beyond pure marketing efforts. They provide additional value to customers. While these services constitute extra costs, they barely generate additional profits. Potential clients take advantage of these services (e.g., search the bookstore database) without necessarily becoming customers. Involvement in Web-based activi-

**Table III** Shifts in Financial Flows along Intercorporate Value Chains

| | Content provider | Internet provider | Consumer |
|---|---|---|---|
| Currently | Receives no payment for content provided | Receives payment on time/volume basis | Content mostly free, pays for time and volume |
| Scenario 1 | Receives payment based on content directly from the consumer | Receives payment on time/volume basis | Pays for content, time, and volume |
| Scenario 2 | Receives a predefined share from the Internet provider | Receives payment on time/volume/content basis and shares with content provider | Pays for time/volume |

ties and increasingly also content provision on the Web seems to have become compulsory in many industry sectors. If eventually all companies achieve significantly lower cost for customized product and service delivery, the result cannot be a competitive advantage, but lower margins for the average player in the sector. Offering content on the Web has to be attractive for the providers in one of two ways: (1) strengthening a company's competitive position with respect to its traditional products (e.g., higher turnover as a consequence of Web activities, or (2) expanding toward additional, profitable product lines (e.g., selling information/content-based products and services).

## SEE ALSO THE FOLLOWING ARTICLES

Advertising and Marketing in Electronic Commerce • Business-to-Business Electronic Commerce • Copyright Laws • Desktop Publishing • Digital Divide, The • Economic Impacts of Information Technology • Electronic Commerce, Infrastructure for • Electronic Data Interchange • Marketing

## BIBLIOGRAPHY

Acken, J. M. (1998). How watermarking adds value to digital content. *Communications of the ACM,* Vol. 41, No. 7, 75–77.

Bakos, Y. (1998). The emerging role of electronic marketplaces on the Internet. *Communications of the ACM,* Vol. 41, No. 8, 35–42.

Bakos, Y., and Brynjolfsson, E. (1999). Bundling information goods: Pricing, profits and efficiency. *Management Science,* Vol. 45, No. 12, 1613–1630.

Choi, S., Stahl, D., and Whinston, A. (1997). *The economics of electronic commerce.* Indianapolis, In: Macmillan Technical Publishing.

Evans, P. B., and Wurster, T. S. (Sep.–Oct. 1997). Strategy and the new economics of information. *Harvard Business Review,* 71–82.

European Commission (1996). *Electronic publishing—Strategic developments for the European publishing industry towards the year 2000.* Brussels: ECSC.

Loebbecke, C. (1998). Content provision on the Web—An economic challenge for TV stations. *Australian Journal of Information Systems,* Vol. 6, Special Edition 1998—Electronic Commerce, 97–106.

Loebbecke, C. (1999). Electronic trading in on-line delivered content. *Proc. Thirty-Second Hawaii International Conference on System Sciences (HICSS-32),* A. Dennis and D. R. King (Eds.).

Shapiro, C., and Varian, H. R. (1998). Versioning: The smart way to sell information. *Harvard Business Review,* 76(6), 106–114.

Shapiro, C., and Varian, H. R. (1999). *Information rules: A strategic guide to the network economy.* Boston: Harvard Business School Press.

Wang, R. Y. *et al.* (Summer 1998). Manage your information as a product. *Sloan Management Review,* 95–105.

Werbach, K. (May–June 2000). Syndication: The emerging model for business in the Internet era. *Harvard Business Review,* 78(3), 84–93.

# Disaster Recovery Planning

**Ata Nahouraii**
*Indiana University of Pennsylvania*

**Trevor H. Jones**
*Duquesne University*

**Donald Robbins**
*Indiana University of Pennsylvania*

## GLOSSARY

**access time** Time and transfer time.

**accounting** Organization and the procedures that are concerned with asset safeguarding, and the reliability of financial records.

**address modification** The process of changing the address part of a machine instruction.

**auditing** The examination of information by a third party other than the user in order to establish substantive and compliance tests. The tests may be conducted internally by internal auditors, or externally by an external auditor.

**audit software** A collection of programs and routines associated with a computer that facilitates the evaluation of machine-readable records for audit purposes.

**audit trail** A means for systematically tracing the processing of a machine-generated output with the original source (input) to verify the accuracy of the processes.

**auxiliary storage** A supplemental part of a computer's permanent storage.

**backup** Pertaining to equipment or application programs that are available for use in the event of failure. This provision now is an important factor in the design of every information processing system, especially in the design of real-time systems where a system failure may bring the total operations of a business to a possible standstill.

**block diagram** A diagram of a system, computer, or program represented by annotated boxes and interconnecting lines to show relationship. It should be noted, a flowchart is a special type of flow diagram that shows the structure or sequence of operations in a program or a process.

**computer audit program** A computer program written for a specific audit purpose or procedure.

**data** Raw facts or an entity without any meaning.

**disk storage** A type of magnetic storage that uses one or more rotating flat concentric plates with a magnetic surface on which data can be recorded by magnetization.

**documentation** The collecting, organizing, storing, and dissemination of documents or the information recorded in documents.

**systems analysis** The examination of an activity, procedure, method, or technique in order to determine what must be specified, designed, developed, and how the necessary operations may best be accomplished.

**A DISASTER RECOVERY AND CONTINGENCY PLAN** ensures a business' survival when it is faced with a technological and/or information systems breakdown. Its primary objective is to prevent a calamity from occurring and to limit the impact of destructive events related to computer-based information systems. Most disaster recovery plans when put into effect, fail to serve the intended purpose. The value of a properly implemented disaster recovery approach is that prevention seeks to stop incidents before they can occur; and recovery restores services so that assets would not be adversely affected. This manuscript attempts to provide some insights as to the environment in which these occurrences can happen, what resources should be considered to achieve the goals of asset safeguarding and maintaining successful information flow within the organization, as well as alternatives for recovery should front end security fail and information loss is imminent. Additionally, innovative technological trends for information security and asset safeguards will be introduced.

## I. INTRODUCTION

It is important to recall significant events that have taken place which may, and have, affected the flow of organizational information. Increasingly, terrorist incidents and Internet abuses over the past years have become more prominent and affect what approaches are needed for implementing emergency response plans (Table I). Examples of some of the most recent high profile situations are:

- The bombing of the USS Cole and the sinking of the Russian submarine Jursk
- The bombing of the Oklahoma Federal Building and World Trade Center (1993)
- The San Francisco earthquakes and Chicago floods
- On September 11, 2001, three hijacked jetliners—American Airlines Flight 11, United Airlines Flight 175, and American Airlines Flight 77—were crashed into the north and south towers of the World Trade Center, and the Pentagon, respectively. The people of the United States stood in shock and horror as this terrorist action took place in less than an hour's time, killing thousands of people and leaving countless injured. The World Trade Center housed many Wall Street technology-based companies, banking firms, law offices, and international trading firms.

When we investigate examples such as these we begin to realize the significant economic impact they have.

Heavy winter storms, in 1994, caused a complete collapse of the roof of the MAC card data center in Northern New Jersey. Thousands of MAC transactions and an unknown, substantial amount of revenue were lost. World Trade Center systems could not be accommodated because the banks' alternate processing facilities were already committed.

Recent crises in the state of California concerning power generation have had negative impacts on business. The *Wall Street Journal* reported in its January 26, 2001, issue that Phelps Dodge, the second largest copper mining company, has had to close its operation which affected 2350 employees.

Intrusion via software or user access also must be considered as potential threats. Examples of security breaches were brought to light by Hoffer and Straub in 1989 and also by Hein and Erickson in 1991. One such example includes Ronald Cojoe's attempt to use his high-level system access and knowledge to defraud the city of Detroit for $123,000. A more humorous example is a malicious employee prank at the *Calgary Herald*. The prank consisted of placing obscene messages in files that seemed to randomly appear on the computer screens of the newspaper's reporters, editors, and management information systems staff. The Hamburg Chaos Computer Club, which intruded into a variety of United States and European corporate and government computers in 1987 is another example of security breakdown. This incident, along with the observations of Allen in 1977, Steinauer in 1986, Hootman in 1989, Vacca in 1994, and Nahouraii in 1995 showed the weaknesses present in the security design of networks. Later it became a prelude to a WORM attack by Robert T. Morris, a first year computer science graduate student at Cornell who created a code that could disrupt any UNIX operating systems activity, making it inoperative. A WORM is software that propagates itself across a network and causes resources of one network to attack another. On November 2, 1988, this type of code was unleashed over the Internet. The devastation resulted in disruption of the operations of 6000 computers nationwide. Since 1988 at least 11 well-identified intrusions causing deletion of files, distributed denial of service (DDoS), or erasing hard drives have struck governmental and corporations around the world. These viruses, code named Michelanglo (1991), Word Concept (1995), Wazzu (1996), Melissa (1999), Chernobyl (1999), Explore.Zip (1999), BubbleBoy (1999), The Love Bug (2000), Killer Resume (2000), Zombies (1/2001), and most recently Anna (2/2001), generally appear in the form of file attachments. DDoS assault was launched against Network Associates' DNS server. The attack lasted for nearly 90 minutes and users had difficulty connecting to Network Associates' web servers. A similar attack was also launched on Microsoft's DNS records the week before the realization that possible weaknesses existed in the Berkeley Internet Name Domain (BIND), the most widely used implementation of DNS according to P. J Connolly (Pj_conolly@infoworld.com), a senior analyst in the InfoWorld Test Center. However dramatic we might consider these circumstances to be, we should keep these occurrences in perspective. According to David Ballard, Product Marketing Manager for Veritas Software Corp., the number

**Table I** Types of Intrusions

| Physical | | Technical (viruses) | Environmental | Fraud/ embezzlement |
|---|---|---|---|---|
| Passive | Active | | | |
| Foreign/domestic collection to gain competitive edge in the global/military arena<br>Typically technical in nature activity<br>Interests as identified by US defense report includes:<br>Radiation transfer tech.<br>Equipment testing and diagnostic software<br>Infrared signature measurement software [IR]<br>Penetration method:<br>Targeting international conferences<br>Exploiting joint research/ventures<br>Co-opting of former employees [GM vs. VW]. | Terroristic nature to gain political/cultural/global attention<br>Environmental causes<br>Fraud/Embezzlement<br>Historical incidents physical<br>Penetration method:<br>1993: New York's World Trade Center<br>1994: Israeli Embassy in London<br>1995: Japan's Subway<br>1995: Federal Building in Oklahoma City<br>1996: Several bombings in Tel Aviv and Jerusalem by HAMAS<br>1997: Killings at Hatsheput Temple in Egypt<br>2000: USS Cole bombing.<br>2001: September 11th World Trade Center and Pentagon attacks. | 1987: Chaos Computer Club<br>1987: IBM Christmas Tree Exec<br>1988: Kevin Mitnick attack on Digital Equipment<br>1989: "No Nukes Worm"<br>1990: Hide and C-eek Trojan Horse<br>1991: SYSMAN.EXE Trojan Horse<br>1995: World Concept<br>1996: Wazzu<br>1999: Melissa<br>1999: Chernobyl<br>2000: Love Bug<br>2000: Killer Resume<br>Dec. '00: Kris Virus | 1989: Hurricane Hugo<br>1989: San Francisco Earthquake<br>1992: Chicago Flood<br>1992: Andrew Hurricane<br>1994: Iowa Flood<br>2001: Ahmed Abad Earthquake (India) | 1991: Charles Keating defrauds U.S. Fed. Home Loan Banks<br>1992: Looting of United Fund by William Aramony, its CEO<br>1994: Calif. Orange County Bankruptcy<br>1995: The Fall of Bank of England<br>1995: The New Era Philanthropy fraud by John G. Bennet<br>1995: Clark Candy, CEO, Michael Carlow, defrauding PNC Bank<br>1999: Informix-accounting irregularities—Xerox; Luccent |

one reason for loss of data is due to accidental deletion, corruption, or other software errors.

There are often debates about the design requirements, specifications, and the language for the development of an application program interface as an ancillary part of the disaster recovery tool. McEnrue and Bourne studied various models of human performance and reaction including psychological and cognitive processes. In their study, subjects were introduced to threatening situations posed by software that led to computer site shutdowns in order to impede data loss. In 1962, Hunt modified the previous model by including synchronous systems and analyzed how humans perceive security and take measures for safeguarding sensitive files with or without automated information systems. Similarly, in the 156 cases of computer fraud that Allen studied, he found that 108 of these cases involved addition, deletion, or modification with input transactions. It should be noted that no system is sufficiently secure to stop all breaches of security. This is especially true in cases that involve fire or flood incidents. As reported by Ron Weber as well as Wayne and Turney, fire and water are often the major causes for computer outages. Because of these possibilities, a considerable amount of effort has been devoted by information scientists, EDP auditors, psychologists, and law enforcement analysts, to formulate a better description for the design and application of disaster recovery and computer security.

With this in mind, designers and users have proposed various control objectives and techniques needed for emergency response. These include a variety of operational procedures for disaster recovery using recent emerging technologies such as biometrics and authentication, the use of hot sites, cold sites, firewalls, encryption, and steganography, that can be installed on computers of differing sizes such as IBM, Sun Stations, Microsoft Windows, Linux, or Unix based operating systems.

In general, disaster expectations can be categorized into four categories:

1. **Physical.** These include terroristic actions, and input output devices failure, e.g., disk head crashes or systems failure as a result of sudden power surge or power interruptions caused by lightening.
2. **Technical.** Virus intrusions are categorized here as well as the detection of operating system or application software faults (bugs).
3. **Environmental.** This may be the most noticeable and newsworthy, but least considered. Examples are flood, fire, and earthquake as well as more insidious items such as a misplaced water main which may freeze and burst causing subsequent water damage.
4. **Fraud/embezzlement.** This category includes deliberate and unauthorized data access, use, and manipulation.

## II. BACKGROUND

In the world of technology, in order for a company to remain viable, top management must establish a policy to safeguard integration of the servers, software, and storage systems. If the top management's system of controls are not reliable, it is unlikely that the viability of the enterprise can be assured. The intent of the controls is to strategically prevent fraud as well as natural disaster that may occur. These controls are generally assimilated into environmental controls by management. Thus, environmental controls require a well-planned, well-executed procedure that assures successful communication, commerce, competitiveness, and growth both internally and among its external business so that the firm is always universally functional. The enterprise must also safeguard against the possibility of fraud execution or sabotage by implementing internal controls that effectively fulfill the auditing compliance standards.

1. Proper segregation of duties
2. Control over installations and changes
3. Control over system librarian and utilities
4. Password control
5. Control over programmable read only [Prom] and erasable programmable read memory [EPROM] programs
6. Control over Utility scan

These controls as suggested by Weber must be periodically reviewed for modification by management so that it can detect easily the system security aberrations caused by environmental hazards, I/O devices, human error, or by computer abuse. Environmental hazards include fires, floods, tornadoes, earthquakes, facility management systems of nuclear power plants, and other natural causes. I/O errors are input or output devices. They include damage to disk packs by faulty disk drives, reader's error, off-line key to disk errors or errors in application programs that destroy or damage data, and mounting of incorrect files by the operational staff. Computer abuse is the violation of a computer system to perform malicious damage, crime, or invasion of privacy. Therefore, management should be alert that malicious damage, including deliberate sabotage, or intentionally attempting to defraud profit, or loss reporting as a means for tax evasion, or concealing performance to shareholders of the commercially traded security stocks, will be visible during internal audit.

Similarly, crimes that include embezzlement, industrial espionage, the sale of commercial secrets or invasion of privacy by access to confidential salary information, and the review of sensitive data by a competing company should be easily detected by the internal controls of top management. The frequency of occurrence of computer abuse is difficult to determine, but the cost per incident reported is considered to be phenomenal.

To put it simply, if environmental controls are absent, so goes the enterprise. That is, in today's economy, if the enterprise fails to connect with its external suppliers and customers as a result of network crashes, or when the databases fail without any asset safeguarding plan in place, a serious and unpredictable problem ranging from embezzlement to system failures will result. Thus, internal controls, when placed by management, should be prepared before a disaster strikes.

## III. STANDARDS

The Foreign Corrupt Practices Act holds the corporate executives accountable for failing to plan adequately for a disaster. Fines of up to $10,000 and 5 years imprisonment may be levied for such a failure. A similar legislation called the Computer Security Act was approved by the United States Senate in 1987. This legislation gives the United States National Bureau of Standards a role in setting computer security measures in the civilian sector as shown by Brandt in 1977. This act mandated that by l992 a computer system or network must be certifiable as secure in four areas as was previously mentioned by Mitch in 1988 and Miller in 1991:

- User identification
- Authentication
- Access control and file security
- Transmission security and management security

In response to these federal requirements, local area network (LAN) vendors are developing software that will support the foregoing requirements. In addition, banking laws, the federal system requirements, the Securities and Exchange Commission along with the Internal Revenue Service (IRS) all have regulations dealing with the responsibility for controlling exposures relating to disasters. For example, the Federal Reserve Board requires that when companies use an Electronic Fund Transfer (EFT) system they must have a recovery plan readily available for recovery within 24 hours. This is discussed in the writings of Wong, Monaco, and Sellaro in 1994. The Auditing Standards Board of American Institute of Public Accountant (AICPA) issued the Statement of Auditing Standards (SAS) No. 82, "Consideration of Fraud in Financial Statement Audits." By increasing the probability of uncovering or detecting fraud, the AICPA intends this standard to increase the integrity and reliability of the financial statements.

The management security control generally is interpreted as an environmental control by auditing firms. These controls as suggested by Porter and Perry in 1992 must have in place a set of objectives. These objectives must:

1. *Establish security objectives.* These objectives contain standards against which actual system security can be judged.
2. *Evaluate security risks.* Management should evaluate file maintenance and recovery security risks for likelihood and cost of occurrence. For example, flood or earthquake will have a low probability of occurrence and a high cost per occurrence, whereas damage to I/O devices because of human negligence will occur more frequently but at a low cost per incident. Internal controls should estimate probabilities and cost associated with each possible security failure so that the expected values of loss when computed may be used as guides to the effectiveness of the security controls.
3. *Develop a security plan.* Management should consider a plan that will be cost effective at an acceptable level, should describe all controls, and identify the purpose of their inclusion.
4. *Assign responsibilities.* Management should assign accountability and assign responsibilities that include implementation of the plan and monitoring of the controls on an ongoing basis.
5. *Test system security.* The controls should be tested by management to make sure the staff is properly trained and that they are aware of the consequences of possible disaster. This process will determine where weaknesses exist in the security plan and assure that the existing plan provides recovery from security failures as intended.

6. *Test the physical isolation of the computer facilities.* Isolation can be accomplished by having a separate building for the computer center or having a secure location in the enterprise.
7. *Test the use of construction standards.* Security risks can be reduced by adequate construction standards. The walls and doors of the computer facilities should be in compliance with American Society of Mechanical Engineers (ASME) earthquake structural specifications and standards. Windows should be avoided completely. Data files and documentation should be stored in safes and vaults, power and communication lines should be protected with a secondary power company supplier or a stand alone power generator.
8. *Test the compliance with fire and water standards.* The risk of environmental damage can be reduced by following construction standards for fire which includes the use of fire-resistant walls, floors, ceilings, and doors. Fire extinguishment systems such as halon gas should be used to extinguish fires and minimize damage to data processing equipment and personnel. Water standards include the use of pumps and drains to minimize water damage from sprinklers or floods. Watertight floors in the computer room will help keep out water from floods or from other parts of the building.
9. *Evaluate system security.* Regularly plan a pilot run to test the recovery plan and the results of testing should be used to evaluate the effectiveness of controls in meeting disaster recovery objectives.
10. *Testing the installation of disaster recovery software.* This test is usually made to make sure the software transparency exists as claimed by its vendor. The test ensures that the data center operating system platform can be easily revised if needed.

In summary, management must include in their internal control policies and procedures needed in planning, building, and maintaining assets safeguarding methods to guard against disaster. This is necessary not only for fraud prevention but also for the impact on the environment. According to Wayne and Turney (1984), management must be certain that they have evaluated the likelihood of failure and have adopted procedures to expeditiously return the organization to normal infrastructure functioning while minimizing data and asset loss, should a catastrophe occur. Management must also ensure its computer security to protect hardware and data against unauthorized access, destructive programs, sabotage, environmental changes, and fraudulent use. The corporations thus need to become increasingly vigilant about these issues due to:

- An increase in installed micro-based computers in their workplaces, currently being augmented and superseded by personal digital assistants (PDA) or other hand-held devices
- The use of automated systems to attain competitive business advantages
- The sophistication of managers and employees in computing technologies
- The increased volumes of sensitive data being kept on various secondary storage devices which are accessible through both internal and external sources
- Greater interconnectivity of computers through networks
- Increased investments in hardware, software, and networking technologies
- The high cost of recovery as a result of data loss
- The sophistication of potential intruders and digital viruses
- The arsenal of mobile computing tools
- The increasing cost effectiveness of e-commerce and the consequential reliance on digital technologies to support these models
- Crowded skies and increased digital information movement created by the economy for consumers

Furthermore, management should continuously address questions such as: What platforms should we use? How do we plan for the unexpected? Will it grow when we grow? Will it work with new technology in the future? Will it build upon our current systems? Can we link to our customers' and suppliers' systems? Can open standards be used? What about outsourcing? How do we finance all of this?

## IV. EMERGING TECHNOLOGIES

The use of access control can be traced back to around l000 B.C. when the Chinese developed a control system to guard their imperial palaces. Each member of the palace staff wore a ring engraved with intricate designs that identified which areas of the palace they were allowed to enter. Today, access controls range from simple locks and keys to sophisticated systems using biometrics' physiological or physical techniques. The private sector as well as governmental agencies have assimilated the use of these techniques for their security and recovery plans to safeguard their information systems and data centers.

Biometrics identification systems are machines that verify the identity of a person based on the examination and assessment of unique personal physiological features (Fig. 1). Characteristics such as signatures, retinal blood vessel patterns, fingerprints, hand geometry, face prints, and speech are ideal as a basis for an identification system because, unlike keys, they cannot be lost or stolen. Biometry refers to the application of statistical methods to describe and analyze data concerning the variation of biological characteristics obtained by either observation or experiment. Biometrics identification systems require that the characteristics of people who will be using the system be gathered in advance. The importance of this was discussed in the writings of Zalud (1989) and Rosen (1990).

When making identification, biometrics systems read in user's characteristics and convert them into a form in which they can be compared to a set of reference samples. Computer algorithms are used to make the comparisons and verify identity. The success of a biometrics method is based upon the characteristic selected for comparison and the nature of the allowances made for variations.

For signature verification, a person will create a reference sample by signing his or her name about six times with a special pen whose movement is recorded by a computer. This information is stored as a reference sample, which is often referred to as a template. The computer then evaluates the variations in the signature with the stored templates. If the matching pressure and movements of the person signing are correctly matched, access is granted. Most of the characteristics that the systems work with can vary over time. Hands can swell from work, heat or allergies, fingerprints can be marred by scratches or embedded dirt, and voices can vary from colds. For these reasons, identified by Rosen (1990), Miller (1991), and Reynolds (1998), most of the machines allow for some degree of variability in the measured characteristics and update the file containing the reference sample after each use.

Of equal importance is the retinal blood vessel analysis introduced for commercial use as a security system by Eyedentify Inc. Eyedentify's system bases verifications on the unique blood vessel patterns in the retina of the user's eye. Eyedentify's retinal pattern verifier is equipped with a lighted concentric circle eye target and a headset. The back of the eye is similar to a map, and the machine reads only a small area of this map. Since no one knows which portion

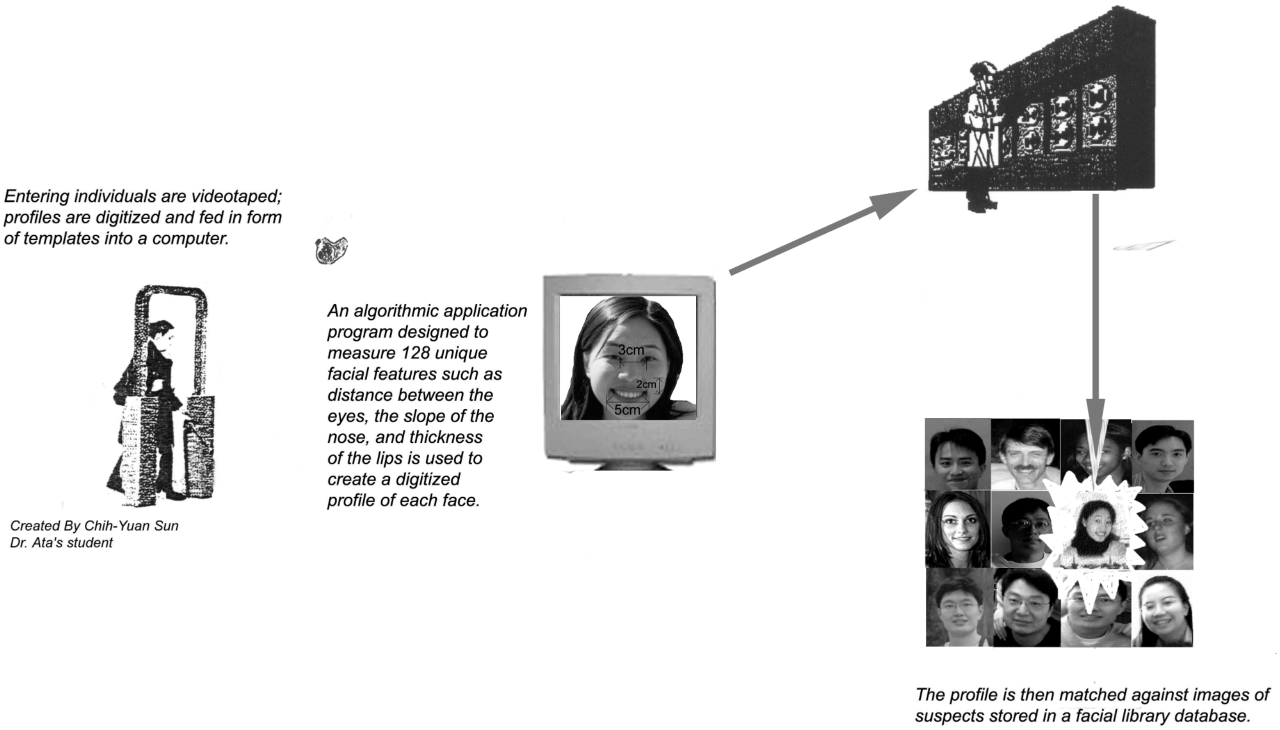

Entering individuals are videotaped; profiles are digitized and fed in form of templates into a computer.
An algorithmic application program designed to measure 128 unique facial features such as distance between the eyes, the slope of the nose, and thickness of the lips is used to create a digitized profile of each face.
3cm
2cm
5cm
Created By Chih-Yuan Sun
Dr. Ata's student
The profile is then matched against images of suspects stored in a facial library database.

**Figure 1** Face recognition system.

of the map is being read, there is no way to duplicate or imitate a retinal pattern. As in signature matching, which is prone to variation over time, a person's retinal patterns change if the subject experiences a heart attack or becomes diabetic, as noted by Sherman (1992).

Fingerprinting is one of the easiest ways to identify a user and serves as the most obvious way to distinguish users. In 1990 Rosen stated fingerprinting can range from the entire fingerprint to the exact portions of the finger. Machines used for fingerprint analysis have a cylindrical light that rotates around the finger and reads the bumps and ridges from a section of the finger. If the fingerprint matches the image stored, access is granted.

For hand geometry, methods that measure hand profile and thickness by light have been found very dependable. Miller (1991), Rosen (1990), and Sherman (1992) stated that a reader uses light to construct a three-dimensional image of a person's hand, examining such characteristics as finger length, width, and hand thickness. Also, in 1990, Rosen noted that the reference and template of the person's hand is updated following each verification.

Originally developed at MIT for law enforcement agencies, face printing combines the use of "fuzzy logic" and artificial intelligence algorithms. One system called FaceTrac, captures images by calculating distances between the eyes, thickness of the lips, angle of cheekbones, and other features such as slope of the nose . . . etc., and generates a profile for each facial characteristic. Such a system was installed during Super Bowl XXXV at the Raymond James Stadium in Tampa by Graphco Technologies, located in Pennsylvania, which markets FaceTrac. The Graphco program has the ability to measure up to 128 distinct facial features. The uniqueness of face printing is that no matter what kind of camouflaging techniques one may use to evade recognition, it can easily be unmasked. Faceprinting systems, previously thought to be too complex and expensive, can now be successfully developed. In 1992 Sherman explained that the machine uses low-cost, microprocessor driven cameras and fuzzy logic with neural networking firmware for the recording of facial images. Facial printing is an excellent method of identification because facial imprints are less susceptible to change from mood and nervousness. Its major advantage is that it doesn't require any physical contact with equipment.

Speech recognition is becoming more popular within the field of biometrics (Figs. 2 and 3). As with the fingerprint technique which analyzes patterns, the voice technique not only measures the voice but also other characteristics. Jaw opening and tongue shape and position are identified. To use this, the user enters a preassigned personal code on the voicekey keypad and then utters the password into the unit's microphone. A decision to grant or deny access is based on the matching of voice templates as described by Rosen (1990) and Tiogo (1990).

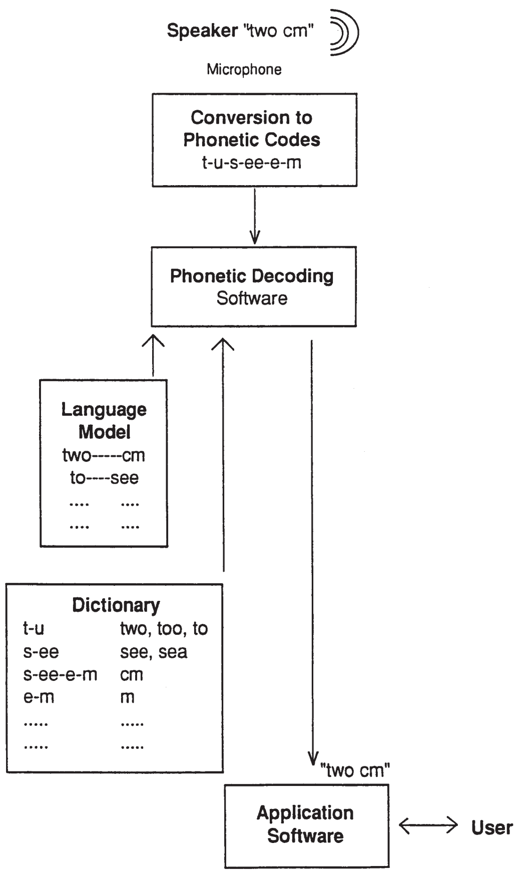


**Figure 2** Voice recognition system illustration.

Steganography, is a new entry in message delivery. The word means "covered writing" in Greek. In contrast to data encryption or cryptography, where data need to be decrypted and can easily be recognized as an encrypted message, steganography aims to hide its messages inside other harmless messages. The obvious message is so benign that it does not infer that it carries a second message. David Kahn's "The Code Breakers" provides an excellent accounting of this topic. Bruce Norman in his *Secret Warfare: The Battle of Codes and Ciphers* recounts numerous examples in the use of steganography during times of war. Null ciphers, commonly called unencrypted message, are also employed by this technique. The method camouflages the real message into a phonetic message.

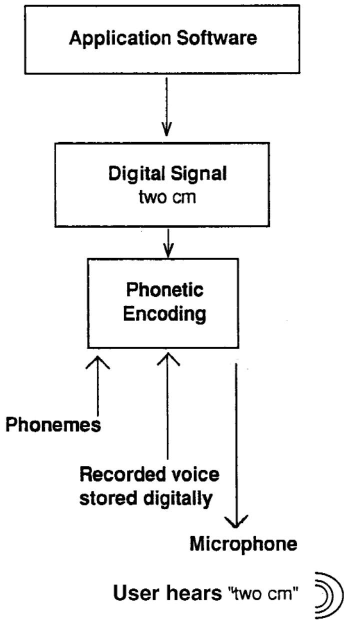


**Figure 3** Voice synthesis system illustration.

The advantage gained by the use of steganography is that when it is detected, a new steganographic application can be devised. The new application can be in the form of drawings, varying lines, colors, or other elements in pictures in order to conceal or reveal the message. Some available programs are JDeg.Shell, BPCS Steganography, PGM Stealth, and Piilo, a Unix-based program which uses images to hide messages.

As we have discussed, biometrics may be used for authentication and securing information resources. However, the practical use of biometrics systems is limited to verification. Verification means that the machine accepts or rejects the claim of an unknown person. The downside of these techniques is that they do not identify users who are not part of its set of reference samples.

The physical techniques are the other methods for computer security. These techniques are specifically designed for the network's safety by authorizing access to the network users through the use of electronic "tokens." The Digital Pathways Inc., of Mountain View, California produces a hand-held authorization device known as Secure Net Key. In 1987 the Delaware Valley Disaster Recovery Information Exchange Group reported that the device uses data encryption techniques to safeguard data transmissions from unauthorized intrusions, thereby reducing potential abuse. This was again stated by Taschek (1993) and Baig (1994). The National Semiconductor Corp. has also recently introduced a "Persona Card" which meets Computer Memory Card International Association (PCMCIA) standards and fits easily in the personal computer slots. It is designed to transmit encrypted data to the source for evaluation or processing. Security Dynamics of Cambridge, MA. has also created an access card known as "SECURE ID" which generates a random code every 60 seconds. Vacca (1994) reported that users must present the latest code to gain access.

More recently, many organizations have adopted a Single-Sign-On [SSO] approach for their computer security. This method is a software-driven technique which permits users to gain access into multiple interrelated applications using a limited number of passwords for user's identification and access control. Access control requirements differ for systems based on centralized concepts from those on decentralized format. Decentralized security devices from high-tech, high-cost random password generators, network routers, and call-back modems require "firewall" configurations. Allen (1977), and then Tiogo (1988) wrote that Information Systems of Glenwood, Maryland, Harris Computer Systems in Ft. Lauderdale, Florida, and Digital Equipment Corp. are among the companies that build firewall security gateways between private networks and Internet. The idea is that all traffic must pass through the firewall. The Sidewinder Fire Wall Software from Secure Computing Corp. of Roseville, Minnesota, lets businesses strike back by feeding an intruder false data and tracing him or her back to their computer.

Another company for physical security is Data Security Inc. of Redwood City, California. It produces user software that uses cryptography where it employs mathematical algorithms to scramble messages and create "digital signatures the equivalent of fingerprints" as explained by Snyder and Caswell (1989). Data Security Inc. worked on a joint venture with Enterprise Integration Technologies Corp. of Palo Alto, California to produce a system called TERISA-Systems which uses this cryptography technology to secure transactions on the World Wide Web (WWW) servers which was detailed by Nolle (1989), Rosen (1990), Snyder and Caswell (1989), Steinauer (1986), and Tiogo (1990).

## V. BACKUP/COPY RESTORATION TECHNIQUES

The necessity for backup operations may be one of the most overlooked and underplanned in the environment of technology. Because of the regulatory requirements, auditors now routinely verify an organization's ability to recover from a disaster. However, the necessity for backup operations is still one of the most overlooked and underplanned in the environment of technology. According to *Info Security News* magazine (http://www. infosecnews. com):

- Companies begin meltdown in less than 5 days after losing critical data.
- Fifty percent of companies that did not recover within 10 days, never fully recovered.
- Ninety-three percent of those companies went out of business within 5 years.

The various strategies and technologies employed in backup operations vary according to the type of operation, the function it is to serve, and the outcome expected.

For centralized operations, the prevalent backup technology remains the movement of data between disk and local tape. This has been increasingly employed in enterprise-level applications where the volumes of data stored for real-time access continues to escalate, driven by technologies such as data warehouses. However, the use of tape backup procedures has begun to come under efficiency pressure as the operational windows for backups shrinks and the volumes of data scale into the multi-terabyte environment. At the time of writing, various partnerships of technology giants have demonstrated the ability to backup data in excess of one terabyte per hour. However, as technologies and the use of technologies change, different problems arise. Backups to tape are sufficient when dealing with single or low number source data, such as enterprise databases. However, as employees move to remote computing, the number of storage locations increases and can conceivably be outside and disconnected from the enterprise. Strategic Research Corp., in a survey of over 200 sites reports that only 18% backed up workstations in addition to the servers. Alternatives to tape backups include parallel backups with tape arrays as well as real-time data replication through Redundant Array of Independent Disk (RAID) configurations. Since disk reads are superior to tape, backups can be performed very efficiently, although at a higher cost.

For distributed environments, which are now becoming much more prevalent due to open system configurations, techniques for the control of backup operations are moving away from server centric to the latest technology known as SAN (storage area networks). According to Dataquest, Inc., "software functionality does not have to reside on the server. The server does not always act as the intermediary for the backup. It can go directly from the user to the backup device." Additionally, more people are putting the software protocol, or agent, on the network. Soon the SAN itself will be able to create the backup.

Imposed on the exact protocol for creating backups are various site strategies and options for the physical storage of the backed up data. Off-site facilities are generally classified into three categories:

1. Hot sites—Separate locations where classified records, based on their priorities can be instantly replicated. These sites have computer workstations, file servers, their own functional security, backup generators, and dedicated or dial-up leased lines. This allows a failed operation to immediately access copies of damaged data and continue operations.
2. Warm sites—This refers to partially equipped backup sites. They consist of peripheral equipment in an off-line mode with minimal CPU capacity to enable the continuation of mission critical tasks only.
3. Cold sites—Sometimes referred to as "relocatable shells." These are sites without any resources, except suitable power supplies, phone lines, and ventilation. The shells are computer ready and transportable to the disaster site for the salvage of equipment and data where possible. The platforms can be salvageable technology or obtained on a short term leased basis.

Finally, one phenomenon which is finding its way into many areas of technology adoption and management, and which is observable within the area of backup and recovery, is that of outsourcing. Due to the scope and specialized and technical requirements of this work, backup procedures, processes and technologies can now be outsourced to third parties allowing removal of the management requirements and expertise from mainstream operations. Internet capabilities have particularly lent themselves to this format. For example, Datasave Services, PLC, provides a service of data storage and archiving in an off-site format. The method is to transfer all changes of a customer's data to an off-site location where it is stored on digital linear tape (DLT). This is archived via modem and the transfer is made overnight following encryption. This provides for extremely high levels of transport security while minimizing disruption to daily operations.

Examples of corporations that provide systems for supporting backup and restoration operations are

1. *Data Recovery System (DRS):* The Data Recovery System, developed by Integrity Solutions Inc., is located in Denver, Colorado. The DRS is enhanced with a backward file recovery capability which adds to all DRS/Update and

DRS/Recover. The backward file recovery uses "before images" to speed up the recovery process and reduce redundant backups. As indicated in *Software Magazine,* this feature is particularly useful when an abend(abnormal end) occurs near the end of a large job. The updated version of DRS also includes enhanced reporting capabilities, journal merge options, a compare facility, and control interval journaling. DRS supports CICS/VS, Vsam files and DL/1, and DB2 under DOS, AS400, or MVS/XA. The price ranges from $2000 to $17,000, depending on the size of mainframe selected.

2. *The Data Center Planner:* The Data Center Planner, developed by Vycor Corp., is located in Landover, Maryland. This is a series of PC software packages for managing DP assets. The Disaster Recovery Planner creates and maintains an on-line disaster recovery plan; Supply Planner controls data center supplies; PC Manager controls the inventory of PCs, including components and software; and Communication Pioneer tracks voices and data communication lines and wirepath.The price of this software ranges from $995 to $5000.
3. *VMCENTER II:* The VMCENTER II was developed by VM software, Inc., located in Reston, Virginia. This software provides system security and management of Direct Access Storage Devices (DASD), operations, performance, capacity, and recovery. Easy-to-use screens simplify auxiliary devices such as tape or disk mounting and scheduling tasks. The installation and maintenance procedure is centralized so that all components and features can be administered by a single user. The price ranges from $25,000 to $88,000, depending on the machine use.
4. *SPANEX:* SPANEX is an automated job scheduling and job restart system for MVS operation developed by Westinghouse Management Systems Software at Pittsburgh, Pennsylvania, with which one of the authors was involved. It provides allowance for daily schedule variations and interaction with external or noncomputer tasks or events. Controls may be centralized or decentralized. The scheduler's restart facility is built-in, and automatically detects the point of failure and initiates appropriate recovery plan. The system is priced at $20,000 and it runs under IBM MVS. The software group was sold to a British company in 1994.
5. *STRATUS (a subsidiary of SGI):* The STRATUS system handles failure without complicated housekeeping by hardwiring computer components together. This system works on one job at a time, but gives the job to two pairs of independent Motorola microprocessors so that it can compare results. Suppose the job at hand is adding $2 to $3. Each microprocessor crunches the numbers, and then the paired microprocessor compares results. If the results from one pair of microprocessors are $5 and $6, then those processors are shut down momentarily, while the computer simply uses the result from the other pair.
6. *Remote Journaling:* Remote journaling offers a higher level of data protection. Software monitors update to Virtual Sequential Access Method (VSAM) files in the IBM world. A remote on-line log is kept of all updates of records. If a failure occurs, data can be updated back to the point of failure. It may take a few hours to reconstruct files, depending on their size and the number of updates made. Both Sunguard and Comdisco Inc. offer remote journaling. Earlier this year IBM announced a plan to offer remote journaling software for its IMS and DB2 database. Some users are using distributed database management system (DBMS) technologies to provide on-line remote data protection.
7. *Electronic Vaulting:* Electronic vaulting is batch transmission of data networks to a remote location, while remote journaling, a more advanced technology, remotely records data updates as they occur. Database shadowing is the most advanced of these new services. With this technology, copies of entire databases are maintained at remote places. Electronic vaulting, which has existed in product form for about two years, allows for the batch transmission of critical data sets via T1 or T3 to a remote site, usually adjacent to standby data centers. One problem, however, is that products are not necessarily designed for multivendor environments. Sunguard Recovery Services of Wayne, Pennsylvania, which was the first to offer electronic vaulting services, originally marketed this service for IBM mainframes customers, but now covers all platforms of diverse systems.
8. *The CTAM Method:* Sheri Anderson, Senior Vice President for Production Systems and Services at Charles Schwab and Co. Inc. in San Francisco has come to depend on what is known in the disaster recovery business as the CTAM method for protecting and accessing critical data in an emergency. Chevy Track Access Method (CTAM)

works like this: users attempting to protect their data, copy them onto tape periodically and transport the tape by truck or jet to a second location for safekeeping. If a disaster occurs at a data center, the tapes can be rushed to a backup data center, loaded, and run. Unfortunately such an approach can result in loss of data if a disaster occurs any time after the last backup, and physically transporting tapes and loading data into backup CPUs can take several hours or even days. For this reason, several firms are evaluating technologies for transporting copies of important data to backup sites electronically. Vendors in the booming disaster recovery business are attempting to accommodate them by offering products and services for electronic vaulting and remote journaling of data. A few have even announced full database shadowing, which has the potential to recover all data back to the point of failure in a matter of minutes.

9. *IBM's Business Recovery Service:* IBM has set its sights on the disaster recovery market with the recent introduction of a service designed to back up data for its large system customers in the event of a disaster. IBM's Business Recovery Service (BRS) is designed to restore data on mid-range and mainframe systems, as well as PCs that are integrated into those environments through LANs. Under BRS, firms will be able to access backup data that is archived in one of 12 "hot sites" IBM is setting up across the nation. For PC data recovery, clients will be able to set up facilities closer to their home offices. In putting together a recovery plan, IBM would put together a solution that would define an alternate site in proximity to the customer's office. In an event of a flood or fire we would be able to go in and duplicate the work environment. Working with IBM, clients will develop a disaster recovery plan that stipulates square footage required in an alternative facility and also specifies cabling, LAN design, and necessary equipment. Should a disaster occur, clients will have access to equivalent systems for data recovery, and IBM will provide technical support. The fee for the disaster recovery service, which is available on a limited basis, range from $500 to $40,000 per month, depending on systems configuration. For example, pricing for a System/AS400 mid-range "with a fairly rich configuration" will cost approximately $40,000 per month. Clients can subscribe to BRS for 1-, 3- or 5-year periods.
10. *Ashton-Tate Corp.'s File Recovery, a subsidiary of Borland:* Ashton-Tate's File Recovery is a recovery program designed to diagnose and repair damaged data files. Originally designed for its Dbase File Recovery and now enhanced for PC-based relational data bases, it is almost completely automated. It successfully restores damaged files, but suffers from poor documentation and provides little guidance on the type of damage or the steps required to repair it.

## VI. CONCLUSION

Protecting a business information system and computer resources requires complete planning. The Computer Virus Industry Association recommends specific solutions to reduce problems regarding computer security from errors of omission or deliberate sabotage. These solutions have been outlined and discussed by multiple authors as well as the Delaware Valley Disaster Recovery Information Exchange Group. These solutions include:

1. Access control
2. Good documentation practices
3. Effective employee training
4. Proper dissemination of information
5. A system of checks and balances

Alternatively, Miller (1991) wrote that a business' security system should have physical safeguards (locks), administrative safeguards (policy), secondary storage safeguards (write-protect), software safeguards (access security), and communications safeguards (data encryption). This checklist is a basic "blueprint" to set up a successful computer security/disaster recovery planning system. Computer security involves the protection of computer hardware, software, and databases, from unauthorized use and possible deliberate destruction. Management has always been concerned with the protection of business and its client data. Management actions that can maximize security include:

- Segregation of duties in the information system environment
- Built-in internal and external system controls
- Audit trail for file and program access controls
- Use of security specialists
- Thorough personnel investigation before hiring
- Bonding of staff

- Prompt removal of discharged personnel
- Good documentation and crosstraining of personnel

As technology and the need for data storage and manipulation become more entwined and insidious in daily lives and operations, the need to protect these collections and allow for the recovery of damaged operations becomes more important. Technologies dealing with user and consumer access to these data continue to become more complex. Without adequate safeguards for security and backup procedures for recovery, institutions and societies reliant on these types of stored information become more susceptible to unauthorized intrusion and disruption.

## SEE ALSO THE FOLLOWING ARTICLES

Computer Viruses • Crime, Use of Computers in • Documentation for Software and IS Development • Firewalls • Security Issues and Measures • Systems Analysis

## BIBLIOGRAPHY

Burch, J. G., Jr., and Sardinas, J. L., Jr. (1978). *Computer control and audit—A total systems approach.* New York: John Wiley & Sons.

Clowes, K. W. (1998). *EDP auditing.* Toronto: Holt, Rinehart and Winston.

Garrett, P. (2001). *Making, breaking codes: An introduction to cryptology.* Upper Saddle River, NJ: Prentice Hall.

Lynch, R. M., and Williamson, R. W. (1976). *Accounting for management—Planning and control;* 2nd ed. New York: McGraw-Hill.

Murdick, R. G., Ross, J. E. , and Claggett, J. R. (1984). *Information systems for modern management,* 3rd ed. Englewood Cliffs, NJ: Prentice Hall.

Pfleeger, C. P. (2000). *Security in computing,* 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR.

Porter, T. W., and Perry, W. E. (1991). *EDP controls and auditing,* 5th ed. Boston: Kent Publishing.

Rahman, M., and Halladay, M. (1988). *Accounting information systems—Principles, applications, and future directions.* Englewood Cliffs, NJ: Prentice Hall.

Stallings, W. (2000). *Network security essentials: Applications and standards.* Upper Saddle River, NJ: Prentice Hall.

Weber, R. (1999). *Information systems control and audit.* Upper Saddle River, NJ: Prentice Hall.

# Discrete Event Simulation

**Jerry Banks**
*AutoSimulations, Inc.*

## GLOSSARY

**discrete-event simulation model** One in which the state variables change only at those discrete points in time at which events occur.

**event** An occurrence that changes the state of the system.

**model** A representation of a real system.

**simulation** The imitation of the operation of a real-world process or system over time. Simulation involves the generation of an artificial history of the system, and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system that is represented.

**system state variables** Collection of all information needed to define what is happening within the system to a sufficient level (i.e., to attain the desired output) at a given point in time.

## I. DEFINITION OF SIMULATION

Simulation is the imitation of the operation of a real-world process or system over time. Simulation involves the generation of an artificial history of the system, and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system that is represented.

Simulation is an indispensable problem-solving methodology for the solution of many real-world problems. Simulation is used to describe and analyze the behavior of a system, ask "what if" questions about the real system, and aid in the design of real systems. Both existing and conceptual systems can be modeled with simulation.

## II. SIMULATION EXAMPLE

Consider the operation of a one-teller bank where customers arrive for service between one and ten minutes apart in time, integer values only, each value equally likely. The customers are served in a time between 1 and 6 minutes, also integer valued, and equally likely. Restricting the times to integer values is an abstraction of reality, since time is continuous, but this aids in presenting the example. The objective is to simulate the bank operation, by hand, until twenty customers are served, and to compute measures of performance such as the percentage of idle time of the teller, the average waiting time per customer, etc. Admittedly, twenty customers is far too few to draw conclusions about the operation of the system for the long run.

To simulate the process, random interarrival and service times need to be generated. Assume that the interarrival times are generated using a spinner that has possibilities for the values 1 through 10. Further assume that the service times are generated using a die that has possibilities for the values 1 through 6.

Table I is called an ad hoc simulation table. The setup of the simulation table is for the purpose of this problem, but does not pertain to all problems.

Column 1, Customer, lists the 20 customers that arrive to the system. It is assumed that Customer 1 arrives at time 0, thus a dash is indicated in Row 1 of

**Table I** Ad Hoc Simulation

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| | Time | | | | Time | Time | | |
| | between | Arrival | Service | Service | service | in | Idle in | |
| Customer | arrivals | time | time | begins | ends | system time | Queue | Time |
| 1 | — | 0 | 2 | 0 | 2 | 2 | 0 | 0 |
| 2 | 5 | 5 | 2 | 5 | 7 | 2 | 3 | 0 |
| 3 | 1 | 6 | 6 | 7 | 13 | 7 | 0 | 1 |
| — | | | | | | | | |
| — | | | | | | | | |
| — | | | | | | | | |
| 18 | 4 | 84 | 5 | 84 | 89 | 5 | 0 | 0 |
| 19 | 7 | 91 | 3 | 91 | 94 | 3 | 2 | 0 |
| 20 | 7 | 98 | 1 | 98 | 99 | 1 | 4 | 0 |
| Totals | | | | | | 72 | 34 | 7 |

Column 2, Time between Arrivals. Rows 2 through 20 of Column 2 were generated using the spinner. Column 3, Arrival Time, shows the simulated arrival times. Since Customer 1 is assumed to arrive at time 0, and there is a 5-minute interarrival time, Customer 2 arrives at time 5. There is a 1-minute interarrival time for Customer 3, thus, the arrival occurs at time 6. This process of adding the interarrival time to the previous arrival time is called bootstrapping. By continuing this process, the arrival times of all 20 customers are determined. Column 4, Service Time, contains the simulated service times for all 20 customers. These were generated by rolling the die.

Now, the simulation of the service process begins. At time 0, Customer 1 arrived, and immediately began service. The service time was 2 minutes, so the service period ended at time 2. The total time in the system for Customer 1 was 2 minutes. The bank teller was not idle since the simulation began with the arrival of a customer. The customer did not have to wait for the teller.

At time 5, Customer 2 arrived, and immediately began service as shown in Column 5. The service time was 2 minutes so the service period ended at time 7 as shown in Column 6. The bank teller was idle from time 2 until time 5, so 3 minutes of idle time occurred. Customer 2 spent no time in the queue.

Customer 3 arrived at time 6, but service could not begin until time 7 as Customer 2 was being served until time 7. The service time was 6 minutes, so service was completed at time 13. Customer 3 was in the system from time 6 until time 13, or for 7 minutes as indicated in Column 7, Time in System. Although there was no idle time, Customer 3 had to wait in the queue for 1 minute for service to begin.

This process continues for all 20 customers, and the totals shown in Columns 7 (Time in System), 8 (Idle Time), and 9 (Time in Queue) are entered. Some performance measures can now be calculated as follows:

$$\text{Average time in system} = 72/20 = 3.6 \text{ minutes}$$

$$\%\text{ idle time} = [34/99](100) = 34\%$$

$$\text{Average waiting time per customer} = 10/20 = 0.5 \text{ minutes}$$

$$\text{Fraction having to wait} = 5/20 = 0.25$$

$$\text{Average waiting time of those that waited} = 7/3 = 2.33 \text{ minutes}$$

This very limited simulation indicates that the system is functioning well. Only 25% of the customers had to wait. About one-third of the time the teller is idle. Whether a slower teller should replace the current teller depends on the cost of having to wait versus any savings from having a slower server.

This small simulation can be accomplished by hand, but there is a limit to the complexity of problems that can be solved in this manner. Also, the number of customers that must be simulated could be much larger than 20 and the number of times that the simulation must be run for statistical purposes could be large. Hence, using the computer to solve real simulation problems is almost always appropriate.

## III. MODELING CONCEPTS

There are several concepts underlying simulation. These include system and model, events, system state variables, entities and attributes, list processing, activities and delays, and finally the definition of discrete-event simulation.

### A. System, Model, and Events

A model is a representation of an actual system. Immediately, there is a concern about the limits or boundaries of the model that supposedly represents the system. The model should be complex enough to answer the questions raised, but not too complex.

Consider an event as an occurrence that changes the state of the system. In the example, events include the arrival of a customer for service at the bank, the beginning of service for a customer, and the completion of a service.

There are both internal and external events, also called endogenous and exogenous events, respectively. For example, an endogenous event in the example is the beginning of service of the customer since that is within the system being simulated. An exogenous event is the arrival of a customer for service since that occurrence is outside of the system. However, the arrival of a customer for service impinges on the system, and must be taken into consideration.

This encyclopedia entry is concerned with discrete-event simulation models. These are contrasted with other types of models such as mathematical models, descriptive models, statistical models, and input-output models. A discrete-event model attempts to represent the components of a system and their interactions to such an extent that the objectives of the study are met. Most mathematical, statistical, and input-output models represent a system's inputs and outputs explicitly, but represent the internals of the model with mathematical or statistical relationships. An example is the mathematical model from physics,

$$\text{Force} = \text{Mass} \times \text{Acceleration}$$

based on theory. Discrete-event simulation models include a detailed representation of the actual internals.

Discrete-event models are dynamic, i.e., the passage of time plays a crucial role. Most mathematical and statistical models are static in that they represent a system at a fixed point in time. Consider the annual budget of a firm. This budget resides in a spreadsheet. Changes can be made in the budget and the spreadsheet can be recalculated, but the passage of time is usually not a critical issue. Further comments will be made about discrete-event models after several additional concepts are presented.

### B. System State Variables

The system state variables are the collection of all information needed to define what is happening within the system to a sufficient level (i.e., to attain the desired output) at a given point in time. The determination of system state variables is a function of the purposes of the investigation, so what may be the system state variables in one case may not be the same in another case, even though the physical system is the same. Determining the system state variables is as much an art as a science. However, during the modeling process, any omissions will readily come to light. (On the other hand, unnecessary state variables may be eliminated.)

In the bank teller example, we might have the following system state variables—at clock time 5 we might have system state variables $LQ(5) = 0$ and $LS(5) = 1$. This is interpreted as the number in the queue at time 5 is 0, and the number in the system at time 5 is 1.

Having defined system state variables and given an example, a contrast can be made between discrete-event models and continuous models based on the variables needed to track the system state. The system state variables in a discrete-event model remain constant over intervals of time and change value only at certain well-defined points called event times. Continuous models have system state variables defined by differential or difference equations giving rise to variables that may change continuously over time.

Some models are mixed discrete-event and continuous. There are also continuous models that are treated as discrete-event models after some reinterpretation of system state variables, and vice versa.

### C. Entities and Attributes

An entity represents an object that requires explicit definition. An entity can be dynamic in that it "moves" through the system, or it can be static in that it serves other entities. In the example, the customer is a dynamic entity, whereas the bank teller is a static entity.

An entity may have attributes that pertain to that entity alone. Thus, attributes should be considered as local values. In the example, an attribute of the entity could be the time of arrival. Attributes of interest in one investigation may not be of interest in another investigation. Thus, if red parts and blue parts are being manufactured, the color could be an attribute. However, if the time in the system for all parts is of concern, the attribute of color may not be of importance. From this example, it can be seen that many entities can have the same attribute or attributes (i.e., more than one part may have the attribute "red").

## D. Resources

A resource is an entity that provides service to dynamic entities. The resource can serve one or more than one dynamic entity at the same time, i.e., operate as a parallel server. A dynamic entity can request one or more units of a resource. If denied, the requesting entity joins a queue, or takes some other action (i.e., diverted to another resource, ejected from the system). (Other terms for queues include files, chains, buffers, and waiting lines.) If permitted to capture the resource, the entity remains for a time, then releases the resource. In the bank example, the teller is a resource.

There are many possible states of the resource. Minimally, these states are idle and busy. But other possibilities exist including failed, blocked, or starved.

## E. List Processing

Entities are managed by allocating them to resources that provide service, by attaching them to event notices thereby suspending their activity into the future, or by placing them into an ordered list. Lists are used to represent queues.

Lists are often processed according to first in first out (FIFO), but there are many other possibilities. For example, the list could be processed by last in first out (LIFO), according to the value of an attribute, or randomly, to mention a few. An example where the value of an attribute may be important is in shortest processing time (SPT) scheduling. In this case, the processing time may be stored as an attribute of each entity. The entities are ordered according to the value of that attribute with the lowest value at the head or front of the queue.

## F. Activities and Delays

An activity is a duration of time whose duration is known prior to commencement of the activity. Thus, when the duration begins, its end can be scheduled. The duration can be a constant, a random value from a statistical distribution, the result of an equation, input from a file, or computed based on the event state. For example, a service time may be a constant 10 minutes for each entity; it may be a random value from an exponential distribution with a mean of 10 minutes; it could be 0.9 times a constant value from clock time 0 to clock time 4 hours, and 1.1 times the standard value after clock time 4 hours; or it could be 10 minutes when the preceding queue contains at most 4 entities and 8 minutes when there are 5 or more in the preceding queue.

A delay is an indefinite duration that is caused by some combination of system conditions. When an entity joins a queue for a resource, the time that it will remain in the queue may be unknown initially since that time may depend on other events that may occur. An example of another event would be the arrival of a rush order that preempts the resource. When the preemption occurs, the entity using the resource relinquishes its control instantaneously. Another example is a failure necessitating repair of the resource.

Discrete-event simulations contain activities that cause time to advance. Most discrete-event simulations also contain delays as entities wait. The beginning and ending of an activity or delay is an event.

## G. Discrete-Event Simulation Model

Sufficient modeling concepts have been defined so that a discrete-event simulation model can be defined as one in which the state variables change only at those discrete points in time at which events occur. Events occur as a consequence of activity times and delays. Entities may compete for system resources, possibly joining queues while waiting for an available resource. Activity and delay times may "hold" entities for durations of time.

A discrete-event simulation model is conducted over time ("run") by a mechanism that moves simulated time forward. The system state is updated at each event along with capturing and freeing of resources that may occur at that time.

# IV. ADVANTAGES AND DISADVANTAGES OF SIMULATION

Competition in the computer industry has led to technological breakthroughs that are allowing hardware companies to continually produce better products. It seems that every week another company announces its latest release, each with more options, memory, graphics capability, and power.

What is unique about new developments in the computer industry is that they often act as a springboard for other related industries to follow. One industry in particular is the simulation-software industry. As computer hardware becomes more powerful, more accurate, faster, and easier to use, simulation software does too.

The number of businesses using simulation is rapidly increasing. Many managers are realizing the benefits of utilizing simulation for more than just the one-time remodeling of a facility. Rather, due to advances in software, managers are incorporating simulation in their daily operations on an increasingly regular basis.

## A. Advantages

For most companies, the benefits of using simulation go beyond just providing a look into the future. These benefits are mentioned by many authors and are included in the following:

- **Choose correctly.** Simulation lets you test every aspect of a proposed change or addition without committing resources to their acquisition. This is critical, because once the hard decisions have been made, the bricks have been laid, or the material-handling systems have been installed, changes and corrections can be extremely expensive. Simulation allows you to test your designs without committing resources to acquisition.
- **Time compression and expansion.** By compressing or expanding time simulation allows you to speed up or slow down phenomena so that you can thoroughly investigate them. You can examine an entire shift in a matter of minutes if you desire, or you can spend two hours examining all the events that occurred during one minute of simulated activity.
- **Understand "Why?"** Managers often want to know why certain phenomena occur in a real system. With simulation, you determine the answer to the "why" questions by reconstructing the scene and taking a microscopic examination of the system to determine why the phenomenon occurs. You cannot accomplish this with a real system because you cannot see or control it in its entirety.
- **Explore possibilities.** One of the greatest advantages of using simulation software is that once you have developed a valid simulation model, you can explore new policies, operating procedures, or methods without the expense and disruption of experimenting with the real system. Modifications are incorporated in the model, and you observe the effects of those changes on the computer rather than the real system.
- **Diagnose problems.** The modern factory floor or service organization is very complex—so complex that it is impossible to consider all the interactions taking place in one given moment. Simulation allows you to better understand the interactions among the variables that make up such complex systems. Diagnosing problems and gaining insight into the importance of these variables increases your understanding of their important effects on the performance of the overall system.

The last three claims can be made for virtually all modeling activities, queueing, linear programming, etc. However, with simulation the models can become very complex and, thus, have a higher fidelity, i.e., they are valid representations of reality.

- **Identify constraints.** Production bottlenecks give manufacturers headaches. It is easy to forget that bottlenecks are an effect rather than a cause. However, by using simulation to perform bottleneck analysis, you can discover the cause of the delays in work-in-process, information, materials, or other processes.
- **Develop understanding.** Many people operate with the philosophy that talking loudly, using computerized layouts, and writing complex reports convinces others that a manufacturing or service system design is valid. In many cases these designs are based on someone's thoughts about the way the system operates rather than on analysis. Simulation studies aid in providing understanding about how a system really operates rather than indicating an individual's predictions about how a system will operate.
- **Visualize the plan.** Taking your designs beyond CAD drawings by using the animation features offered by many simulation packages allows you to see your facility or organization actually running. Depending on the software used, you may be able to view your operations from various angles and levels of magnification, even 3-D. This allows you to detect design flaws that appear credible when seen just on paper in a 2-D CAD drawing.
- **Build consensus.** Using simulation to present design changes creates an objective opinion. You

avoid having inferences made when you approve or disapprove of designs because you simply select the designs and modifications that provided the most desirable results, whether it be increasing production or reducing the waiting time for service. In addition, it is much easier to accept reliable simulation results, which have been modeled, tested, validated, and visually represented, instead of one person's opinion of the results that will occur from a proposed design.

- **Prepare for change.** We all know that the future will bring change. Answering all of the "what-if" questions is useful for both designing new systems and redesigning existing systems. Interacting with all those involved in a project during the problem-formulation stage gives you an idea of the scenarios that are of interest. Then you construct the model so that it answers questions pertaining to those scenarios. What if an automated guided vehicle (AGV) is removed from service for an extended period of time? What if demand for service increases by 10%? What if . . . ? The options are unlimited.
- **Wise investment.** The typical cost of a simulation study is substantially less than 1% of the total amount being expended for the implementation of a design or redesign. Since the cost of a change or modification to a system after installation is so great, simulation is a wise investment.
- **Train the team.** Simulation models can provide excellent training when designed for that purpose. Used in this manner, the team provides decision inputs to the simulation model as it progresses. The team, and individual members of the team, can learn by their mistakes, and learn to operate better. This is much less expensive and less disruptive than on-the-job learning.
- **Specify requirements.** Simulation can be used to specify requirements for a system design. For example, the specifications for a particular type of machine in a complex system to achieve a desired goal may be unknown. By simulating different capabilities for the machine, the requirements can be established.

## B. Disadvantages

The disadvantages of simulation include the following:

- **Model building requires special training.** It is an art that is learned over time and through experience. Furthermore, if two models of the same system are constructed by two competent individuals, they may have similarities, but it is highly unlikely that they will be the same.
- **Simulation results may be difficult to interpret.** Since most simulation outputs are essentially random variables (they are usually based on random inputs), it may be hard to determine whether an observation is a result of system interrelationships or randomness.
- **Simulation modeling and analysis can be time-consuming and expensive.** Skimping on resources for modeling and analysis may result in a simulation model and/or analysis that is not sufficient for the task.
- **Simulation may be used inappropriately.** Simulation is used in some cases when an analytical solution is possible, or even preferable. This is particularly true in the simulation of some waiting lines where closed-form queueing models are available, at least for long-run evaluation.

## C. Offsetting the Disadvantages

In defense of simulation, these four disadvantages, respectively, can be offset as follows:

- **Simulators.** Vendors of simulation software have been actively developing packages that contain models that only need input data for their operation. Such models have the generic tag "simulators" or templates.
- **Output analysis.** Most simulation-software vendors have developed output-analysis capabilities within their packages for performing very extensive analysis. This reduces the computational requirements on the part of the user, although they still must understand the analysis procedure.
- **Faster and faster.** Simulation can be performed faster today than yesterday, and even faster tomorrow. This is attributable to the advances in hardware that permit rapid running of scenarios. It is also attributable to the advances in many simulation packages. For example, many simulation software products contain constructs for modeling material handling using transporters such as conveyors, and automated guided vehicles.
- **Limitations of closed-form models.** Closed-form models are not able to analyze most of the complex systems that are encountered in practice. In nearly fourteen years of the author's consulting practice and current employment with a simulation software and consulting vendor, not one problem was encountered that could have been solved by a closed-form solution.

## V. STEPS IN A SIMULATION STUDY

Figure 1 shows a set of steps to guide a model builder in a thorough and sound simulation study.

### 1. Problem Formulation

Every simulation study begins with a statement of the problem. If the statement is provided by those that have the problem (client), the simulation analyst must take extreme care to insure that the problem is clearly understood. If a problem statement is prepared by the simulation analyst, it is important that the client understand and agree with the formulation. It is suggested that a set of assumptions be prepared by the simulation analyst and agreed to by the client. Even with all of these precautions, it is possible that the problem will need to be reformulated as the simulation study progresses.

### 2. Setting of Objectives and Overall Project Plan

Another way to state this step is "prepare a proposal." This step should be accomplished regardless of location of the analyst and client, viz., as an external or internal consultant. The objectives indicate the questions that are to be answered by the simulation study. The project plan should include a statement of the various scenarios that will be investigated. The plans for the study should be indicated in terms of time that will be required, personnel that will be used, hardware and software requirements if the client wants to run the model and conduct the analysis, stages in the investigation, output at each stage, cost of the study and billing procedures, if any.

### 3. Model Conceptualization

The real-world system under investigation is abstracted by a conceptual model, a series of mathematical and logical relationships concerning the components and the structure of the system. It is recommended that modeling begin simply and that the model grow until a model of appropriate complexity has been developed. For example, consider the model of a manufacturing and material-handling system. The basic model with the arrivals, queues, and servers is constructed. Then, add the failures and shift schedules. Next, add the material-handling capabilities. Finally, add the special features. Constructing an unduly complex model will add to the cost of the study and the time for its completion without increasing the quality of the output. Maintaining client involvement will enhance the quality of the resulting model and increase the client's confidence in its use.

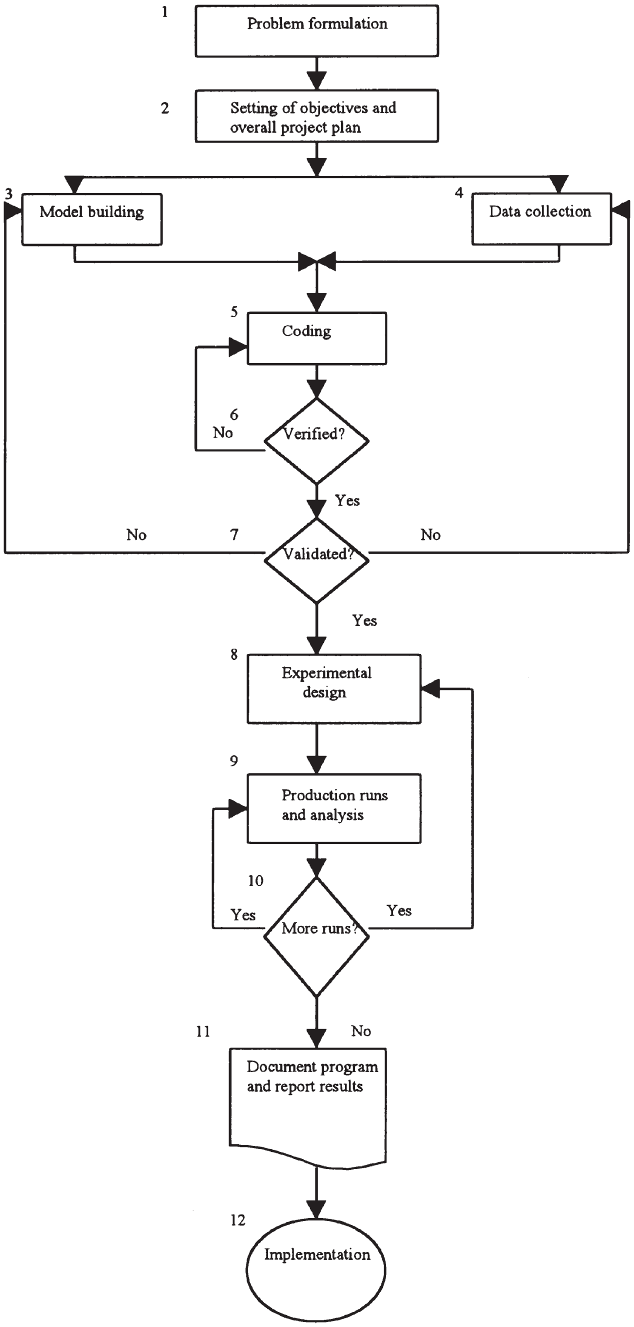


**Figure 1** Steps in a simulation study. [Reprinted with permission from Banks, J., Carson, J. S., Nelson, B. L., and Nicol, D. M. (2000). *Discrete event system simulation,* 3rd ed. Englewood Cliffs, NJ: Prentice Hall.]

### 4. Data Collection

Shortly after the proposal is "accepted" a schedule of data requirements should be submitted to the client. In the best of circumstances, the client has been collecting the kind of data needed in the format required, and can submit these data to the simulation analyst in electronic format. Oftentimes, the client indicates that the required data are indeed available. However, when the data are delivered they are found to be quite different than anticipated. For example, in the simulation of an airline reservation system, the simulation analyst was told "we have every bit of data that you want over the last five years." When the study commenced, the data delivered were the average "talk time" of the reservationist for each of the years. Individual values were needed, not summary measures. Model building and data collection are shown as contemporaneous in Fig. 1. This is to indicate that the simulation analyst can readily construct the model while the data collection is progressing.

### 5. Model Translation

The conceptual model constructed in Step 3 is coded into a computer recognizable form, an operational model.

### 6. Verified?

Verification concerns the operational model. Is it performing properly? Even with small textbook-sized models, it is quite possible that they have verification difficulties. These models are orders of magnitude smaller than real models (say 50 lines of computer code versus 2000 lines of computer code). It is highly advisable that verification take place as a continuing process. It is ill advised for the simulation analyst to wait until the entire model is complete to begin the verification process. Also, use of an interactive run controller, or debugger, is highly encouraged as an aid to the verification process.

### 7. Validated?

Validation is the determination that the conceptual model is an accurate representation of the real system. Can the model be substituted for the real system for the purposes of experimentation? If there is an existing system, call it the base system, then an ideal way to validate the model is to compare its output to that of the base system. Unfortunately, there is not always a base system. There are many methods for performing validation.

### 8. Experimental Design

For each scenario that is to be simulated, decisions need to be made concerning the length of the simulation run, the number of runs (also called replications), and the manner of initialization, as required.

### 9. Production Runs and Analysis

Production runs, and their subsequent analysis, are used to estimate measures of performance for the scenarios that are being simulated.

### 10. More Runs?

Based on the analysis of runs that have been completed, the simulation analyst determines if additional runs are needed and if any additional scenarios need to be simulated.

### 11. Documentation and Reporting

Documentation is necessary for numerous reasons. If the simulation model is going to be used again by the same or different analysts, it may be necessary to understand how the simulation model operates. This will enable confidence in the simulation model so that the client can make decisions based on the analysis. Also, if the model is to be modified, this can be greatly facilitated by adequate documentation.

The result of all the analysis should be reported clearly and concisely. This will enable the client to review the final formulation, the alternatives that were addressed, the criterion by which the alternative systems were compared, the results of the experiments, and analyst recommendations, if any.

### 12. Implementation

The simulation analyst acts as a reporter rather than an advocate. The report prepared in Step 11 stands on its merits, and is just additional information that the client uses to make a decision. If the client has been involved throughout the study period, and the simulation analyst has followed all of the steps rigorously, then the likelihood of a successful implementation is increased.

## SEE ALSO THE FOLLOWING ARTICLES

Continuous System Simulation • Model Building Process • Optimization Models • Simulation Languages • Software Process Simulation

## BIBLIOGRAPHY

Banks, J., ed. (1998). *Handbook of simulation: Principles, methodology, advances, applications, and practice.* New York: John Wiley.

Banks, J., Carson, J. S., Nelson, B. L., and Nicol, D. M. (2000). *Discrete event system simulation,* 3rd ed. Upper Saddle River, NJ: Prentice Hall.

Banks, J., and Norman, V. (November 1995). Justifying simulation in today's manufacturing environment. *IIE Solutions.*

Carson, J. S. (1993). Modeling and simulation world views, in *Proceedings of the 1993 Winter Simulation Conference,* (G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles, eds.) pp. 18–23, 1–4, Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Law, A. M., and Kelton, W. D. (2000). *Simulation modeling and analysis,* 3rd ed. New York: McGraw-Hill.

Pegden, C. D., Shannon, R. E., and Sadowski, R. P. (1995). *Introduction to simulation using SIMAN,* 2nd ed. New York: McGraw-Hill.

Schriber, T. J. (1991). *An introduction to simulation using GPSS/H.* New York: John Wiley.

# Distributed Databases

**M. Tamer Özsu**
*University of Waterloo*

## GLOSSARY

**atomicity** The property of transaction processing whereby either all the operations of a transaction are executed or none of them are (all-or-nothing).

**client/server architecture** A distributed/parallel DBMS architecture where a set of client machines with limited functionality access a set of servers which manage data.

**concurrency control algorithm** Algorithm that synchronize the operations of concurrent transactions that execute on a shared database.

**deadlock** An occurrence where each transaction in a set of transactions circularly waits on locks that are held by other transactions in the set.

**distributed database management system** A database management system that manages a database that is distributed across the nodes of a computer network and makes this distribution transparent to the users.

**durability** The property of transaction processing whereby the effects of successfully completed (i.e., committed) transactions endure subsequent failures.

**isolation** The property of transaction execution which states that the effects of one transaction on the database are isolated from other transactions until the first completes its execution.

**locking** A method of concurrency control where locks are placed on database units (e.g., pages) on behalf of transactions that attempt to access them.

**logging protocol** The protocol that records, in a separate location, the changes that a transaction makes to the database before the change is actually made.

**one copy equivalence** Replica control policy that asserts that the values of all copies of a logical data item should be identical when the transaction that updates that item terminates.

**query optimization** The process by which the "best" execution strategy for a given query is found from among a set of alternatives.

**query processing** The process by which a declarative query is translated into low-level data manipulation operations.

**quorum-based voting algorithm** A replica control protocol where transactions collect votes to read and write copies of data items. They are permitted to read or write data items if they can collect a quorum of votes.

**read-once/write-all protocol (ROWA)** The replica control protocol which maps each logical read operation to a read on one of the physical copies and maps a logical write operation to a write on all of the physical copies.

**serializability** The concurrency control correctness criterion that requires that the concurrent execution of a set of transactions be equivalent to the effect of some serial execution of those transactions.

**termination protocol** A protocol by which individual sites can decide how to terminate a particular transaction when they cannot communicate with other sites where the transaction executes.

**transaction** A unit of consistent and atomic execution against the database.

**transparency** Extension of data independence to distributed systems by hiding the distribution, fragmentation, and replication of data from the users.

**two-phase commit (2PC)** An atomic commitment protocol that ensures that a transaction is terminated the same way at every site where it executes. The name comes from the fact that two rounds of messages are exchanged during this process.

**two-phase locking** A locking algorithm where transactions are not allowed to request new locks once they release a previously held lock.

## I. INTRODUCTION

The maturation of database management system (DBMS) technology has coincided with significant developments in computer network and distributed computing technologies. The end result is the emergence of distributed DBMS. These systems have started to become the dominant data management tools for highly data-intensive applications. Many DBMS vendors have incorporated some degree of distribution into their products.

A *distributed database* (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network. A *distributed database management system* (distributed DBMS) is the software system that permits the management of the distributed database and makes the distribution transparent to the users. The term "distributed database system" (DDBS) is typically used to refer to the combination of DDB and the distributed DBMS. These definitions point to two identifying architectural principles. The first is that the system consists of a (possibly empty) set of *query sites* and a nonempty set of *data sites*. The data sites have data storage capability while the query sites do not. The latter only run the user interface routines in order to facilitate the data access at data sites. The second is that each site (query or data) is assumed to logically consist of a single, independent computer. Therefore, each site has its own primary and secondary storage, runs its own operating system (which may be the same or different at different sites), and has the capability to execute applications on its own. A computer network, rather than a multiprocessor configuration, interconnects the sites. The important point here is the emphasis on loose interconnection between processors that have their own operating systems and operate independently.

## II. DATA DISTRIBUTION ALTERNATIVES

A distributed database is physically distributed across the data sites by *fragmenting* and *replicating* the data. Given a relational database schema, fragmentation subdivides each relation into horizontal or vertical partitions. *Horizontal fragmentation* of a relation is accomplished by a selection operation that places each tuple of the relation in a different partition based on a fragmentation predicate (e.g., an `Employee` relation may be fragmented according to the location of the employees). *Vertical fragmentation* divides a relation into a number of fragments by projecting over its attributes (e.g., the `Employee` relation may be fragmented such that the `Emp_number`, `Emp_name`, and `Address` information is in one fragment, and `Emp_number`, `Salary`, and `Manager` information is in another fragment). Fragmentation is desirable because it enables the placement of data in close proximity to its place of use, thus potentially reducing transmission cost, and it reduces the size of relations that are involved in user queries. Based on the user access patterns, each of the fragments may also be *replicated*. This is preferable when the same data are accessed from applications that run at a number of sites. In this case, it may be more cost-effective to duplicate the data at a number of sites rather than continuously moving it between them. Figure 1 depicts a data distribution where `Employee`, `Project`, and `Assignment` relations are fragmented, replicated, and distributed across multiple sites of a distributed database.

## III. ARCHITECTURAL ALTERNATIVES

There are many possible alternatives for architecting a distributed DBMS. The simplest is the *client/server architecture,* where a number of client machines access a single database server. The simplest client/server systems involve a single server that is accessed by a number of clients (these can be called *multiple-client/single-server*). In this case, the database management problems are considerably simplified since the database is stored on a single server. The pertinent issues relate to the management of client buffers and the caching of data and (possibly) locks. The data management is done *centrally* at the single server. A more distributed, and more flexible, architecture is the *multiple-client/multiple-server* architecture where the database is distributed across multiple servers that have to communicate with each other in responding to user queries and in executing transactions. Each client machine has a "home" server to which it directs user requests. The communication of the servers among themselves is transparent to the users. Most current DBMSs implement one or the other type of

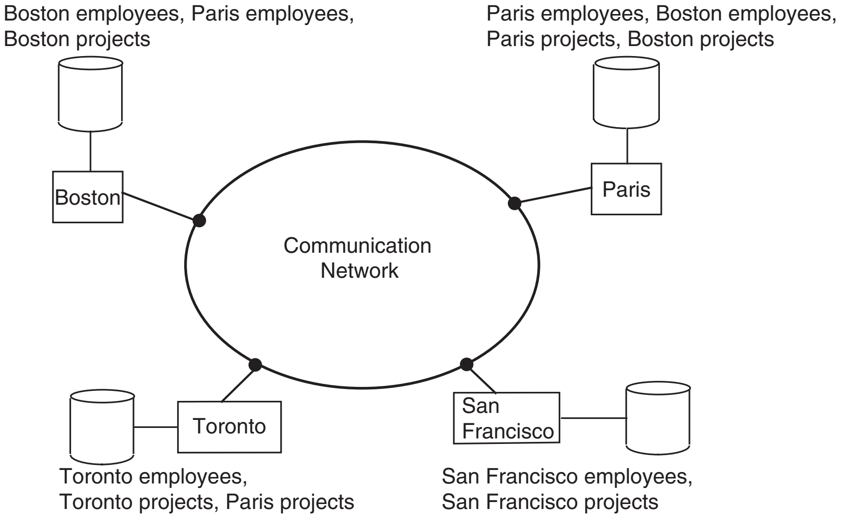

Boston employees, Paris employees,
Boston projects
Paris employees, Boston employees,
Paris projects, Boston projects
Boston
Paris
Communication
Network
Toronto
San
Francisco
Toronto employees,
Toronto projects, Paris projects
San Francisco employees,
San Francisco projects

**Figure 1** A fragmented, replicated, and distributed database example.

the client/server architectures. A truly distributed DBMS does not distinguish between client and server machines. Ideally, each site can perform the functionality of a client and a server. Such architectures, called *peer-to-peer,* require sophisticated protocols to manage data that is distributed across multiple sites. The complexity of required software has delayed the offering of peer-to-peer distributed DBMS products.

If the DDBSs at various sites are autonomous and (possibly) exhibit some form of heterogeneity, they are usually referred to as *multidatabase systems* or *federated database systems.* If the data and DBMS functionality distribution is accomplished on a multiprocessor computer, then it is referred to as a *parallel database system.* These are different than a DDBS where the logical integration among distributed data is tighter than is the case with multidatabase systems or federated database systems, but the physical control is looser than that in parallel DBMSs. In this article, we do not consider multidatabase systems or parallel database systems.

## IV. OVERVIEW OF TECHNICAL ISSUES

A distributed DBMS has to provide the same functionality that its centralized counterparts provide, such as support for declarative user queries and their optimization, transactional access to the database involving concurrency control and reliability, enforcement of integrity constraints, and others. In the remaining sections we discuss some of these functions; in this section we provide a brief overview.

*Query processing* deals with designing algorithms that analyze queries and convert them into a series of data manipulation operations. Besides the methodological issues, an important aspect of query processing is *query optimization.* The problem is how to decide on a strategy for executing each query over the network in the most cost-effective way, however, cost is defined. The factors to be considered are the distribution of data, communication costs, and lack of sufficient locally available information. The objective is to optimize where the inherent parallelism of the distributed system is used to improve the performance of executing the query, subject to the above-mentioned constraints. The problem is NP—hard in nature—and the approaches are usually heuristic.

User accesses to shared databases are formulated as *transactions,* which are units of execution that satisfy four properties: *atomicity, consistency, isolation,* and *durability*—jointly known as the ACID properties. Atomicity means that a transaction is an atomic unit and either the effects of all of its actions are reflected in the database, or none of them are. Consistency generally refers to the correctness of the individual transactions; i.e., that a transaction does not violate any of the integrity constraints that have been defined over the database. Isolation addresses the concurrent execution of transactions and specifies that actions of concurrent transactions do not impact each other. Finally, durability concerns the persistence of database changes in the face of failures. The ACID properties are enforced by means of concurrency control algorithms and reliability protocols.

*Concurrency control* involves the synchronization of accesses to the distributed database, such that the integrity of the database is maintained. The concurrency control problem in a distributed context is somewhat different than in a centralized framework. One not only has to worry about the integrity of a single database, but also about the consistency of multiple copies of the database. The condition that requires all the values of multiple copies of every data item to converge to the same value is called *mutual consistency*.

*Reliability protocols* deal with the termination of transactions, in particular, their behavior in the face of failures. In addition to the typical failure types (i.e., transaction failures and system failures), distributed DBMSs have to account for communication (network) failures as well. The implication of communication failures is that, when a failure occurs and various sites become either inoperable or inaccessible, the databases at the operational sites remain consistent and up to date. This complicates the picture, as the actions of these sites have to be eventually reconciled with those of failed ones. Therefore, recovery protocols coordinate the termination of transactions so that they terminate uniformly (i.e., they either abort or they commit) at all the sites where they execute. Furthermore, when the computer system or network recovers from the failure, the distributed DBMS should be able to recover and bring the databases at the failed sites up to date. This may be especially difficult in the case of network partitioning, where the sites are divided into two or more groups with no communication among them.

Distributed databases are typically replicated; that is, a number of the data items reside at more than one site. Replication improves performance (since data access can be localized) and availability (since the failure of a site does not make a data item inaccessible). However, management of replicated data requires that the values of multiple copies of a data item are the same. This is called the *one copy equivalence* property. Distributed DBMSs that allow replicated data implement replication protocols to enforce one copy equivalence.

## V. DISTRIBUTED QUERY OPTIMIZATION

*Query processing* is the process by which a declarative query is translated into low-level data manipulation operations. SQL is the standard query language that is supported in current DBMSs. Query optimization refers to the process by which the "best" execution strategy for a given query is found from among a set of alternatives.

In distributed DBMSs, the process typically involves four steps (Fig. 2): (1) query decomposition; (2) data localization; (3) global optimization; and (4) local optimization. Query decomposition takes an SQL query and translates it into one expressed in relational algebra. In the process, the query is analyzed semantically so that incorrect queries are detected and rejected as early as possible, and correct queries are simplified. Simplification involves the elimination of redundant predicates that may be introduced as a result of query modification to deal with views, security enforcement, and semantic integrity control. The simplified query is then restructured as an algebraic query.

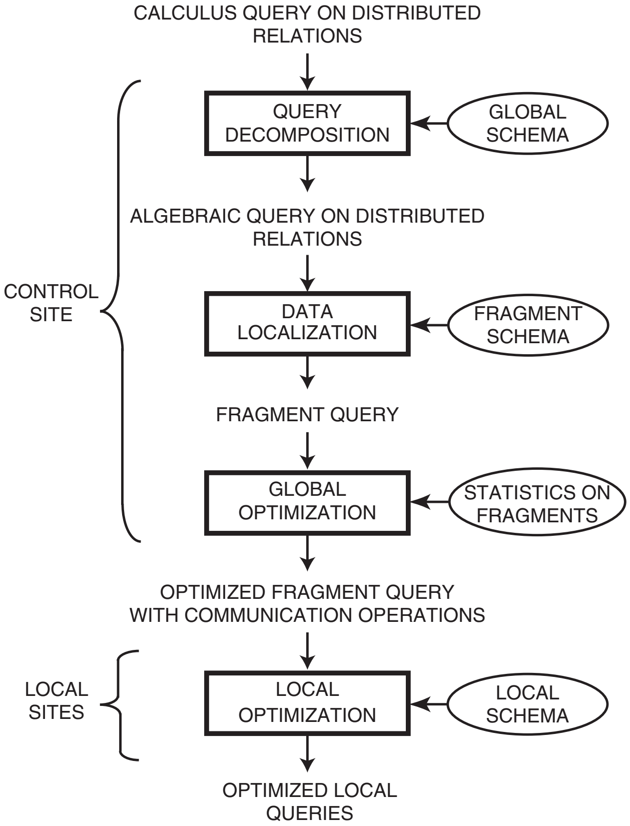


**Figure 2** Distributed query processing methodology.

The initial algebraic query generated by the query decomposition step is input to the second step: data localization. The initial algebraic query is specified on global relations irrespective of their fragmentation or distribution. The main role of data localization is to localize the query's data using data distribution information. In this step, the fragments that are involved in the query are determined and the query is transformed into one that operates on fragments rather than global relations. As indicated earlier, fragmentation is defined through fragmentation rules that can be expressed as relational operations (horizontal fragmentation by selection, vertical fragmentation by projection). A distributed relation can be reconstructed by applying the inverse of the fragmentation rules. This is called a *localization program*. The localization program for a horizontally (vertically) fragmented query is the union (join) of the fragments. Thus, during the data localization step each global relation is first replaced by its localization program, and then the resulting fragment query is simplified and restructured to produce an equivalent query that only involves fragments that contribute to the query result. Simplification and restructuring may be done according to the same rules used in the decomposition step. As in the decomposition step, the final fragment query is generally far from optimal; the process has only eliminated those queries whose performance is likely to be worse (due to their involvement of unnecessary fragments).

For a given SQL query, there is more than one possible algebraic query. Some of these algebraic queries are "better" than others. The quality of an algebraic query is defined in terms of expected performance. The process of query optimization involves taking the initial algebraic query and, using algebraic transformation rules, transforming it into other algebraic queries until the "best" one is found. The "best" algebraic query is determined according to a cost function that calculates the cost of executing the query according to that algebraic specification. In a distributed setting, the process involves global optimization to handle operations that involve data from multiple sites (e.g., join) followed by local optimization for further optimizing operations that will be performed at a given site.

The input to the third step, global optimization, is a fragment query, that is, an algebraic query on fragments. The goal of query optimization is to find an execution strategy for the query that is close to optimal. Remember that finding the optimal solution is computationally intractable. An execution strategy for a distributed query can be described with *relational algebra operations* and *communication primitives* (send/receive operations) for transferring data between sites. The previous layers have already optimized the query; for example, by eliminating redundant expressions. However, this optimization is independent of fragment characteristics such as cardinalities. In addition, communication operations are not yet specified. By permuting the order of operations within one fragment query, many equivalent query execution plans may be found. Query optimization consists of finding the "best" one among candidate plans examined by the optimizer.*

The final step, local optimization, takes a part of the global query (called a subquery) that will run at a particular site and optimizes it further. This step is very similar to query optimization in centralized DBMSs. Thus, it is at this stage that local information about data storage, such as indexes, are used to determine the best execution strategy for that subquery.

The query optimizer is usually modeled as consisting of three components: a search space, a cost model, and a search strategy. The *search space* is the set of alternative execution plans to represent the input query. These plans are equivalent, in the sense that they yield the same result but they differ on the execution order of operations and the way these operations are implemented. The *cost model* predicts the cost of a given execution plan. To be accurate, the cost model must have accurate knowledge about the parallel execution environment. The *search strategy* explores the search space and selects the best plan. It defines which plans are examined and in which order.

In a distributed environment, the cost function, often defined in terms of time units, refers to computing resources such as disk space, disk I/Os, buffer space, CPU cost, communication cost, etc. Generally, it is a weighted combination of I/O, CPU, and communication costs. Nevertheless, a typical simplification made by distributed DBMSs is to consider communication cost as the most significant factor. This is valid for wide area networks, where the limited bandwidth makes communication much more costly than it is in local processing. To select the ordering of operations it is necessary to predict execution costs of alternative candidate orderings. Determining execution costs before query execution (i.e., static optimization) is based on fragment statistics and the formulas for estimating the cardinalities of results of relational operations. Thus the optimization decisions depend on the available statistics on fragments. An important aspect of query optimization is *join ordering*, since permutations of the joins within the query may lead to improvements of several orders of magnitude. One basic technique for optimizing a sequence of distributed join operations is through use of the semijoin operator. The main value of the semijoin in a distributed system is to reduce the size of the join operands and thus the communication cost. However, more recent techniques, which consider local processing costs as well as communication costs, do not use semijoins because they might increase local processing costs. The output of the query optimization layer is an optimized algebraic query with communication operations included on fragments.

## VI. DISTRIBUTED CONCURRENCY CONTROL

Whenever multiple users access (read and write) a shared database, these accesses need to be synchronized to ensure database consistency. The synchronization is

*The difference between an optimal plan and the best plan is that the optimizer does not, because of computational intractability, examine all of the possible plans.

achieved by means of *concurrency control algorithms* that enforce a correctness criterion such as *serializability.* User accesses are encapsulated as transactions, whose operations at the lowest level are a set of read and write operations to the database. Concurrency control algorithms enforce the *isolation* property of transaction execution, which states that the effects of one transaction on the database are isolated from other transactions until the first completes its execution.

The most popular concurrency control algorithms are *locking*-based. In such schemes, a lock, in either shared or exclusive mode, is placed on some unit of storage (usually a page) whenever a transaction attempts to access it. These locks can be two types: *shared,* indicating that more than two transactions are allowed to access the data, and *exclusive,* indicating that the transaction needs to be the only one accessing data. Shared locks are also called *read locks,* since two transactions can read the same data unit, while exclusive locks are also called *write locks,* indicating that two transactions cannot revise the values of the data unit concurrently. The locks are placed according to lock compatibility rules such that *read-write, write-read,* and *write-write* conflicts are avoided. The compatibility rules are the following:

1. If transaction T1 holds a shared lock on data unit D1, transaction T2 can also obtain a shared lock on D1 (no conflict).
2. If transaction T1 holds a shared lock on data unit D1, transaction T2 cannot obtain an exclusive lock on D1 (read-write conflict).
3. If transaction T1 holds an exclusive lock on data unit D1, transaction T2 cannot obtain a shared lock (write-read conflict) or an exclusive lock (write-write conflict) on D1.

It is a well-known theorem that if lock actions on behalf of concurrent transactions obey a simple rule, then it is possible to ensure the serializability of these transactions: "No lock on behalf of a transaction should be set once a lock previously held by the transaction is released." This is known as *two-phase locking,* since transactions go through a growing phase when they obtain locks and a shrinking phase when they release locks. In general, releasing of locks prior to the end of a transaction is problematic. Thus, most of the locking-based concurrency control algorithms are *strict* in that they hold on to their locks until the end of the transaction.

In distributed DBMSs, the challenge is to extend both the serializability argument and the concurrency control algorithms to the distributed execution environment. In these systems, the operations of a given transaction may execute at multiple sites where they access data. In such a case, the serializability argument is more difficult to specify and enforce. The complication is due to the fact that the serialization order of the same set of transactions may be different at different sites. Therefore, the execution of a set of distributed transactions is serializable if and only if the execution of the set of transactions at each site is serializable, and the serialization orders of these transactions at all these sites are identical.

Distributed concurrency control algorithms enforce this notion of *global serializability.* In locking-based algorithms there are three alternative ways of enforcing global serializability: centralized locking, primary copy locking, and distributed locking algorithm.

In *centralized locking,* there is a single lock table for the entire distributed database. This lock table is placed, at one of the sites, under the control of a single lock manager. The lock manager is responsible for setting and releasing locks on behalf of transactions. Since all locks are managed at one site, this is similar to centralized concurrency control and it is straightforward to enforce the global serializability rule. These algorithms are simple to implement, but suffer from two problems. The central site may become a bottleneck, both because of the amount of work it is expected to perform and because of the traffic that is generated around it; and the system may be less reliable since the failure or inaccessibility of the central site would cause system unavailability. *Primary copy locking* is a concurrency control algorithm that is useful in replicated databases where there may be multiple copies of a data item stored at different sites. One of the copies is designated as a primary copy and it is this copy that has to be locked in order to access that item. All the sites know the set of primary copies for each data item in the distributed system, and the lock requests on behalf of transactions are directed to the appropriate primary copy. If the distributed database is not replicated, copy locking degenerates into a distributed locking algorithm.

In *distributed (or decentralized) locking,* the lock management duty is shared by all the sites in the system. The execution of a transaction involves the participation and coordination of lock managers at more than one site. Locks are obtained at each site where the transaction accesses a data item. Distributed locking algorithms do not have the overhead of centralized locking ones. However, both the communication overhead to obtain all the locks and the complexity of the algorithm are greater.

One side effect of all locking-based concurrency control algorithms is that they cause *deadlocks*. The detection and management of deadlocks in a distributed system is difficult. Nevertheless, the relative simplicity and better performance of locking algorithms make them more popular than alternatives such as *timestamp-based algorithms* or *optimistic concurrency control*.

## VII. DISTRIBUTED RELIABILITY PROTOCOLS

Two properties of transactions are maintained by reliability protocols: *atomicity* and *durability*. Atomicity requires that either all the operations of a transaction are executed or none of them are (all-or-nothing property). Thus, the set of operations contained in a transaction is treated as one atomic unit. Atomicity is maintained in the face of failures. Durability requires that the effects of successfully completed (i.e., committed) transactions endure subsequent failures.

The underlying issue addressed by reliability protocols is how the DBMS can continue to function properly in the face of various types of failures. In a distributed DBMS, four types of failures are possible: *transaction, site (system), media (disk),* and *communication*. Transactions can fail for a number of reasons: due to an error in the transaction caused by input data or by an error in the transaction code, or the detection of a present or potential deadlock. The usual approach to take in cases of transaction failure is to abort the transaction, resetting the database to its state prior to the start of the database.

Site (or system) failures are due to a hardware failure (e.g., processor, main memory, power supply) or a software failure (bugs in system code). The effect of system failures is the loss of main memory contents. Therefore, any updates to the parts of the database that are in the main memory buffers (also called *volatile database*) are lost as a result of system failures. However, the database that is stored in secondary storage (also called *stable database*) is safe and correct. To achieve this, DBMSs typically employ *logging protocols,* such as Write-Ahead Logging, which record changes to the database in system logs and move these log records and the volatile database pages to stable storage at appropriate times. From the perspective of distributed transaction execution, site failures are important since the failed sites cannot participate in the execution of any transaction.

Media failures refer to the failure of secondary storage devices that store the stable database. Typically, these failures are addressed by introducing redundancy of storage devices and maintaining archival copies of the database. Media failures are frequently treated as problems local to one site and therefore are not specifically addressed in the reliability mechanisms of distributed DBMSs.

The three types of failures described above are common to both centralized and distributed DBMSs. Communication failures, on the other hand, are unique to distributed systems. There are a number of types of communication failures. The most common ones are errors in the messages, improperly ordered messages, lost (or undelivered) messages, and line failures. Generally, the first two of these are considered to be the responsibility of the computer network protocols and are not addressed by the distributed DBMS. The last two, on the other hand, have an impact on the distributed DBMS protocols and, therefore, need to be considered in the design of these protocols. If one site is expecting a message from another site and this message never arrives, this may be because (1) the message is lost, (2) the line(s) connecting the two sites may be broken, or (3) the site that is supposed to send the message may have failed. Thus, it is not always possible to distinguish between site failures and communication failures. The waiting site simply timeouts and has to assume that the other site is incommunicado. Distributed DBMS protocols have to deal with this uncertainty. One drastic result of line failures may be network partitioning in which the sites form groups where communication within each group is possible but communication across groups is not. This is difficult to deal with in the sense that it may not be possible to make the database available for access while at the same time guaranteeing its consistency.

The enforcement of atomicity and durability requires the implementation of *atomic commitment protocols* and *distributed recovery protocols*. The most popular atomic commitment protocol is *two-phase commit*. The recoverability protocols are built on top of the local recovery protocols, which are dependent upon the supported mode of interaction (of the DBMS) with the operating system.

Two-phase commit (2PC) is a very simple and elegant protocol that ensures the atomic commitment of distributed transactions. It extends the effects of local atomic commit actions to distributed transactions by insisting that all sites involved in the execution of a distributed transaction agree to commit the transaction before its effects are made permanent (i.e., all sites terminate the transaction in the same manner). If all the sites agree to commit a transaction then all the actions of the distributed transaction take effect; if one of the sites declines to commit the operations at that site, then all of the other sites are required to abort the transaction. Thus, the fundamental 2PC rule states: if even one site rejects to commit (which means it votes to abort) the transaction, the distributed transaction has to be aborted at each site where it executes, and if all the sites vote to commit the transaction, the distributed transaction is committed at each site where it executes.

The simple execution of the 2PC protocol is as follows (Fig. 3). There is a *coordinator* process at the site where the distributed transaction originates, and *participant* processes at all the other sites where the transaction executes. Initially, the coordinator sends a "prepare" message to all the participants each of which independently determines whether or not it can commit the transaction at that site. Those that can commit send back a "vote-commit" message while those who are not able to commit send back a "vote-abort" message. Once a participant registers its vote, it cannot change it. The coordinator collects these messages and determines the fate of the transaction according to the 2PC rule. If the decision is to commit, the coordinator sends a "global-commit" message to all the participants; if the decision is to abort, it sends a "global-abort" message to those participants who had earlier voted to commit the transaction. No message needs to be sent to those participants who had originally voted to abort since they can assume, according to the 2PC rule, that the transaction is going

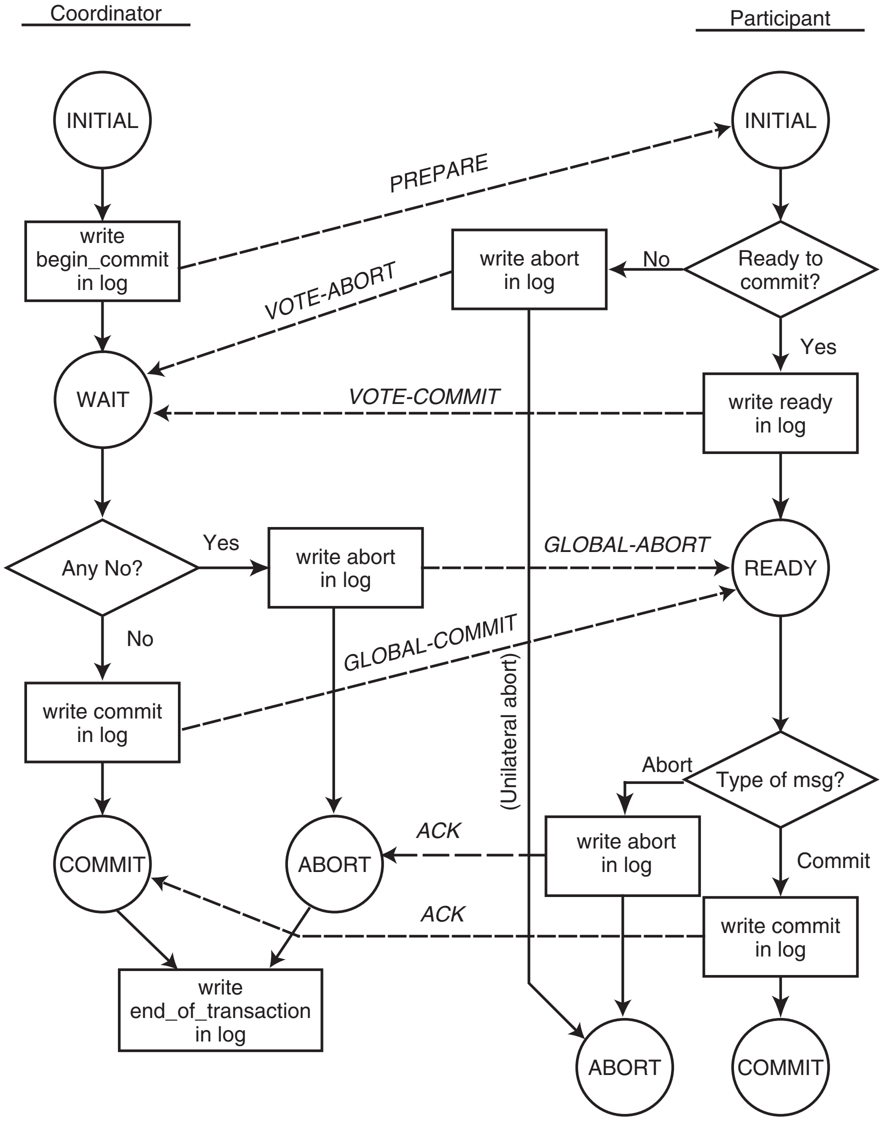


**Figure 3** 2PC protocol actions.

to be eventually globally aborted. This is known as the "unilateral abort" option of the participants.

There are two rounds of message exchanges between the coordinator and the participants; hence the name 2PC protocol. There are a number of variations of 2PC, such as the linear 2PC and distributed 2PC, that have not found much favor among distributed DBMS vendors. Two important variants of 2PC are the *presumed abort 2PC* and *presumed commit 2PC.* These are important because they reduce the message and I/O overhead of the protocols. Presumed abort protocol is included in the X/Open XA standard and has been adopted as part of the ISO standard for Open Distributed Processing.

One important characteristic of 2PC protocol is its *blocking* nature. Failures can occur during the commit process. As discussed above, the only way to detect these failures is by means of a timeout of the process waiting for a message. When this happens, the process (coordinator or participant) that timeouts follows a *termination protocol* to determine what to do with the transaction that was in the middle of the commit process. A nonblocking commit protocol is one whose termination protocol can determine what to do with a transaction in case of failures under any circumstance. In the case of 2PC, if a site failure occurs at the coordinator site and one participant site while the coordinator is collecting votes from the participants, the remaining participants cannot determine the fate of the transaction among themselves, and they have to remain blocked until the coordinator or the failed participant recovers. During this period, the locks that are held by the transaction cannot be released, which reduces the availability of the database. There have been attempts to devise nonblocking commit protocols (e.g., three-phase commit), but the high overhead of these protocols has precluded their adoption.

The inverse of termination is recovery. When a failed site recovers from the failure, what actions does it have to take to recover the database at that site to a consistent state? This is the domain of *distributed recovery protocols.* If each site can look at its own log and decide what to do with the transaction, then the recovery protocol is said to be *independent.* For example, if the coordinator fails after it sends the "prepare" command and while waiting for the responses from the participants, upon recovery, it can determine from its log where it was in the process and can restart the commit process for the transaction from the beginning by sending the "prepare" message one more time. If the participants had already terminated the transaction, they can inform the coordinator. If they were blocked, they can now resend their earlier votes and resume the commit process. However, this is not always possible and the failed site has to ask others for the fate of the transaction.

## VIII. REPLICATION PROTOCOLS

In replicated distributed databases, each logical data item has a number of physical instances. For example, the salary of an employee *(logical data item)* may be stored at three sites *(physical copies).* The issue in this type of a database system is to maintain some notion of consistency among the copies. The most discussed consistency criterion is *one copy equivalence,* which asserts that the values of all copies of a logical data item should be identical when the transaction that updates it terminates.

If replication transparency is maintained, transactions will issue read and write operations on a logical data item $x$. The replica control protocol is responsible for mapping operations on $x$ to operations on physical copies of $x$ ($x_1$, ..., $x_n$). A typical replica control protocol that enforces one copy equivalence is known as *Read-Once/Write-All* (ROWA) protocol. ROWA maps each read on $x$ [Read($x$)] to a read on one of the physical copies $x_i$ [Read($x_i$)]. The copy that is read is insignificant from the perspective of the replica control protocol and may be determined by performance considerations. On the other hand, each write on logical data item $x$ is mapped to a set of writes on all copies of $x$.

The ROWA protocol is simple and straightforward, but it requires that all copies of all logical data items that are updated by a transaction be accessible for the transaction to terminate. Failure of one site may block a transaction, reducing database availability.

A number of alternative algorithms have been proposed which reduce the requirement that all copies of a logical data item be updated before the transaction can terminate. They relax ROWA by mapping each write to only a subset of the physical copies. The *majority consensus algorithm* is one such algorithm which terminates a transaction as long as a majority of the copies can be updated. Thus, all the copies of a logical data item may not be updated to the new value when the transaction terminates.

This idea of possibly updating only a subset of the copies, but nevertheless successfully terminating the transaction, has formed the basis of quorum-based voting for replica control protocols. The majority consensus algorithm can be viewed from a slightly different perspective: it assigns equal votes to each copy and a transaction that updates that logical data item can successfully complete as long as it has a majority of the votes. Based on this idea, a *quorum-based voting algorithm* assigns a (possibly unequal) vote to each copy of a replicated data item. Each operation then has to obtain a *read quorum* ($V_r$) or a *write quorum* ($V_w$) to read or write a data item, respectively. If a given data item has a total of $V$ votes, the quorums have to obey the following rules:

1. $V_r + V_w > V$ (a data item is not read and written by two transactions concurrently, avoiding the read-write conflict);
2. $V_w > V_r/2$ (two write operations from two transactions cannot occur concurrently on the same data item; avoiding write-write conflict).

The difficulty with this approach is that transactions are required to obtain a quorum even to read data. This significantly and unnecessarily slows down read access to the database. An alternative quorum-based voting protocol that overcomes this serious performance drawback has also been proposed. However, this protocol makes unrealistic assumptions about the underlying communication system. It requires that all sites detect failures that change the network's topology instantaneously, and that each site has a view of the network consisting of all the sites with which it can communicate. In general, communication networks cannot guarantee to meet these requirements. The single copy equivalence replica control protocols are generally considered to be restrictive in terms of the availability they provide. Voting-based protocols, on the other hand, are considered too complicated with high overheads. Therefore, these techniques are not used in current distributed DBMS products. More flexible replication schemes have been investigated where the type of consistency between copies is under user control. A number of *replication servers* have been developed or are being developed with this principle. Unfortunately, there is no clear theory that can be used to reason about the consistency of a replicated database when the more relaxed replication policies are used.

## SEE ALSO THE FOLLOWING ARTICLES

Database Systems • Hyper-Media Databases • Management Information Systems • Object-Oriented Databases • Relational Database Systems • Structured Query Language • Temporal Databases

## BIBLIOGRAPHY

Bernstein, P. A., and Newcomer, E. (1997). *Principles of transaction processing for the systems professional.* San Mateo, CA: Morgan Kaufmann.

Gray, J., and Reuter, A. (1993). *Transaction processing: concepts and techniques.* San Mateo, CA: Morgan Kaufmann.

Helal, A. A., Heddaya, A. A., and Bhargava, B. B. (1997). *Replication techniques in distributed systems.* Boston, MA: Kluwer Academic Publishers.

Kumar, V., (ed.) (1996). *Performance of concurrency control mechanisms in centralized database systems.* Englewood Cliffs, NJ: Prentice Hall.

Özsu, M. T., and Valduriez, P. (1999). *Principles of distributed database systems,* 2nd ed. Englewood Cliffs, NJ: Prentice Hall.

Sheth, A., and Larson, J. (September 1990). Federated databases: Architectures and integration. *ACM Computing Surveys,* Vol. 22, No. 3, 183–236.

Yu, C., and Meng, W. (1998). *Principles of query processing for advanced database applictions.* San Francisco: Morgan Kaufmann.

# Documentation for Software and IS Development

**Thomas T. Barker**
*Texas Tech University*

## GLOSSARY

**application** A computer program, usually the "application" of a file-sharing, client-server, router, or other specialized technology.

**developer** A manager or producer within an organization who is responsible for creating software programs.

**development methodology** A series of stages of a software or hardware creation following a pattern based on experience and theory of program design.

**Information Process Maturity Model (IPMM)** A tool to create and measure processes used for technical publications whereby a core process repeats and replicates itself into a self-sustaining and constantly improving activity.

**software** A set of computer instructions, called "code," designed to perform a purpose.

**specifications (specs)** A document containing requirements for a computer program or software manual. It tells the contents and purpose of the program or document.

**DOCUMENTATION FOR SOFTWARE AND INFORMATION SYSTEMS DEVELOPMENT** is the text or discourse system whereby computer programs and systems used by organizations and society are created, explained, analyzed, and taught to the people who use them. Documentation falls into two types: development documentation and user documentation. Development documentation comprises those texts that support the programming activities required to produce the computer program. User documentation comprises those texts that teach, guide, and support work by novice and experienced users of the computer program. Often, but not always, these activities occur simultaneously, during development, so that the information obtained through both program user requirements research and audience analysis can inform and improve the actual product development. More commonly, however, the program is fully or nearly developed before it is turned over to the writers for user documentation.

The focus with documentation has traditionally been on the implementation side of computer software and hardware production, as opposed to the marketing side or the social and cultural side. It articulates activities associated with a recurrent cycle of development and implementation leading to more refined development and implementation. It uncovers and makes available to others in the discourse community the developers' thinking on how to make the best products possible within the constraints of users and organizations. By its very nature, documentation (especially the audience analysis and system testing activities) captures the needs of a population and makes it available in rich detail to those with the necessary technical skill to respond appropriately with yet better programs. In fact, the value of documentation comes from its ability to articulate important human activities, ones that operate at the very core of the information revolution.

Software documentation, both of the development and user type, has had to struggle against rejection by the community of users it dedicates itself to serving because of inconsistency in quality. Inconsistent quality has shaped the character of the documents produced to support computer programs. For instance, poor quality in early spreadsheet documentation sent writers to libraries and other design resources to reshape their work into something more functional and user-based. Thus, inconsistent quality shaped a whole trend in cognitive-psychology based document design techniques. Current trends in usability testing indicate another example of the shaping of design by reactions to charges of inconsistent quality.

In the area of development documentation, developers have to deal with a number of problems that plague software engineering: flaws in the design and development process, lack of quality assurance, and lack of consistent testing of product. The most important charge lies in the area of consistency in development. Because of the antiquated, design-first model in use, developers and clients had little way to realistically predict quality in software systems. Lack of an efficient process led software developers to develop more user-responsive methods—a process that led to many failures as well as successes and continues to this day.

## I. DEVELOPMENT DOCUMENTATION

Development documentation comprises those documents that propose, specify, plan, review, test, and implement the products of development teams in the software industry. Those products include programs as operating systems (Linux, Microsoft Windows), application programs (Microsoft Quicken, Eudora), and programming languages (C+ +, Visual Basic). Development documents reside on computer drives where librarians—so designated members of a development team—keep track of the information generated in them, making the most recent versions available to persons who will use it. Development documents include proposals, user or customer requirements descriptions, test and review reports (suggesting product improvements), and self-reflective documents written by team members, analyzing the process from their perspective. Most development documents go into the company's knowledge archive, a resource that becomes increasingly important with the development of new information tracking and control called "knowledge management."

## II. PURPOSE AND SCOPE OF DEVELOPMENT DOCUMENTATION

The purpose of development documentation is both to drive development and track development. So it serves to create product and also create information about development itself. In driving development it represents communication among members of the development team. Table I shows the members of a software development team and their duties.

The roles of development team members can vary greatly depending on specific projects. For example, some projects require intensive design, so a graphic artist or design specialist may join the team, or the quality assurance function might fall to separate persons such as a usability tester or "configuration manager" (someone who monitors the development stages themselves and the communication among members). But despite variations in the team, all members contribute to improving the software quality.

The onset of electronic communication tools—e-mail, file transfer, web pages—has greatly helped development team members communicate among themselves about their work. As we will see below, useful electronic forms such as discussion lists have developed to allow for user information, in the form of requirements or suggestions, to enter in a meaningful way into the development process and affect the outcome.

However, of the two purposes for development documentation, the tracking function, not the project management function, has led to greater quality improvement in software because it allows managers and team members (programmers, client representatives, writers, and testers) to reflect on their efforts and improve on them in subsequent development cycles. Typically, tracking information consists of production totals, cost totals broken down into categories of resource, cost per page of manuals, total man-hours on a project broken down into project participants, lines of code per hour, and lines of code per module. Tracking information also consists of usability data that publication and production designers can apply to the product cycle. Sophisticated project management systems allow managers to track finely grained behaviors that, theoretically, allow for optimal performance.

Tracking information for development projects surged with the introduction, during the 1980s, of computerized management systems in business and industry—the same wave that put millions of novice users in front of computer screens. Tracking information, which has a predictive aspect, collects the data in a timely fashion so managers can adjust the process.

**Table I** Software Development Team Members

| Role on the development team | Duties | Kinds of documents |
|---|---|---|
| Client | Assess investment potential of the project; arrange for funding | Requirements specifications, product review forms |
| Developer | Guides the project participants; arranges with client | Proposal, project correspondence |
| Project manager | Organizes and keeps the project on schedule; provides resources for members | Program specifications, document specifications, project plan, review report |
| Designer | Uses the software requirements document to create a design for the program | Design specifications |
| Programmer(s) | Writes computer instructions that conform with the design specifications document | Computer program files, internal program documentation, test reports |
| Technical writer(s) | Performs a user analysis; designs and writes user documentation | User's guide, user analysis report |
| Quality assurance manager | Plans and executes usability tests and process checks | Test report, quality assurance report |

Tracking information also helps inform managers' strategy and enhance their strategic power in development meetings. The information, collected efficiently and analyzed correctly, helps support the writer as advocate for user involvement in documentation. Finally, tracking information in the form of performance statistics for individual development team members can affect the hiring trends and, thus, the character of subsequent work in the mature organization.

## III. DEVELOPMENT MODELS IN SOFTWARE

Development models derive from the engineering community—such as the Software Engineering Institute at Carnegie Mellon University—or whatever discipline in which the product development occurred. These models basically follow two patterns: the waterfall method and the rapid application development method (both described below). The waterfall method benefits from the clear specification of the product at the time of design, and then the execution of that design specification through a series of stages based on institutional departments until the product is complete. The rapid application development method benefits from the experience of building a prototype or model of the entire product, and then testing that model and reforming it according to the test results until the product is complete. The third pattern, the object modeling method, varies the pattern of both types by allowing for both upfront design and prototyping but providing an improved coordination with programming.

### A. Waterfall Method and Its Documentation

The waterfall method of software documentation consists of a series of stages called "phases" of the development life cycle. The life cycle of the product includes the stages in Fig. 1. The waterfall method derives its name from the stair-step fashion by which development events proceed from one stage to another. This development method assumes that all or most of the important information about user requirements is available to the development team at the beginning of the project. The development team (usually cross-functional) then follows the stages from idea to implementation, with each stage building on the next and not really going back. The process assumes a sign-off from one phase to the next as each phase adds value to the developing product. Because of the phase-by-phase structure in this model, it is often used best to develop complex products requiring detailed specifications and efficient team communication. The degree of communication required to make this model work makes it difficult to handle in anything but a mature organization where resources, processes, and communication behaviors and protocols are well established.

In the waterfall method of development a heavy emphasis falls on the product specification document.

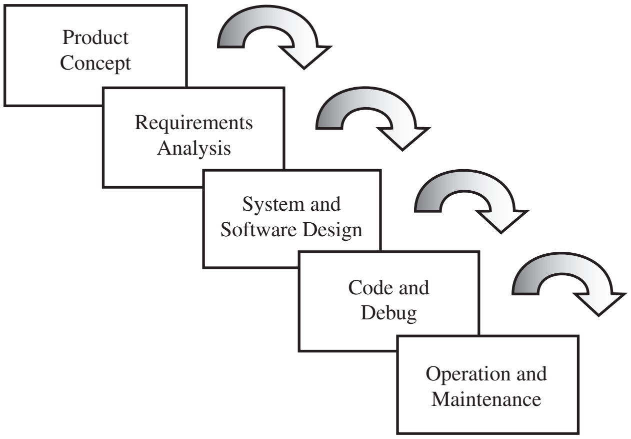


**Figure 1** The waterfall method.

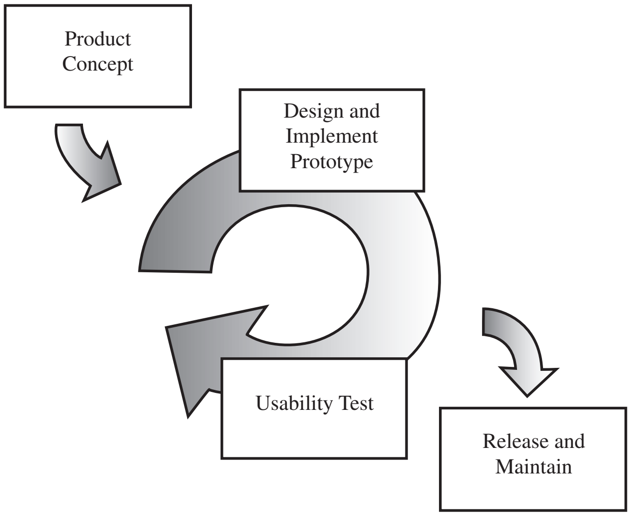


**Figure 2** The rapid application development method.

Seen as a blueprint for the entire project, the specification document needs to communicate with all the development team members (programmers, quality control persons, writers, sponsors, clients, managers, supervisors, and process control representatives). In the best of projects the product specifications document gets updated regularly to maintain its function as the central, directing script of the development. More commonly, however, the specification document gets forgotten as the programmers default to what is known as the *code and fix* process. The code and fix process is extremely time consuming and inefficient as it follows a random pattern of reacting to bugs and problems instead of a coordinated, document-driven process. The communication overhead required by the code and fix process can soon wreck the schedule and consume the entire remaining project budget. Besides that drawback, often the market window for a product would close up before this time-consuming process resulted in marketable product.

## B. Rapid Application Development Method and Its Documentation

The rapid application development (or RAD) method of development follows a different philosophy than the waterfall method. It capitalizes on one main problem of the waterfall method: the gradual divergence of what the programmers actually create from the original specifications (a digression sometimes called "code creep"). As illustrated in Fig. 2, the rapid application development method places an emphasis on user involvement (during cycles of testing) as an ongoing source of design innovation. In contrast to the waterfall method, the idea is to use prototypes of products (software program interface mockups) to draw design requirements from actual users and implement them quickly. Quick implementation (up to 1/3 the time used for the waterfall method) allows for teams to work more responsively to market demands and the demands of technological advances that also drive development.

The rapid application development model requires a very high-tech environment where software interface design tools and publications management processes allow for fast response to the ever-evolving specification of user needs requirements. For example, extensive usability testing often requires a lab or coordination with other testing groups in an organization. Budgeting and managing of tests, recruiting of test subjects, and planning for extra redesign meetings can slow down a process designed to be flexible and streamlined.

## C. Object Modeling and Its Documentation

Object modeling is a software development methodology that requires the developers and customers to express and record user requirements and development tasks using a consistent, highly abstract annotation system. Known as a modeling language, the annotation system for object modeling results in *case models* (embodying user specifications) that allow all the members of a development team to work in parallel, thus creating a very time-efficient method of production (see Fig. 3).

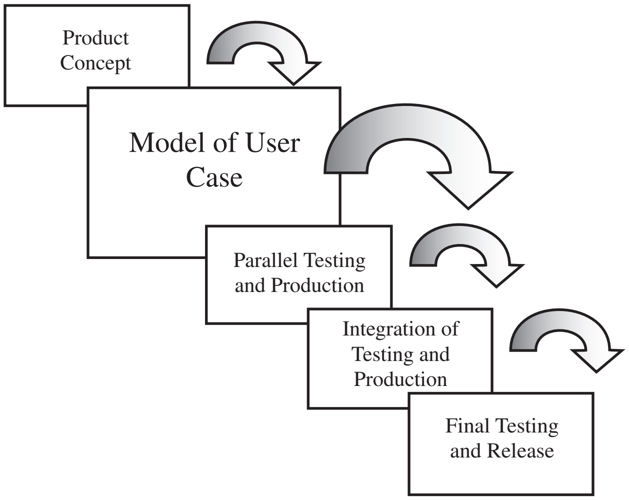

Product Concept
Model of User Case
Parallel Testing and Production
Integration of Testing and Production
Final Testing and Release

**Figure 3** The object modeling development method.

Object modeling allows developers and programmers to analyze the user's experience (say, "renting a video," or "filling a shopping cart") in ways that lead to greater degree of control over programming and development. This control comes from the ability of standardized languages such as the OPEN Process Specification and the Unified Modeling Language to express relationships among user requirements. Using the video rental store as an example, the case of "user rents a video" (which describes a user activity with the system) can be made dependent on other cases, such as "registers as a customer" and "has no outstanding rentals." With a finely grained analysis of dependencies and relationships among user requirements universally specified, the development team can cooperate independently and with increased efficiency.

At the level of management of the documentation process, the object modeling requires many of the same documents as the waterfall or the rapid application development methods. However, it relies heavily on diagrams (such as the user case model diagram or the business model diagram) and on declarations of modeling standards used in the particular project. These very complicated and conventional diagrams play the key role in describing the abstract model of user tasks and activities that lies at the heart of the object modeling method.

## IV. TYPES OF DEVELOPMENT DOCUMENTS

The genres of development documents fall into two categories: internal and external. Internal development documents track project status, report on changes, tests, reviews, and all the other tasks surrounding organizational projects. External development documents are documents intended for customers and other users of the finished software product.

### A. Product Specification (Internal)

A product specification document, often referred to as a product "spec" is the document that tells the persons in the project (developers, programmers, writers, marketers) what the outcomes or deliverables will be. These deliverables include the program and user documents. The specification document itself contains an overview of the concept for the software product, tools for production, list of program features, user requirements, and overall design of the program code.

### B. Project Management Plan (Internal)

A project management plan consists of statements that describe the actions taken by the development team using whatever method chosen to produce the software project. Like project plan documents in other engineering fields, it takes its shape and form from engineering management models. The management plan includes a schedule, project task list, list of responsibilities of team members, and any other documents (in an archive) relating to the project.

### C. Internal Code Documentation (Internal)

Internal code documentation consists of written text inside the actual program files that records information about a specific modules of code. A program that contains thousands of modules (each about a paragraph or so of outline-looking words and symbols) could contain an equal number of brief descriptive paragraphs written by the programmers at the time the code was written and tested. Internal code is written using special "remark" tags that distinguish the programmer's notes from the program code itself. Internal code documentation can help users who adapt programs (scripts such as java scripts and cgi scripts) and require only an explanation of how a module works and what variables are in it to make it work for them.

### D. Test/Usability Report (Internal)

The test report is one of the most important information gathering and synthesizing documents in the array of development documents. It records the results of the various forms of product testing done by programmers, writers, and, increasingly, usability experts on the development team. Increasingly, with object modeling models and rapid development, usability testing has become not just something done after a program is finished, but a method of development itself, allowing software engineers and technical writers to work closely with clients in application development.

Test reports usually contain descriptions of test situations, criteria for evaluators, test protocols, presentation of results, discussion of results, and a listing of rectifications done to the product as a result of the test.

### E. Maintenance Documentation (External)

Maintenance documentation is text, usually step-by-step procedures, that inform programmers and system administrators how to fix (sometimes called "patch") a program once it is in full operation. Maintenance also means resetting files after business cycles and other routine, nonproblem oriented programming. This kind of maintenance usually refers to large, enterprise programs such as those used in government, utilities, and manufacturing and shipping. Such programs do not get replaced often and frequently require programmers to add or alter functionality. Programmers rely on maintenance documentation (and internal code documentation, discussed above) to keep the program running and responding to evolving user requirements.

### F. System Overview Documents (Internal)

System overview documents are those documents that describe the overall technical and functional structure of a program. Readers of these documents are typically administrators in charge of the department supported by the technology, and sometimes software and hardware vendors interested in bidding on new systems or reengineering the old ones. System overviews themselves look like charts with boxes arranged in ways that represent the whole system, use cases, user scenarios, and other design and management documents.

## V. USER DOCUMENTATION

User documentation refers to texts directed to an audience of users of computer systems (as opposed to those who build, sell, and maintain it). Users make up a very diverse group because they work in all fields of industry, government, and education. The growth of diversity of users, in fact, is one of the most interesting stories in software documentation.

Before the mid to late 1970s "computers" consisted of enormous mainframes that worked primarily with numbers, creating counts and calculations for finance and government. They maintained databases of tax, payroll, inventory, and other business and government related information.

User documentation (computer manuals) at this time consisted of descriptions of systems, emphasizing the control structure (modules) in the program and the interface (menus). Descriptions of interfaces, for example, emphasized menu structures ("Using the Data Entry Screen") instead of the familiar step-by-step of today ("How to Enter Client Data").

As computing technology became much smaller and cheaper during the early 1980s, in the form of the personal computer, many people began using the computing machine. These people, of course, did not have the training in computer science to help them figure out how to use software. So, overall, advancements such as competing operating systems and ever increasing processing speeds kept users dazed and confused.

To respond to the large population of naïve users with highly technical information needs, technical communicators looked to related areas of research and began appropriating ideas to help get computer ideas across to people who knew little about programs they used. For example, the academic literature of this period reflected explorations of cognitive psychology as a feeder discipline for technical communication. Cognitive psychology emphasizes understanding through mental models. Writers and document designers reckoned that they could help users master the hurdles of reconceptualizing computers by using information structuring and presentation methods developed in cognitive psychology. Other disciplines explored for document design solutions included: communications theory, rhetoric, end-user computing, human factors, artificial intelligence, and various information design theories and methods (user-centered design, Information Mapping, Standard Typography for Organizing Proposals).

User documentation matured in the 1990s, absorbing the Internet and the shift from single computers to networked, client/server configurations and functioning much more efficiently as a delivery system for technical information to naïve and intermittent users. Documentation shifted in the early 1990s to both print and online (using Microsoft's *WinHelp* help compiler program) and in the early 2000s is predominantly an online medium with print as a backup media. The influences from feeder disciplines has resulted in a highly flexible information product, delivering information on demand employing a heavy reliance on usability-based development methods, minimal presentation, and single-sourced information architectures.

## VI. PURPOSE AND SCOPE OF USER DOCUMENTATION

Along with the shift of users from geeks to the general public, the purpose and scope of user documentation have evolved from a focus on narrow-minded instruction in program features to a focus on integration of software into workplace surroundings. Document designers can reflect elements of the user's workplace and precise information needs in document products so that the user is not only able to use the program but flourish at work with it. The purpose of documentation, thus, has grown to encompass designing the user's experience—taking a holistic and systematic view of the user's training and information needs—and responding to ensure peak performance. The next section examines some of the methods currently in place for developing software documentation.

## VII. USER DOCUMENTATION DEVELOPMENT METHODOLOGIES

Developing user documents follows methods derived from two groups of researchers: the engineering community and the English studies community. In fact, as the population of computer users changes, matures, grows, and learns, methodologies have also changed. Each methodology below implies a different method of using information about user needs in document development. The choice of method depends highly on the type and style of organization, the organizational culture, and compatibility of publications production and management with product development and its teams.

In many workplaces, the real case aside, you use the methodology you say you use, without making sharp distinctions between one approach and another. The terms "task analysis" and "minimalism" are used fairly loosely.

### A. Task Analysis

Of all the methodologies for developing successful user documentation, the task analysis has gained more followers and resulted in more overall productivity in the workplace than any competing types of document design. Certainly task-oriented manuals fit the information rich workplace of today better than the warmed-over system documentation that companies used to shrink wrap and foist on the bewildered public as "manuals."

Overall the primary difference between task analysis and the default method of document design lies in the definition of the workplace "task" as the primary unit of information. It localizes the solution to the learning problem as one of content. As content, tasks differ from jobs in that a job can consist of many tasks. Tasks become the primary content development method, breaking as they do into discrete stages and easy-to-follow steps. In contrast to manuals using a descriptive orientation (as characterized documentation from the earlier era) tasks represent an organization catering to the use of the program by a professional in the workplace. Because tasks represent the structure of jobs and, ultimately organizations, task orientation in manuals means that manuals reflect significant information about the readers' organization.

### B. Minimalism

The minimalist methodology for organizing user documentation is based on the principles of cognitive psychology and learning theory. Those theories posit that the user brings an understanding or "schema" of thought to a task and the idea of instruction is to make sure that the learner's schema, after the instruction, matches the appropriate one needed to operate the software. For example, a learning of information on financial transfers needs to acquire the "schema" of electronic funds transfer (as in altering computerized records) instead of the schema of physical funds transfer (as in an armored truck). The easiest way to create the appropriate schema in the user's mind, minimalists argue, is to expose the learner to the kinds of tools and expectations offered by a technology (a software program) and allow him or her to make productive connections independently as part of the learning.

The idea behind minimalism underlies other attempts by instructional theorists to create "learning environments" in which situations do the teaching. Clearly, theorists reason, such learning would guarantee an individual experience for each learner and thus enhance the transfer of knowledge into the workplace. Such experiential learning is based, in the case of the primary minimalist researcher today, John Carroll, formerly of IBM, on the following four principles: (1) emphasize learner actions, (2) provide tools relevant to the user's task, (3) support error recognition and recovery (to mirror natural learning), and (4) support different learning purposes: doing, understanding, and locating.

While these principles underlie the minimalist approach, they belie the revolutionary nature of the manuals its practitioners create. Minimalist manuals bury the step-by-step approach, preferring stories ("scenarios") that mimic workplace actions. These manuals also are a lot shorter than the compendium types often produced using task orientation. Shorter manuals with less verbiage allow for open-ended, yet highly focused learning.

### C. Usability

The usability approach to software documentation takes its lead from the rapid design prototyping of its sibling industry: software engineering. Like software engineering, the usability approach starts with an information technology basis, for the simple reason that its main requirement—repeated document testing—can hardly work in the slow-poke world of print. The usability approach seeks the solution to the software learning problem in design issues. Design issues, such as navigation, hyperlinking, and so forth, can be measured and improved based on user characteristics (as in usability based system design). Thus, the emphasis in this approach falls on navigation, structure, search and site mapping, and other technically oriented measures that attempt to achieve a usable information interface.

Drafts, help-system prototypes, and web help pages are documents that lend themselves to the exhaustive prototype testing and constant revision required of the usability testing approach. The usability method is also useful in the design of interface-based help systems such as performance enhancing, embedded help systems.

### D. Information Process Maturity Model

The most successful and fully developed model of document production is, beyond doubt, the Information Product Maturity Model (IPMM) described by Joann Hackos and used in many organizations as the foundation of their quality documentation processes. Used to rank organizations and measure maturity of processes, this model is not so much a methodology as it is a model of how to run your methodology. It encourages the publications managers to develop consistency and measurability in processes.

The key to the IPMM, like the SEI CMM, is to identify key practices as part of a repeatable process, and then convince your team and organization to follow the process while at the same time developing ways to measure and improve it. Once measurements are in place the process just gets better and better, with constant, monitored improvement as the goal. As you can see, such an approach to developing publications would require considerable interaction among diverse groups of people: developers working in parallel or in coordination with publications, personnel who monitor and evaluate one another's processes, managers and supervisors who may need to be convinced of the wisdom of the best practices approach, and so on.

## VIII. TYPES OF USER DOCUMENTS

User documents fall into categories of purpose, and there are three general purposes for reading associated with software use: reading to learn, reading to do, and reading to understand. These three purposes relate roughly to three kinds of documents: tutorials that teach, procedures that guide action, and reference manuals that explain in detail. Other interesting contrasts can be found among these purposes, but for the most part they identify the top-level categories of user documentation for software systems.

### A. Procedural

Procedural documents are based on step-by-step units organized into segments that, ideally, should follow the user's workplace activities. In fact, the procedure gains its strength from the user's interest not in the document, but in using the document to put the program to work. Procedural documents include users guides and online help.

#### 1. User's Guide

The User's Guide is the most popular and useful type of user documentation because it essentially encapsulates all that a user can do with a program between two covers. Pages of users guides are filled with step-by-step procedures that lead or "guide" the reader from one action to another, chronologically to a specified end result. Examples of procedures include:

- How to install DocPro
- Using the *Client Interview* feature to set up accounts

- Printing your work
- Importing files from other word processors
- Obtaining server use statistics by IP number

User's guides are usually organized by categories of software features. Software features often are grouped by most frequently used program modules or menus, sequences of jobs or workplace tasks, or some other method relating to the user's need for intermittent assistance. Information access in user's guides usually involves a table of contents, an index, a list of figures and tables, header and footer information, and chapter indicators (tabs, icons, colored pages, edge bleeds).

### 2. Online Help

Online help began as the online equivalent of the user's guide, but quickly developed into a variety of formats for delivering step-by-step and other information while the user was actually in front of the screen and got, for some reason, stuck. For this reason, online help satisfies the user's need for point of need troubleshooting by providing a variety of access methods to documents.

The typical online help file contains what is known as the three-pane layout: help buttons across the top pane (back, next, etc.), clickable table of contents in the left vertical pane (Getting Started, Setting up Accounts, Adding New Vendors, etc.), and the information (procedure or help topic) in the information window or viewing area under the top pane. Because of their electronic nature, help systems, usually developed using Microsoft's help development program WinHelp, can provide more than one table of contents in the left-hand pane, where typically the access methods are: (1) table of contents, (2) keywords, and (3) index. The user can click on a simulated index tab at the top of the left pane and select from the topics listed to view a specific procedure in the right-hand viewing area.

## B. Reference

Reference documentation represents manuals whose purpose it is to provide background information about all the elements of a functionality of a program (and sometimes its source code). Reference documentation is most familiar to advanced users who know how to operate a program but also need to maintain it and in some cases modify it. To perform these tasks users need descriptive information about the features of the software from a programmer's point of view: What source data files does the program use? What variables does the program identify?, and so on.

Reference documents take their organization from the information they contain: patterning sections of manuals on program modules, alphabetical order, numerical order, and so on. For this reason they are often referred to as "system manuals." Reference documents, whether online or hard copy, rely heavily on indexes and search keywords to help experienced users locate important but small amounts of information quickly.

### 1. System Manual

The system manual is usually a system-oriented document that lists functionalities and program parts and explains what they do and how they do them. Often this form of documentation consists of lists and tables of data telling program modules, overviews of how the systems work, and explanations of program logic.

A typical system manual might have the following contents:

- Overview of System Modules
- Functional Modules
- Program Applets
- List of Data Files
- Database Specifications
- Program Integration Codes

### 2. Troubleshooting Manual

The troubleshooting manual is a hybrid of procedures and tables that help maintenance programmers and administrators configure and set up programs. Readers for troubleshooting manuals are usually reacting to problems with the system and usually seek a procedure or explanation to fill their information needs. Troubleshooting manuals usually follow a problem-by-problem organization, or take the form of a decision tree.

| Problem | Solution |
|---|---|
| System dead | Check plug |
| System frozen | Pull plug |

### 3. FAQ and User-Based Documents

FAQs and user-based documents are a newer breed of document that takes the idea of user functionality to the next level: gaining the user's participation in developing content for manuals. As the numbers of computers grows with a system its users begin to apply it in different situations, and with communication technologies (Internet form and e-mail lists) can communicate their questions, successes, and failures to one another and back to the company. Documents so developed are called user-based documents because their content consists of the discourse among knowledgeable, fellow users. Such documents also channel traffic away from the help desk and support personnel, acting as a filter for support center callers. The disadvantage, of course, is that often the questions are so narrowly focused that they don't, in reality, get asked very frequently. The other disadvantage of user-based documents is that they lead to formulaic FAQs consisting of program features repackaged as "questions," as in: How do I use the Print feature? What are the advantages of upgrading to a more expensive version of this program?

## C. Tutorial

Tutorial documentation is writing that intends to teach a program to a user so that the user can perform the tasks from memory. Tutorials often focus on the main tasks associated with what is referred to as the "typical use" scenario. This scenario consists of the steps a user takes to perform the most common program tasks. Typical uses of a word processor, for example, would include opening a program, entering text, saving the text, and closing the program. The idea behind tutorial user documentation is to encourage adoption of the program's features by the user by teaching him or her the basics and letting job motivations take over. Tutorial user documentation consists of quick start guides, job performance aids, wizards, and demonstrations.

### 1. Quick Start Guide

The Quick Start Guide is a form of tutorial that takes a subset of program functions (usually ones related closely to the typical use scenario for the target reader) and leads the reader through them as an introduction to the program. Quick start guides often will bypass the usual steps as an attempt to satisfy the user who is impatient with plodding through the user's manual. Often they are detached from the main set of documents and presented in brochure, pamphlet, or poster-board media.

### 2. Job Performance Aid

Job performance aids are a kind of tutorial-on-demand for users. Known as JPAs, performance aids make training in the software more usable by making it available when the user really needs it. The design of performance-based documentation systems derives from research on training and aims to provide opportunities for learning to the user while on the job. In its more advanced forms job performance is informed by knowledge management processes within an organization. Because the user can determine which training to implement and when, job performance aids support autonomous learning—a pattern consistent with current viewpoints of the knowledge-managed workplace.

#### a. Wizard

Wizards are types of online tutorial documents that come to the user's rescue when he or she faces a difficult—long, technically complicated—task and needs teaching on the spot. Wizard information can encompass everything from basic skill instruction to complicated, advance procedures (that only a "wizard" would know). Common situations requiring wizards would be setting up an initial web site (requiring much configuration) or importing spreadsheets from other programs. The wizard is a true hybrid form, merging procedural aims (just guide and don't teach) and instructional aims (teach through performance support). Wizards also rely on technological delivery, as the most immediate media for delivering the information is the user's computer, operating system, and browser window.

Structurally the wizard technology interacts with the user following a step-by-step sequence. Along the way the wizard program performs the background work (calling up screens, searching for programs, checking for hardware, and so on) while the user can focus on supplying just the specific information needed in the workplace. For example, if I need to register the person sitting across the desk from me, I want to concentrate on getting the right information from that person—name, address, e-mail, and so on—and not have to worry about finding and opening the registration database, checking and saving the information, and closing the database. From my point of view, all I need to see is a series of sentences explaining what I'm supposed to submit and then confirmation that I have performed the task successfully. To the user the process is almost seamless.

#### b. Embedded Help

Embedded help is a form of performance aid that includes procedural and other information in the interface itself. An example of embedded help is those sentences that appear above data entry fields that tell you whether the field requires case-sensitive text or not. In other examples, you will find more elaborate and systematized presentational methods. The goal of all embedded help is to eliminate the need for a manual or help system by providing screen task basics as a part of the interface. Being part of the interface means that embedded help systems rely on working with software engineers to develop the interface. While embedded help has the advantage of easy access, it also gives little control over the content of the help system to the user. Other examples of embedded help include mouse-over text, popup text, stretch text, and image alternative text.

## SEE ALSO THE FOLLOWING ARTICLES

Database Development Process • End-User Computing Managing • End-User Computing Tools • Ergonomics • Prototyping • System Development Life Cycle • Systems Design • User/System Interface Design

## BIBLIOGRAPHY

Barker, T. T. (2002). *Writing software documentation: A task-oriented approach.* New York: Allyn & Bacon.

Bødker, S. (1991). *Through the interface: A human activity approach to user interface design.* Hillsdale, NJ: Erlbaum.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computing,* Vol. 5, 61–72.

Carroll, J. M. (Ed.) (1998). *Minimalism beyond the Nurnberg funnel.* Cambridge, MA: MIT Press.

Greenbaum, J., and Morten, K., (Eds.) (1991). *Design at work: Cooperative design of computer systems.* Hillsdale, NJ: Erlbaum.

Haas, C. (1996). *Writing technology: Studies on the materiality of literacy.* Hillsdale, NJ: Erlbaum.

Hackos, J. T., and Redish, J. C. (1998). *User and task analysis for interface design.* New York: J. Wiley.

Hutchins, E. (1995). *Cognition in the wild.* Cambridge, MA: MIT Press.

Miller, C. R. (1984). Genre as social action. *Quarterly Journal of Speech,* Vol. 70, 157–178.

Mirel, B. (1998). Applied constructivism' for user documentation. *Journal of Business and Technical Communication,* Vol. 12, No. 1, 7–49.

Nardi, B. A., and O'Day, V. L. (1999). *Information ecologies: Using technology with heart.* Cambridge, MA: MIT Press.

Norman, D. A., and Draper, S. W. (Eds.) (1986). *User centered system design: New perspectives on human-computer interaction.* Hillsdale, NJ: Erlbaum.

Winsor, D. (1999). Genre and activity systems: The role of documentation in maintaining and changing engineering activity systems. *Written Communication,* Vol. 16, No. 2, 200–224.

Zuboff, S. (1988). *In the age of the smart machine: The future of work and power.* New York: Basic Books.