SECTION II
TECHNOLOGY

A. Information Technology
B. Manufacturing and Production Systems
C. Service Systems

II.A
Information Technology

CHAPTER 3
Tools for Building Information Systems

Robert M. Barker
University of Louisville

Brian L. Dos Santos
University of Louisville

Clyde W. Holsapple
University of Kentucky

William P. Wagner
Villanova University

Andrew L. Wright
University of Louisville

1. INTRODUCTION

Spanning the series of revolutionary changes in computer technology and hardware has been an ongoing evolution of software and methodology. This evolution has been motivated by the intent of harnessing raw computing power as a means for increasing human productivity and enhancing organizations' competitiveness. The result is a vast array of computer systems in today's organizations. These systems are being used on a daily basis to assist managers and operating personnel in all kinds of work environments, ranging from manufacturing to service tasks. Two of the most important types of these computer systems are *information systems* and *decision support systems*. In this chapter we will focus on prominent software tools that can be used in building an information system (IS). Decision support systems are dealt with in Chapter 4.

Progress in the development of information systems has been greatly affected by ongoing advances in hardware, tools, and methods. Many information systems in use today would have been economically or technically infeasible a mere decade ago. Continuous decreases in hardware costs per unit of storage and processing power have increased the ubiquity and size of ISs, making them essential components of even modest-sized organizations. Software tools that greatly improve a developer's leverage in implementing information systems have led to ISs of greater sophistication, having more features and better interfaces. For example, software that makes it easy to store, manipulate, and update data (i.e., database management systems) and that allows development of programs in high-level languages (i.e., fourth-generation languages) provides tools that greatly affect the costs of developing information systems and the capabilities that these systems can provide.

In addition to improvements in tools available for IS development, advances have been made in methodologies for developing information systems. In some instances, automated tools support methodology. A structured development method, commonly referred to as the systems life cycle, is routinely used by professional IS developers when building information systems. In addition, various structured development techniques have been devised to support different phases of the systems life cycle. For example, techniques such as data flow diagrams, structure charts, and decision tables have had positive impacts on systems development. One class of software tools, commonly referred to as computer-aided software engineering tools, facilitates the use of these methods and techniques. This chapter examines both the tools that are used in the actual building of information systems (e.g., database management software) and tools that support methodologies for system development.

1.1. The Nature of Information Systems

The core function of an information system is record keeping. Record keeping is able to represent and process information about some domain of interest such as a job shop, an inventory, suppliers of parts for a manufacturing operation, customer accounts, and so forth. Regardless of its application domain, an information system is able to store records describing one or more states of that domain (e.g., current state, historical states, and hypothetical states). The manner in which this descriptive knowledge is organized in a computer is a problem of information representation. The way in which information is represented strongly influences how it can be processed (e.g., updated, retrieved for presentation).

The principal value of an information system being able to represent and process large volumes of descriptive knowledge is that it can give engineers, managers, and operating personnel a way of recording, viewing, and monitoring states of an application domain. An information system provides its users with means for invoking any of a variety of standard reports. When an IS is designed, the developer must determine what reports will be allowed. Each type of report has a particular layout by which it presents information gleaned from the stored data. Thus, in implementing an IS, the developer must be concerned not only with record keeping issues, but with devising means for extracting information represented in the records and mapping it into corresponding report layouts.

Often the predefined reports are produced at predefined times. These may be periodic, as in the case of a weekly sales report or monthly expense report, or they may be event-triggered, as in the case of a stockout report or shipment report. In either case, the reports that an IS can produce may enable users to keep abreast of the current (or past) state of affairs pertaining to some aspect of an organization's operations. Such awareness is essential for a manager's ability to control those operations and can furnish a useful background for decision making. However, the ability of a traditional management information system to support decision making is quite limited because its reports are predefined, tend to be unavailable on the spur of the moment, and are based only on descriptive knowledge (Holsapple and Whinston, 1996).

Except in the most structured cases, the situation a decision maker faces can be very dynamic, with information needs arising unexpectedly and changing more rapidly than ISs can be economically revised to produce new kinds of reports. Even when the newly needed information exists in a set of predefined reports, it may be a needle in the haystack, distributed across several reports, or presented in a way not particularly helpful to the decision maker. It may be incomplete or unfocused or require some value-added processing. In contrast, the ideal of a decision support system (DSS) is to provide focused, complete, fully processed knowledge in the desired presentation format.

Periodic or event-triggered reports from an IS can be useful. However, in dynamic or unstructured decision situations, it is important for a decision maker to control the timing of reporting. With an IS, one must wait for the next periodic or triggered report. Beyond scheduled or event-triggered reporting, a DSS tends to allow desired knowledge to be easily and readily requested on an as-needed basis.

Keeping sufficiently up-to-date records of relevant descriptive knowledge is important as a prerequisite for IS report production. However, other kinds of knowledge enter into decision making, such as procedural knowledge and reasoning (Holsapple 1995). Procedural knowledge involves step-by-step specifications of how to do something. For instance, a solver is an algorithm that typically operates on existing descriptive knowledge to yield new descriptive knowledge in the guise of expectations, beliefs, facts, or model solutions. Reasoning knowledge is concerned with what conclusions are valid under some set of circumstances. For instance, a rule indicates what conclusion to draw when a premise is satisfied; a set of rules can be subjected to inference in order to produce diagnoses or recommendations. Traditional ISs have little concern with storing, updating, and manipulating such knowledge representations as solver libraries or rule sets. DSSs are not only concerned with them, but with how to integrate them as well.

Aside from being able to produce any standard report from a predefined portfolio of report types, an IS may be equipped with a query facility to provide greater flexibility in the kinds of reports that can be produced. After an IS has been implemented, it is not unusual for its users to want reports other than those in the predefined portfolio. The usefulness of these kinds of reports may have been overlooked in the original design of the system, or new reports may be needed because of changing conditions in a user's work environment. A query facility provides a nonprocedural way to state what should be in the desired report, allowing a user to specify what is desired without having to specify how to extract data from records in order to produce the report. An information system equipped with a query facility is a step in the direction of a decision support system. It gives the user a way to address some of the ad hoc knowledge needs that arise in the course of decision making.

1.2. Classes of Information Systems

Having characterized information systems and distinguished them from decision support systems, we can now look at classes of ISs. One approach to classification is based on functional application domains: ISs for finance applications, manufacturing applications, human resource applications, accounting applications, sales applications, and so forth. Another classification approach is to differentiate ISs in terms of the underlying technologies used to build them: file management, database management, data warehouses, Visual Basic, C++, COBOL, HTML, XML, artificial intelligence, and so on. However, it is not uncommon for multiple technologies to be employed in a single IS.

Here, we classify ISs in terms of their intended scopes. Scope can be thought of in two complementary ways: the scope of the records that are kept by the IS and the scope of IS usage. The IS classes form a progression from those of a local scope, to ISs with a functional scope to enterprise-wide ISs to ISs that are transorganizational in scope. Some tools for building information systems are universal in that they can be used for any of these classes. Examples include common programming languages and database management tools. Others are more specialized, being oriented toward a particular class. Representative examples of both universal and specialized tools are presented in this chapter.

1.2.1. Local Information Systems

Local ISs are in wide use today. This is due to such factors as the tremendous rise in computer literacy over the past two decades, the economical availability of increasingly powerful computing devices (desktop and handheld), and the appearance of more convenient, inexpensive development tools. Local ISs are used in both small and large organizations. They are also prominent outside the organizational setting, ranging from electronic calendars and address books to personal finance systems to genealogy systems.

A local IS does record keeping and reporting that relate to a specific task performed by an individual. For instance, a salesperson may have an IS for tracking sales leads. The IS would keep records of the salesperson's current, past, and prospective customers, including contact information, customer characteristics (e.g., size, needs, tastes), history of prior interactions, and schedules of planned interactions. The IS would allow such records to be updated as needed. Reports produced by this IS might include reports showing who to contact at a customer location and how to contact them, a list of all customers having a particular characteristic (e.g., having a need that will be addressed by a new product offering), or a list of prospects in a given locale and a schedule showing who to contact today.

Operation of a local IS tends to be under its user's personal control. The user typically operates the IS directly, assuming responsibility for creating and updating its records and requesting the production of reports. The behavior of a local IS, in terms of what records it can keep and what reports it can produce, may also be under the user's control. This is the case when the IS software is custom-built to meet the particular user's requirements. The user either personally builds the IS software or has it built by a professional IS developer according to his or her specifications. At the opposite extreme, the user may have little control over the behavior of the local IS's software. This is the case when the user obtains off-the-shelf, packaged software that has been designed by a vendor to suit most of the needs of a large class of IS users.

Thus, the user of a local IS has a choice between creating or obtaining custom-built IS software and acquiring ready-made software. Customized construction has the advantage of being tailor-made to suit the user's needs exactly, but it tends to be time-consuming and expensive. Ready-made software is quickly available and relatively inexpensive, but the behavior of an off-the-shelf software package may not match the IS user's needs or desires. Such mismatches range from crippling omissions that render a package unworkable for a particular user to situations where the behavior is adequate in light of cost and time savings.

A middle ground between custom-made and ready-made software is package software that can be configured to behave in a certain way. In principle, this is packaged software that yields a variety of local ISs. The user selects the particular IS behavior that most closely suits his or her needs and, through an interactive process, configures the packaged software to exhibit this behavior. This approach may be somewhat more expensive and time-consuming than strictly ready-made packages, but it also permits some tailoring without the degree of effort required in the custom-made approach.

1.2.2. Functional Information Systems

A functional IS is one that performs record keeping and reporting related to some function of an organization such as production, finance, accounting, marketing, sales, purchasing, logistics, personnel, or research. Such systems include those that give reports on production status, inventory levels, financial positions, accounts (e.g., payable, receivable), sales levels, orders, shipments, fringe benefits, and so forth. These ISs are much broader in scope than local ISs and may be thought of as being more for departmental use than individual use. A functional IS tends to be larger and more complex in terms of records it possesses and processes. Often there are many individuals who use a given functional IS.

Functional ISs are typically administered by IS professionals rather than individual users. It would be even more unusual for an individual user to build his or her own functional IS, as the size and complexity of these systems usually calls for formal system development methodologies. The options for building a functional IS are using customized development, purchasing ready-made software, and purchasing configurable software. Customized development can be performed by an organization's IS department or contracted out to an external developer. In either case, the developer works closely with prospective users of the system, as well as a functional sponsor (e.g., a department head), to ascertain exactly what the system needs to do. The customized development approach is the most flexible for building a system that closely fits users' needs and departmental requirements.

With ready-made software, building a functional IS becomes largely a matter of using the software to enter the system's initial records (and to update them over time). This software allows production of a predefined set of reports from the IS's records. The resultant functional IS can be constructed much more quickly than its custom-made counterpart, but the users will need to conform to the system rather than having a system specifically designed to conform to their needs.

Configurable software forms a middle ground, offering some options in the nature of records to be stored and processed for a given functional application, as well as various reporting options. After a developer sets up a configuration that sufficiently approximates the desired system's behavior, records are loaded into the system for subsequent updating and reporting.

As functional ISs proliferate within a department and across departments, issues of integration and consistency arise. For instance, information held in two functional ISs may be needed in a single report, which cannot be produced (even with an ad hoc query facility) from either IS on its own. Or two functional ISs may store some of the same kinds of records. If updates to these records in the two systems are not made simultaneously, the inconsistencies between their records can lead to inconsistencies between the reports produced by the two systems. One way to try to address integration and consistency issues is to devise additional systems that recognize interrelationships between functional ISs and serve to link them for update or reporting purposes. A second way is to build information systems at an enterprise scale, encompassing multiple functionality in a single IS.

1.2.3. Enterprise Information Systems

Enterprise ISs cross traditional functional boundaries. They keep records pertinent to multiple organizational functions, maintaining them in a consistent fashion and producing both functional and cross-functional reports. The magnitude and complexity of enterprise ISs far exceed those of functional ISs. The creation and operation of enterprise ISs are strictly in the domain of IS professionals. The earliest examples of these systems were custom-made via very large-scale projects. At the time, no off-the-shelf software solutions were available on an enterprise-wide scale. However, over the past decade several vendors have successfully offered configurable off-the-shelf software that functions as a tool for building enterprise ISs. These are known by such names as enterprise resource planning (ERP) and enterprise asset management (EAM) software. For simplicity, the former term is used here.

In some cases, ERP offerings are industry-specific. For instance, one might be oriented toward record keeping and reporting needs of the oil industry, another toward the automotive industry, and yet another toward retailers. ERP offerings also tend to be functionally modular. For instance, an organization may decide to start with implementations of accounting, finance, and production functions. Later, an integral human resources module may be added. In any case, building an enterprise IS with ERP tools is a matter of configuring the record storage mechanisms and configuring the behavior of ERP software to try to fit specific enterprises. This is by no means a trivial task and often causes a redesign of the organization to fit the IS.

The main attraction of an enterprise IS is the potential for consistency of information and integration of information usage. As an example, when a salesperson enters an order, the records describing it are immediately available to others in the enterprise. The factory receives a report of it and starts production. The logistics function receives reports on production progress, allowing it to schedule shipments. Inventory and procurement receive reports of production, leading to replenishment of raw materials and parts. Accounting receives reports on orders, production, and shipments; accounts records are updated accordingly. The enterprise's senior management receives reports allowing it to monitor sales activity, production progress, and inventory levels.

1.2.4. Transorganizational Information Systems

Transorganizational ISs are those that involve information flows to and from entities beyond an enterprise's boundaries. Information for updating records comes directly from customers, suppliers, partners, regulators, and others. Reports are issued directly to these same entities. All of this transorganizational activity is either linked to various internal ISs (local, functional, or enterprise) or designed as an extension of enterprise ISs. Especially notable examples are so-called supply chain management systems and customer relationship management systems. Configurable off-the-shelf software for building each of these is available, in some cases allowing them to be treated as additional ERP modules.

More broadly, transorganizational ISs form a major element of most organizations' electronic commerce initiatives. The Internet and the World Wide Web have been tremendous facilitators of transorganizational ISs, which take two main forms: those that involve accepting and reporting information from and to other businesses (B2B electronic commerce) and those that involve accepting and reporting information from and to consumers (B2C electronic commerce). As companions to transorganizational ISs, many examples of Web-oriented decision support systems exist (Holsapple et al. 2000).

1.3. Overview of Information System Development and Tools

Given an application domain and a class of potential users, we are confronted with the problem of how to create a useful information system. The act of creation spans such activities as analysis, design, and implementation. System analysis is concerned with determining what the potential users want the system to do. System design involves transforming analysis results into a plan for achieving those results. Implementation consists of carrying out the plan, transforming the design into a working information system.

IS implementation may involve software tools such as programming-language compilers and database management software. Or it may involve configuring off-the-shelf software packages. The activity of design can be strongly influenced by what tools are to be used for implementation. Both the analysis and design activities can themselves be supported by tools such as data flow diagrams, data dictionaries, HIPO (hierarchical input, process, output) charts, structure charts, and tools for computer-assisted software engineering (CASE).

In the early days of IS development, the principal software tool used by developers was a programming language with its attendant compiler or interpreter. This tool, together with text-editing software, was used to specify all aspects of an information system's behavior in terms of programs. When executed, these programs governed the overall flow of the information system's actions. They accomplished information storage and processing tasks. They also accomplished user interaction tasks, including the interpretation of user requests and production of reports for users. Section 2 provides an overview of programming, accompanied by highlights of two languages widely used for IS implementation: Visual Basic and C++.

Although programming is a valuable way for developers to specify the flow of control (i.e., what should happen and when) in an information system's behavior, its use for specifying the system's data representation and processing behaviors has steadily diminished in concert with the proliferation of database management tools. Today, database management systems are a cornerstone of information system development. Most popular among the various approaches to database management is the relational, which is described in Section 3. In addition to packaging these as separate tools from conventional (so-called third-generation) languages such as COBOL, C, and FORTRAN, various efforts have been made to integrate database management and programming facilities into single facility. The resultant tools are examples of fourth-generation languages.

Programming and database management tools can be used in implementing ISs in any of the four classes discussed in Section 2. There is great variation in off-the-shelf packages available for IS implementation both within and across the IS classes. Section 2.5 considers prominent Web-based tools used for implementing transorganizational information systems. Section 3 provides a brief overview of database management, which forms the foundation for most information systems today. Section 4 focuses on tools for the class of enterprise ISs. Finally, Section 5 considers ancillary tools that can be valuable in the activity of developing information systems. These are examined in the context of the system development life cycle of analysis, design, implementation, and maintenance. We will focus on tools often used by IS professionals during the analysis and design phases.

2. PROGRAMMING LANGUAGES

2.1. Overview

This section describes how programming languages may be used to build information systems. First, a brief historical review of programming languages helps explain how programming tools have become more powerful and easier to use over the years. We characterize today's modern programming languages, such as C++ and Visual Basic, by considering the advantages and disadvantages of each. Then we review some basic programming concepts by showing some typical examples from these languages. Finally, we consider how program development is affected when the World Wide Web is targeted as a platform. This includes a comparison of Internet programming tools, such as HTML, Java, and CGI scripting.

2.2. Historical Review

In the earliest days of program development, programmers worked directly with the computer's own machine language, using a sequence of binary digits. This process was tedious and error-prone, so programmers quickly started to develop tools to assist in making programming easier for the humans involved. The first innovation allowed programmers to use mnemonic codes instead of actual binary digits. These codes, such as *LOAD, ADD,* and *STORE,* correspond directly to operations in the computer's instruction set but are much easier to remember and use than the sequence of binary digits that the machines required. A programming tool, called an *assembler,* translates mnemonics into machine codes for the computer to execute. This extremely low-level approach to programming is hence referred to as *assembly language* programming. It forces the programmer to think in very small steps because the operations supported by most computers (called *instruction sets*) are very simple tasks, such as reading from a memory location or adding two numbers. A greater problem with this approach is that each computer architecture has its own assembly language, due to differences in each CPU's instruction set. This meant that early programmers had to recreate entire information systems from scratch each time they needed to move a program from one computer to another that had a different architecture.

The next major innovation in computer programming was the introduction of *high-level languages* in the late 1950s. These languages, such as COBOL and FORTRAN, allow a programmer to think in terms of larger, more complex steps than the computer's instructions set allows. In addition, these languages were designed to be portable from machine to machine without regard to the underlying computer architecture. A tool called a *compiler* translates the high-level statements into machine instructions for the computer to execute. The compiler's job is more complex than an assembler's simple task of one-to-one translation. As a result, the machine code that is generated is often less efficient than that produced by a well-trained assembly programmer. For this reason, programming in assembly language is still done today when execution time is absolutely critical. The cost of developing and maintaining high-level code is significantly reduced, however. This is especially important given another trend in computing, towards cheaper and more powerful hardware and more expensive and harder-to-find programmers.

As high-level languages became more prevalent, programmers changed to a new development paradigm, called *structured programming*. This approach, embodied in languages such as Pascal and C from the 1970s and 1980s, emphasizes functional decomposition, that is, breaking large programming tasks into smaller and more manageable blocks, called functions. Data used within a function block is local to that function and may not generally be seen or modified by other blocks of code. This style of programming is better suited to the development of large-scale information systems because different functions can be assigned to different members of a large development team. Each function can be thought of as a black box whose behavior can be described without revealing how the work is actually being done but rather in terms of input and output. This principle of *information hiding* is fundamental to the structured programming approach. Another advantage is *code reuse,* achieved by using the more general-purpose functions developed for one project over again in other systems.

As the information systems being developed became more complex, development with structured programming languages became increasingly difficult. A new paradigm was required to overcome the problem, and it arrived in the late 1980s in the form of *object-oriented programming* (OOP). This continues to play an important role today. In structured programming, functions are the dominant element and data are passed around from one function to another, with each function having a dependency on the structure of the data. If the way the data are represented changes, each function that manipulates the data must, in turn, be modified. In OOP, data are elevated to the same level of importance as the functions. The first principle of object-oriented programming is called *encapsulation*. It involves joining data together with the functions that manipulate that data into an inseparable unit, usually referred to as a *class*. A class is a blueprint for actual *objects* that exist in a program. For example, the class *Clock* would describe how all clocks (objects or instances of the class) in a program will behave. It does not create any clocks; it just describes what one would be like if you built one, much as an architect's blueprint describes what a building would look like if you built one.

The importance of encapsulation is seen when one considers the well-known Year 2000 problem encountered in many programs written with structured languages, such as COBOL. In many of these information systems, literally thousands of functions passed dates around as data, with only two digits reserved for storing the year. When the structure of the data had to be changed to use four digits for the year, each of these functions had to be changed as well. Of course, each function had to be identified as having a dependency on the date. In an OOP language, a single class would exist where the structure of the data representing a date is stored along with the only functions that may manipulate that data directly. Encapsulation, then, makes it easier to isolate the changes required when the structure of data must be modified. It strengthens information hiding by making it difficult to create a data dependency within a function outside a class.

OOP also makes code reuse easier than structured programming allows. In a structured environment, if you need to perform a task that is similar but not identical to an existing function, you must create a new function from scratch. You might copy the code from the original function and use it as a foundation, but the functions will generally be independent. This approach is error-prone and makes long-term program maintenance difficult. In OOP, one may use a principle called *inheritance* to simplify this process. With this approach, an existing class, called the *base class* or *superclass,* is used to create a new class, called the *derived class* or *subclass*. This new derived class is like its parent base class in all respects except what the programmer chooses to make different. Only the differences are programmed in the new class. Those aspects that remain the same need not be developed from scratch or even through copying and pasting the parent's code. This saves time and reduces errors.

The first OOP languages, such as SmallTalk and Eiffel, were rarely used in real world projects, however, and were relegated to university research settings. As is often the case with easing system development and maintenance, program execution speed suffered. With the advent of C++, an OOP hybrid language, however, performance improved dramatically and OOP took off. Soon thereafter, new development tools appeared that simplified development further. *Rapid Application Development* (RAD) tools speed development further by incorporating a more visual development environment for creating graphical user interface (GUI) programs. These tools rely on wizards and code generators to create frameworks based on a programmer's screen layout, which may be easily modified. Examples include Borland's Delphi (a visual OOP language based on Pascal), Borland's C++ Builder (a visual C++ language), and Microsoft's Visual Basic.

2.3. C++

The C++ language was originally developed by Bjarne Stroustrup (Stroustrup 1994) but is now controlled and standardized by the American National Standards Institute (ANSI). It is an extension (literally, an increment) of the C programming language. C is well known for the speed of its compiled executable code, and Stroustrup strove to make C++ into a similarly efficient object-oriented language (see Stroustrup 1994 for a description of the development of the language). Technically, C++ is a hybrid language in that it supports both structured (from C) and object-oriented development. In order to achieve the speed that earlier OOP languages could not, C++ makes some sacrifices to the purity of the OO model. The tradeoffs required, however, helped make OOP popular with real-world developers and sparked a revolution in modern program development.

Like C before it, C++ is a language with a rich abundance of operators and a large standard library of code for a programmer's use. In its current form, C++ includes the Standard Template Library, which offers most of the important data structures and algorithms required for program development, including stacks, queues, vectors, lists, sets, sorts, and searches (Stroustrup 1997). Programs written using C++'s standard components are easily ported from one platform to another. The language lacks, however, a standard library of graphics routines for creating a graphical user interface program under different operating systems. Instead, each compiler vendor tends to offer its own classes and functions for interacting with a specific operating system's windowing routines. For example, Microsoft offers the Microsoft Foundation Classes (MFC) for developing Windows applications with its compiler, while Borland offers its Object Windows Library (OWL) for the same purpose. This makes it difficult to port GUI programs from one vendor's compiler on one operating system to another vendor's compiler on another operating system.

A simple example (Main and Savitch 1997) is shown below that declares a basic *Clock* class. It is an abstraction of a real-world clock used for telling time.

```
class Clock {
 public:
  Clock();
  void set_time(int hour, int minute, bool morning);
  void advance(int minutes);
  int get_hour() const;
  int get_minute() const;
  bool is_morning() const;
 private:
  int hour_24, // Stores the current hour
  minute_24; // Stores the current minute
};
```

This class is typical of the structure of most classes in C++. It is divided into two sections, *public* and *private*. In keeping with information hiding, the data are normally kept private so that others outside of the class may not examine or modify the values stored. The private data are exclusively manipulated by the class functions, normally referred to as *methods* in OOP. The functions are declared in the public section of the class, so that others may use them. For example, one could ask a clock object its minute by using its *get_minute*() function. This might be written as in the following code fragment:

```
Clock c; // Creates an actual object, c
cout << ``The minute is `` << c.get_minute() ;<< endl;
```

Of course, this simple class provides a method for modifying the time stored. The public method *set_time*() is used for this purpose. By forcing the use of this method, the class designer can ensure that the data are manipulated in accordance with appropriate rules. For example, the *set_time*() method would not allow the storage of an illegal time value. This could not be guaranteed if the data were public and directly available for modification. The implementation of this method is shown below:

```
void Clock::set_time(int hour, int minute, bool morning) {
 assert((hour >= 1) && (hour <= 12));
 assert((minute >= 0) && (minute <= 59));
 minute_24 = minute;
 if ((morning)&&(hour == 12))
  hour_24 = 0;
 else if ((!morning) && (hour < 12))
  hour_24 = hour + 12;
 else
  hour_24 = hour;}
```

Note how the values passed to the method are tested via an *assert*() statement before they are used. Only if the assertion proves true are the values used; otherwise a runtime error message is produced.

To show an example of inheritance, consider extending the *Clock* class to create a new class, *CuckooClock*. The only difference is that this type of clock has a bird that chirps on the hour. In all other respects it is identical to a regular clock.

```
class CuckooClock : public Clock {
 public:
  bool is_cuckooing( ) const;
};

bool CuckooClock::is_cuckooing( ) const {
 return (get_minute( ) == 0);}
```

Note, we declare the new *CuckooClock* class in terms of the existing *Clock* class. The only methods we need to describe and implement are those that are new to or different from the base class. Here there is only one new function, called *is_cuckooing*(), which returns true on the hour. It is important to understand that *CuckooClock* is a *Clock*. In important respects, an instance of *CuckooClock* can do anything that a *Clock* object can do. In fact, the compiler will allow us to send a *CuckooClock* object anywhere a *Clock* object is expected. For example, we might have a function written to compare two *Clock* objects to see if one is "equal to" the other.

```
bool operator ==(const Clock& c1, const Clock& c2) {
 return ((c1.get_hour() == c2.get_hour()) &&
 (c1.get_minute() == c2.get_minute()) &&
 (c1.is_morning() == c2.is_morning())); }
```

We may confidently send a *CuckooClock* to this function even though it is written explicitly to expect *Clock* objects. Because of inheritance, we do not need to write a different version of the function for each class derived from *Clock—CuckooClock* is a *Clock*. This simplifies things considerably for the programmer. Another interesting note about this code segment is that C++ allows us to provide definitions for most of its built-in operators in the context of a class. This is referred to as *operator overloading,* and C++ is rare among programming languages in allowing this kind of access to operators. Here we define a meaning for the equality operator (==) for *Clock* objects.

In summary, C++ is a powerful and rich object-oriented programming language. Although widely recognized as difficult to work with, it offers efficient execution times to programmers that can master its ways. If you want portable code, you must stick to creating console applications. If this is not a concern, modern compilers assist in creating GUI applications through wizards, such as in Microsoft's Visual C++. This is still more difficult to accomplish using C++ than a RAD tool such as Visual Basic, which will be discussed next.

2.4. Visual Basic

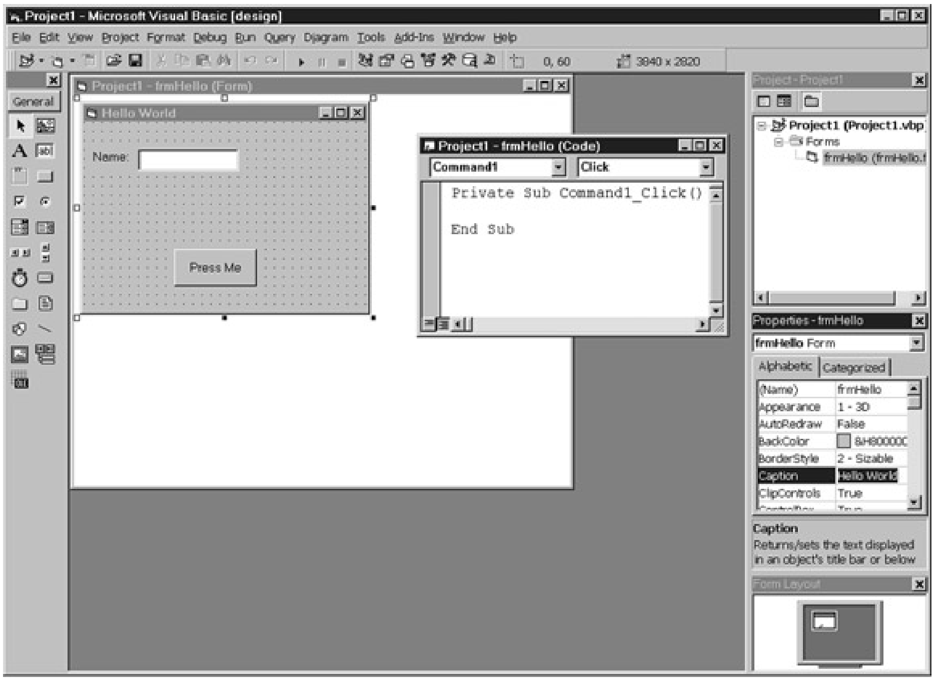
Visual Basic (VB) is a programming language and development tool from Microsoft designed primarily for rapidly creating graphical user interface applications for Microsoft's Windows operating system. First introduced in 1991, the language is an extension of BASIC, the Beginners' All-Purpose Symbolic Instruction Code (Eliason and Malarkey, 1999). BASIC has been in use since John Kemeny and Thomas Kurta introduced it in 1965 (Brookshear 1999). Over the years, Visual Basic has evolved more and more towards an object-oriented programming model. In its current version, 6.0, VB provides programmers with the ability to create their own classes using encapsulation, but it does not yet support inheritance. It simplifies the creation of Windows applications by making the enormously complex Windows Application Program Interface (consisting of over 800 functions) available through easy-to-use objects such as forms, labels, command buttons, and menus (Eliason and Malarkey 1999).

As its name suggests, Visual Basic is a highly *visual* tool. This implies not only that the development environment is GUI-based but also that the tool allows you to design a program's user interface by placing components directly onto windows and forms. This significantly reduces development time. Figure 1 provides a snapshot of the Visual Basic development environment.

Another feature that makes Visual Basic into a Rapid Application Development tool is that it supports both interpreted and compiled execution. When VB is used in *interpreted* mode, the tool allows the programmer to quickly see the effects of their code changes without a lengthy compilation directly to machine code. Of course, run-time performance is better when the code is actually compiled, and this can be done before distributing the application.

In Visual Basic, objects encapsulate properties, methods, and events. *Properties* are an object's data, such as a label's caption or a form's background color. *Methods* are an object's functions, as in C++ and other OOP languages. *Events* are typically user-initiated actions, such as clicking a form's button with the mouse or making a selection from a menu, that require a response from the object. Figure 1 shows some of the properties of the highlighted command button object, *Command1*. Here the button's text is set to "Press Me" via the *Caption* property. In the code window for the form, the *Click* event's code is displayed for this command button. This is where you would add the code to respond when the user clicks on the form's button. The tool only provides the outline for the code, as seen here. The programmer must add actual VB statements to perform the required action. Visual Basic does provide many wizards, however, to assist in creating different kinds of applications, including forms that are connected back to a database. These wizards can significantly reduce development time and improve reliability.

The fundamental objects that are important to understand in Visual Basic development are *forms* and *controls*. A form serves as a container for controls. It is the "canvas" upon which the programmer



```
Private Sub Command1_Click()

End Sub
```

(Name)	frmHello
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H80000C
BorderStyle	2 - Sizable
Caption	Hello World
ClipControls	True

Caption
Returns/sets the text displayed in an object's title bar or below

Figure 1 The Visual Basic Development Environment.

"paints" the application's user interface. This is accomplished by placing controls onto the form. *Controls* are primarily GUI components such as command buttons, labels, text boxes, and images. In Figure 1, the standard toolbox of controls is docked on the far left of the screen. These controls are what the user interacts with when using the application. Each control has appropriate properties and events associated with it that affect its appearance and behavior. For example, consider the form shown in Figure 2.

This form shows the ease with which you can create a database-driven application. There are three controls on this form. At the bottom of the form is a *data* control. Its properties allow the programmer to specify a database and table to connect to. In this example, it is attached to a simple customer database. At the top are two *textbox* controls, used for displaying and editing textual information. In this case, these controls are *bound* to the data control, indicating that their text will be coming from the associated database table. The data control allows the user to step through each record in the table by using the left and right arrows. Each time the user advances to another record from the table, the bound controls update the user's text to display that customer's name and address. In the code fragment below, you can see some of the important properties of these controls.

```
Begin VB.Data Data1
 Caption = ``Customer''
 Connect = ``Access''
 DatabaseName = ``C:\VB Example\custdata''
 DefaultCursorType= 0 `DefaultCursor
 DefaultType = 2 `UseODBC
 Exclusive = 0 `False
 ReadOnly = 0 `False
 RecordsetType = 1 `Dynaset
 RecordSource = ``Customer''
End

Begin VB.TextBox Text1
 DataField = ``customer name''
 DataSource = ``Data1''
End

Begin VB.TextBox Text2
 DataField = ``street address''
 DataSource = ``Data1''
End
```

In the data control object, *Data1,* there are several properties to consider. The *Connect* property specifies the type of database being attached, here an Access database. The *DatabaseName* property specifies the name and location of the database. *RecordSource* specifies the table or stored query that will be used by the data control. In this example, we are connecting to the *Customer* table of the example database. For the bound text controls, the programmer need only specify the name of the data control and the field to be used for the text to be displayed. The properties that need to be set for this are *DataSource* and *DataField,* respectively.

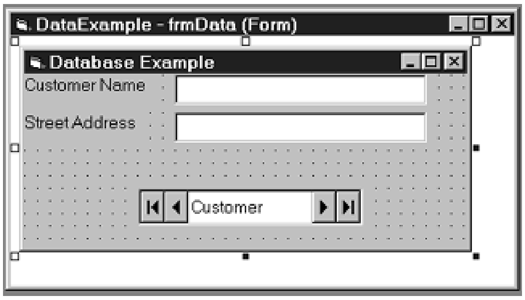


Figure 2 Database Example.

As these examples demonstrate, Visual Basic makes it relatively easy to create complex GUI applications that run exclusively under the Windows operating system. When these programs are compiled into machine code, the performance of these applications is quite acceptable, although not yet as fast as typical C++ programs. The object-oriented model makes development easier, especially since most of the properties can be set visually, with the tool itself writing the necessary code. Unfortunately, VB still does not support inheritance, which limits a programmer's ability to reuse code. Recent additions in the language, however, make it easier to target the Internet as an application platform. By using *Active X controls,* which run under Windows and within Microsoft's Internet Explorer (IE) web browser, and *VBScript* (a subset of VB that runs in IE), you can create applications that may be ported to the World Wide Web. Your choice of controls is somewhat limited and the user must have Microsoft's own web browser, but in a corporate Intranet environment (where the use of IE can be ensured) this might be feasible. For other situations a more flexible solution is required. The next section will explore some of the tools for achieving this.

2.5. Web-Based Programming

Developing applications where the user interface appears in a web browser is an important new skill for programmers. The tools that enable a programmer to accomplish this type of development include HTML, Java, CGI, Perl, ColdFusion, ASP, etc.—a veritable cornucopia of new acronyms. This section explains these terms and shows how these new tools may be used to create web pages that behave more like traditional software applications.

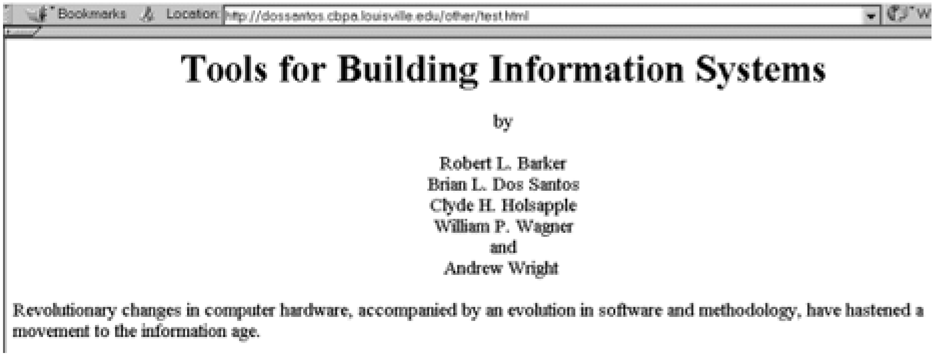
2.5.1. HTML

The HyperText Markup Language (HTML) is *the* current language of the World Wide Web. HTML is a markup language, not a programming language. Very simply, a document is "marked up" to define its appearance. This involves placing markup tags (or commands) around text and pictures, sound, and video in a document. The general syntax for HTML tags is:

```
<Tagname [options]>your text here</ tagname>
```

These tags indicate how to display portions of the document. Opening tags, along with available options, are enclosed in < and > and closing tags include the / before the tagname. In some instances both opening and closing tags are required to indicate what parts of the documents the tags should affect. In other instances no closing tag is required. The contents of the file (including HTML tags and web page contents) that generates the simple web page shown in Figure 3 are as follows:

```
<html>
<head>
<title>Tools for Building Information Systems</title>
</head>
<body>
<center><h1>Tools for Building Information Systems</h1>
by<p>
Robert L. Barker<br>Brian L. Dos Santos<br>Clyde H.
```


Bookmarks
Location: http://dossantos.cbpa.louisville.edu/other/test.html

Tools for Building Information Systems

by

Robert L. Barker
Brian L. Dos Santos
Clyde H. Holsapple
William P. Wagner
and
Andrew Wright

Revolutionary changes in computer hardware, accompanied by an evolution in software and methodology, have hastened a movement to the information age.

Figure 3 Web Page and HTML Tags.

```
Holsapple<br>William P. Wagner<bR>and<br>Andrew Wright
</center><p>
Revolutionary changes in computer hardware, accompanied by
an evolution in software and methodology, have hastened a
movement to the information age.
</body>
</html>
```

HTML documents may then be processed and displayed using a browser that runs on any number of computer platforms, from UNIX workstations down to cellular telephones. These documents are simple text files that can be created with WYSIWYG tools (such as Microsoft's FrontPage or Netscape's Composer) or even a simple text editor such as the Windows Notepad. HTML relies on *tags,* keywords enclosed in angle brackets, like `<h1>`, `<b>`, `<li>`, etc., to specify what type of content or formatting is intended. HTML standards are controlled by an organization called the World Wide Web Consortium (W3C), and their website (http://www.w3.org/) has the most accurate and up-to-date information about HTML.

The latest version of HTML is 4.0 (http://www.w3.org/MarkUp/), and several of today's browsers support most of its features. In addition, the leading producers of web browsers, Microsoft and Netscape, frequently include support for proprietary features not specified in the W3C's version of HTML. These extensions can make it difficult to create extremely complicated documents that will work on both browsers. This is especially true for *dynamic* HTML documents, or documents whose contents change even after the page is loaded into a browser (Castro, 1999). HTML 4.0 includes support for Cascading Style Sheets (http://www.w3.org/Style/), which gives page designers easier control of large and complicated websites. HTML is essentially a system for formatting documents. It includes both structuring and formatting tags, which makes it difficult to maintain complex websites. The introduction of cascading style sheets allows a designer to separate structuring and formatting tags. Formatting tags are maintained in cascading style sheets so that HTML is used only for structuring documents. Style sheets are very powerful but not as easy to use as basic HTML. The future, however, lies in Extensible Markup Language (XML) (http://www.w3.org/XML/). XML is designed to allow groups to create their own markup languages, like HTML, specifically suited to their needs (Castro, 1999). It is especially important for its ability to put structured data into a text format. This might make it easy to view and edit a spreadsheet or database via the Web, for example. In fact, the W3C is rewriting HTML (as XHTML) and its Cascading Style Sheets (as XSL) using XML (see http://www.w3.org/TR/xhtml1/ and http://www.w3.org/Style/XSL/).

HTML, by itself, does not provide the capabilities required for electronic commerce. For e-commerce, it is necessary to develop applications that are interactive in nature, permitting visitors to dynamically obtain information they need (e.g., search a product catalog) or complete some task (e.g., submit an order). Such capabilities typically require that user-provided data be processed in real time and require interaction with databases. This type of processing may be accomplished via programming languages such as Perl, Visual Basic, and C++. Recently, several tools have emerged that reduce the need for traditional programming while essentially achieving the same results (e.g., ColdFusion and Tango). These capabilities are often achieved via the Common Gateway Interface (CGI).

2.5.2. CGI

To create truly powerful websites, a programmer must be able to access current data, typically from a corporate database. For example, commerce sites need to be able to tell a user whether a product is in stock and record how many items a customer would like to purchase. This cannot be done with static HTML files. Instead, processing is required on the server-side before the contents of a web page are delivered to a user's web browser. One widely used approach is the Common Gateway Interface (CGI). CGI specifies a common method for allowing Web pages to communicate with programs running on a Web server.

A typical CGI transaction begins when a user submits a form that he or she has filled out on a web page. The user may be searching for a book by a certain author at an online bookstore and have just entered the author's name on a search page. The HTML used for displaying the form also specifies where to send the data when the user clicks on the submit button. In this case, the data is sent to a CGI program for processing. Now the bookstore's CGI program searches its database of authors and creates a list of books. In order to return this information to the user, the CGI program must create a response in HTML. Once the HTML is generated, control may once again be returned to the web server to deliver the dynamically created web page. CGI programs may be written in nearly any programming language supported by the operating system but are often written in specialized scripting languages like Perl.

CGI provides a flexible, but inefficient mechanism for creating dynamic responses to Web queries. The major problem is that the web server must start a new program for each request, and that incurs significant overhead processing costs. Several more efficient alternatives are available, such as writing programs directly to the Web server's application program interface. These server APIs, such as Microsoft's ISAPI and Netscape's NSAPI, allow more efficient transactions by reducing overhead through the use of dynamic link libraries, integrated databases, and so on. Writing programs with the server APIs is not always an easy task, however. Other alternatives include using products like Allaire's ColdFusion or Microsoft's Active Server Pages (ASP).

2.5.3. *Java*

Java is a programming language from Sun Microsystems that is similar to C/C++ in syntax and may be used for general-purpose applications and, more significantly, for embedding programs in web pages (Hall 1998). Although Java is an extremely young language, it has gained widespread acceptance because of its powerful and easy-to-use features. It is not yet a mature product, however, and experiences some stability problems as the language continues to evolve.

Java has a lot in common with C++, so much so that it is often referred to as "C++ Lite" by experienced programmers. It is an object-oriented programming language that was built from the ground up, unlike the hybrid C++. It is designed to be a cross-platform development tool; in other words, the code is easy to port from one computer architecture and operating system to another. It does a better job of this than C++ does by including a standard set of classes for creating graphical user interfaces. This allows a GUI application designed on a Macintosh to be run under Windows or Linux, for example. Extremely complex interfaces may still require tweaking to achieve complete interchangeability of the interface, but new additions to the language (the Swing classes) make this less necessary. Java is widely considered a simpler language to use than C++ because it forgoes some of the more troublesome C++ (though powerful) features. For example, Java does not explicitly use pointers and has automatic memory management, areas where C++ programmers often introduce bugs into their programs. This makes Java much easier to work with and somewhat less error-prone. Java only supports *single inheritance,* which means that a derived class may only have a single parent base class. C++ allows the use of the more powerful form, *multiple inheritance,* which allows several parents for a derived class. Java does not support *templates,* which allow C++ programmers to create functions and classes that work for many different types of data without needing to be implemented separately for each. Java also does not support operator overloading. Like C++, however, Java does have a rich and powerful set of standard libraries. Java's libraries even include components for network programming and database access (through JDBC), which standard C++ lacks. Even with all these improvements to C++, Java would be a footnote in the annals of programming if it were not for its web features.

Java allows the creation of specialized applications called *applets,* which are designed to be run within a web browser. Most modern web browsers support Java in some form, and Sun offers plug-ins that provides the most up-to-date language features. Early applets focused on creating more dynamic web pages through animations and other graphical components easily created in Java but hard to reproduce in pure HTML. Today, applets are used for delivering entire software applications via the Web. This has many advantages over a traditional development and distribution model. For example, programmers do not have to worry about the end-user platform because applets run in a browser. Users don't have to worry about applying patches and bug fixes because they always get the latest version from the Web. Security is an obvious concern for applets because it might be possible to write a malicious Java applet and place it on a web page for unsuspecting users. Luckily, the Java community has created a robust and ever-improving security model to prevent these situations. For example, web browsers generally restrict applets from accessing memory locations outside their own programs and prevent writing to the user's hard drive. For trusted applets, such as those being run from a company's Intranet, the security restrictions can be relaxed by the end user to allow more powerful functionality to be included.

2.5.4. *ColdFusion*

ColdFusion is a tool for easily creating web pages that connect to a database. Developed by Allaire (http://www.allaire.com), ColdFusion is being used in thousands of Web applications by leading companies, including Reebok, DHL, Casio, and Siemens. The ColdFusion Markup Language (CFML) allows a user to create Web applications that are interactive and interface with databases. The syntax of CFML is similar to HTML, which is what makes it relatively easy to use; it makes ColdFusion appealing to Web designers who do not have a background in programming. ColdFusion encapsulates in a single tag what might take ten or a hundred lines of code in a CGI or ASP program. This allows for more rapid development of applications than competing methods provide. However, if a tag does not do exactly what you need it to, then you must either change your application or resort to a programmatic approach. Although ColdFusion is not a programming language per se, knowledge of fundamental programming concepts and SQL is essential for sophisticated applications. CFML allows a user to display output based on the results of a query to a database and update databases based on user input. In order to use CFML, a ColdFusion Application Server (CFAS) must be installed. CFAS works in conjunction with a web server. The CFAS can do much more than query/update databases; it interfaces with many different Internet services to provide the functionality that a website designer desires. Figure 4 shows how a ColdFusion Application Server interacts with a web server, databases, and other services it provides.

The following sequence of actions illustrates how ColdFusion functions to produce dynamic results.

1. A user submits a form via a browser. The form is designed to be processed by a file containing CFML code. This file must have a ‘‘cfm’’ extension.
2. The web server recognizes the cfm extension and hands the file to the CFAS for processing.
3. The CFAS executes the instructions per the CFML code. This typically results in information from a database to be included in the web page. In addition, it may interact with other Internet services, such as other web servers, e-mail, etc.
4. The web page generated by the CFAS is then returned to the web server.
5. The web server sends this page to the browser.

Allaire offers versions that run under Microsoft Windows NT and the Unix-based operating systems Solaris and HP-UX. ColdFusion programs interface with many different Web server APIs to provide greater efficiency than traditional CGI programs.

2.5.5. ASP

One of Microsoft’s methods for creating more dynamic web pages is through its web server’s support of Active Server Pages. ASP lets programmers use the VBScript programming language on the server-side to create HTML content on-the-fly. Because ASP is built into the Web server, it is more efficient than other CGI programs. It offers greater flexibility and control than a product like ColdFusion but requires greater programming knowledge to achieve results.

3. DATABASE MANAGEMENT TOOLS

3.1. DBM Concepts

One of the most important functions that an information system must perform is data management (i.e., record keeping). Prior to the advent of true database software, computer data management was characterized by excessive data redundancy (or duplication), large data dependence (or coupling),

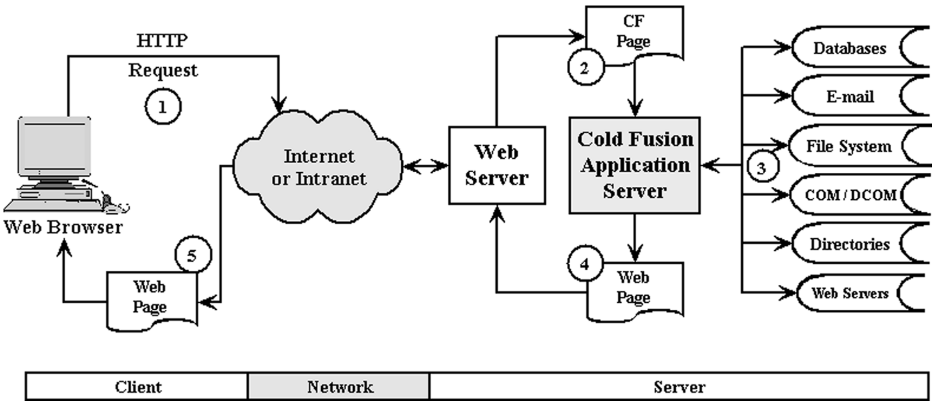


Figure 4 ColdFusion Application Server.

and diffused data ownership (McLeod 1998). In early data processing systems, the data files were often created with "little thought as to how those files affected other systems. Perhaps much, or even all, of the new data in a new file was already contained in an existing file" (McLeod 1998). This resulted in significant data redundancy and several associated problems. Obviously, data duplication results in wasted space, but worse than this are synchronization problems. "Multiple copies of data were often updated on different frequencies. One file would be updated daily, another weekly, and a third monthly. Reports based on one file could conflict with reports based on another, with the user being unaware of the differences" (McLeod 1998). These early systems also exhibited a high degree of dependence between the data specifications and the programs written to manipulate the data. Every time the representation of the data or the file storage format changed, all of the programs that depended on that data would also need to be changed. Other problems resulted from a lack of agreement on data ownership and standardization. "This situation of the 1950s and early 1960s may have been the reason why many of the first management information systems failed in the late 1960s. Information specialists appeared unable to deliver systems that provided consistent, accurate, and reliable information" (McLeod 1998).

Database systems sought to address these issues by separating the *logical* organization of the data from the *physical* organization. The logical organization is how the *user* of the system might view the data. The physical organization, however, is how the *computer* views the data. A database, then, may be defined as "an integrated collection of computer data, organized and stored in a manner that facilitates easy retrieval" by the user (McLeod 1998). It seeks to minimize data redundancy and data dependence. A database is often considered to consist of a hierarchy of data: files, records, and fields. A *field* is the smallest organized element of data, consisting of a group of characters that has a specific meaning. For example, a person's last name and telephone number would be common fields. Related fields are collected and stored in a structure referred to as a *record*. A specific person's record might consist of his or her name, address, telephone number, and so on (all values associated with that individual). Related records are grouped into a *file,* such as a file of employees or customers. Keep in mind that the physical representation of the database will be shielded from the user and should not limit the user's ability retrieve information from the system.

A database management system (DBMS) is a "collection of programs that manage the database structure and controls access to the data stored in the database. The DBMS makes it possible to share the data in the database among multiple applications or users" (Rob and Coronel 1997). In a sense, it sits between the user and the data, translating the users' requests into the necessary program code needed to accomplish the desired tasks. General Electric's Integrated Data Store (IDS) system, introduced in 1964, is considered by most to be the first generalized DBMS (Elmasri and Navathe 2000).

Database systems are often characterized by their method of data organization. Several popular models have been proposed over the years. Conceptual models focus on the logical nature of the data organization, while implementation models focus on how the data are represented in the database itself. Popular conceptual models include the entity-relationship (E-R) model and the object-oriented model discussed below. These models typically describe relationships or associations among data as one-to-one, one-to-many, or many-to-many. For example, in most organizations an employee is associated with one department only but a department may have several employees. This is a one-to-many relationship between department and employee (*one* department has *many* employees). But the relationship between employee and skills would be many-to-many, as a single employee might possess several skills and a single skill might be possessed by several employees (*many* employees have *many* skills).

Implementation models include the hierarchical model, network model, and relational model. Each model has advantages and disadvantages that made it popular in its day. The relational model is easily understood and has gained widespread acceptance and use. It is currently the dominant model in use by today's database management systems.

3.1.1. Relational DBMS

The relational database model, introduced by E. F. Cod in 1970, is considered a major advance for users and database designers alike. At the time, however, many considered the model impractical because its "conceptual simplicity was bought at the expense of computer overhead" (Rob and Coronel 1997). As processing power has increased, the cost of implementing the relational model has dropped rapidly. Now you will find relational DBMS implementations from mainframes down to microcomputers in products like DB2, Oracle, and even Microsoft's Access.

To the user, the relational database consists of a collection of *tables* (or *relations* in the formal language of the model). Each table may be thought of as a matrix of values divided into rows and columns. Each row represents a record and is referred to as a *tuple* in the model. Each column is a field and is called an *attribute* in the model. Below are some simple tables for Employee and Department.

Employee	EmployeeID	EmpName	EmpPhone	DepartmentID
	1234	Jane Smith	5025551234	11
	2111	Robert Adams	8125551212	20
	2525	Karen Johnson	6065553333	11
	3222	Alan Shreve	5025558521	32
	3536	Mary Roberts	8125554131	20

Department	DepartmentID	DeptName
	11	Marketing
	20	Engineering
	32	Accounting

Pay special attention to how relationships between tables are established by sharing a common field value. In this example, there is a one-to-many relationship between the Employee and Department tables established by the common *DepartmentID* field. Here, Robert Adams and Mary Roberts are in the Engineering department (*DepartmentID* = 20). The tables represented here are independent of one another yet easily connected in this manner. It is also important to recognize that the table structure is only a logical structure. How the relational DBMS chooses to represent the data physically is of little concern to the user or database designer. They are shielded from these details by the DBMS.

3.1.2. Processing

One of the main reasons for the relational model's wide acceptance and use is its powerful and flexible ad hoc query capability. Most relational DBMS products on the market today use the same query language, Structured Query Language (SQL). SQL is considered a fourth-generation language (4GL) that "allows the user to specify *what* must be done *without* specifying *how* it must be done" (Rob and Coronel 1997). The relational DBMS then translates the SQL request into whatever program actions are needed to fulfill the request. This means far less programming for the user than with other database models or earlier file systems.

SQL statements exist for defining the structure of the database as well as manipulating its data. You may create and drop tables from the database, although many people use tools provided by the DBMS software itself for these tasks. SQL may also be used to insert, update, and delete data from existing tables. The most common task performed by users, however, is data retrieval. For this, the *SELECT* statement is used. The typical syntax for *SELECT* is as follows:

```
SELECT <field(s)>
FROM <table(s)>
WHERE <conditions>;
```

For example, to produce a list of employee names and telephone numbers from the Engineering department using the tables introduced earlier, you would write:

```
SELECT EmpName, EmpPhone
FROM Employee
WHERE DepartmentID = 20;
```

Of course, you may use the typical complement of logical operators, such as *AND, OR,* and *NOT,* to control the information returned by the query.

The *SELECT* statement may also be used to join data from different tables based on their common field values. For example, in order to produce a list of employee names and their departments, you would write:

```
SELECT Employee.EmpName, Department.DeptName
FROM Employee, Department
WHERE Employee.DepartmentID = Department.DepartmentID;
```

In this example, the rows returned by the query are established by matching the common values of *DepartmentID* stored in both the *Employee* and *Department* tables. These joins may be performed across several tables at a time, creating meaningful reports without requiring excessive data duplication.

It is also possible to sort the results returned by a query by adding the *ORDER BY* clause to the *SELECT* statement. For example, to create an alphabetical listing of department names and IDs, you would write:

```
SELECT DeptName, DepartmentID
FROM Department
ORDER BY DeptName;
```

These examples demonstrate the ease with which SQL may be used to perform meaningful ad hoc queries of a relational database. Of course, SQL supports many more operators and statements than those shown here, but these simple examples give a hint of what is possible with the language.

3.2. Object-Oriented Databases

The evolution of database models (from hierarchical to network to relational) is driven by the need to represent and manipulate increasingly complex real-world data (Rob and Coronel 1997). Today's systems must deal with more complex applications that interact with multimedia data. The relational approach now faces a challenge from new systems based on an object-oriented data model (OODM). Just as OO concepts have influenced programming languages, so too are they gaining in popularity with database researchers and vendors.

There is not yet a uniformly accepted definition of what an OODM should consist of. Different systems support different aspects of object orientation. This section will explore some of the similar characteristics shared by most of these new systems. Rob and Coronel (1997) suggest that, at the very least, an OODM

1. Must support the representation of *complex objects*
2. Must be *extensible,* or capable of defining new data types and operations
3. Must support *encapsulation* (as described in Section 2)
4. Must support *inheritance* (as described in Section 2)
5. Must support the concept of *object identity,* which is similar to the notion of a primary key from the relational model that uniquely identifies an object. This object identity is used to relate objects and thus does not need to use *JOIN*s for this purpose.

Consider an example involving data about a *Person* (from Rob and Coronel 1997). In a typical relational system, *Person* might become a table with fields for storing name, address, date of birth, and so on as strings. In an OODM, *Person* would be a class, describing how all instances of the class (*Person* objects) will be represented. Here, additional classes might be used to represent name, address, date of birth, and so on. The *Address* class might store city, state, street, and so on as separate attributes. *Age* might be an attribute of a *Person* object but whose value is controlled by a method, or function of the class. This inclusion of methods makes objects an *active* component in the system. In traditional systems, data are considered *passive* components waiting to be acted upon. In an OODM, we can use inheritance to create specialized classes based on the common characteristics of an existing class, such as the *Person* class. For example, we could create a new class called *Employee* by declaring *Person* as its superclass. An *Employee* object might have additional attributes of salary and social security number. Because of inheritance, an *Employee* is a *Person* and, hence, would also gain the attributes that all *Person* objects share—address, date of birth, name, and so on. *Employee* could be specialized even further to create categories of employees, such as manager, secretary, cashier, and the like, each with its own additional data and methods for manipulating that data.

Rob and Coronel (1997) summarize the advantages and disadvantages of object-oriented database systems well. On the positive side, they state that OO systems allow the inclusion of more semantic information than a traditional database, providing a more natural representation of real-world objects. OO systems are better at representing complex data, such as required in multimedia systems, making them especially popular in CAD/CAM applications. They can provide dramatic performance improvements over relational systems under certain circumstances. Faster application development time can be achieved through the use of inheritance and reuse of classes. These OO systems are not without problems, however. Most of them are based on immature technology, which can lead to product instability. This is exacerbated by the lack of standards and an agreed-upon theoretical foundation. OO database systems especially lack a standard ad hoc query language (like SQL for relational systems). In addition, the initial learning curve is steep, particularly for individuals who are well accustomed to the relational model. On the whole, it seems clear that object-oriented databases are worth consideration. As standards develop over time and people become more familiar with object-oriented systems in general, OO databases could prove to be the successor to the relational model.

3.3. Data Warehouses

One of the major changes in the information systems area within the past decade is a recognition that there are two fundamentally different types of information systems in all enterprises: *operational systems* and *informational systems*. Operational systems are just what their name implies: they are the systems that help us run the enterprise day-to-day. These are the backbone systems of any enterprise: the order entry, inventory, manufacturing, payroll, and accounting systems. Because of their importance to the organization, operational systems were almost always the first parts of the enterprise to be computerized. Over the years, these operational systems have been enhanced and maintained to the point that they are completely integrated into the organization. Indeed, most large enterprises around the world today couldn't operate without their operational systems and the data that these systems maintain. Operational data normally is current (the present) and is focused on an area within the enterprise.

On the other hand, other functions go on within the enterprise that have to do with planning, forecasting, and managing the organization. These functions are also critical to the survival of the organization, especially in our fast-paced world. Functions like marketing planning, engineering planning, and financial analysis also require information systems to support them. But these functions are different from operational ones, and the types of systems and information required are also different. The information needed to aid such major decisions that will affect how the enterprise will operate, now and in the future, typically is obtained by analyzing operational data over a long time period and covering many different areas within an enterprise.

A data warehouse is a subject-oriented, integrated, time-variant, nonvolatile collection of data to support management decision making. The data in a data warehouse are stored in a manner that will support the information and analytical processing needs of the organization over a long historical time period. They are typically physically separated from the operational databases that an organization must maintain and update to function effectively on a day-to-day basis (Singh 1998; Barquín and Edelstein 1997).

There are many data flows from the operational databases to the data warehouse and from the data warehouse to users. Data must flow from the enterprise's legacy systems to a more compact form, then to the data warehouse storage, and finally out to consumers. The flows necessary to manage the warehouse itself (metaflow) must also be considered (Hackathorn 1995). For each flow, data warehouses require tools to make these processes function effectively (Mattison 1996).

The tools for developing data warehouses can be grouped into three categories, based on their activities: acquisition tools (for inflow), storage tools (for upflow and downflow), and access products (for outflow) (Mattison 1996). Acquisition tools are necessary to perform tasks such as modeling, designing, and populating data warehouses. These tools extract data from various sources (operational databases and external sources) and transform it (i.e., condition it, clean it up, and denormalize it) to make the data usable in the data warehouse. They also establish the metadata, where information about the data in the warehouse is stored (Francett 1994).

Storage is usually managed by relational DBMSs and other special tools in ways that enable the use of data for effective decision support (Mattison 1996). Access to warehouse contents is provided by data mining tools using such methods as neural networks and data discovery mechanisms (Mattison 1996), plus multidimensional analysis software that supports end users in accessing and analyzing the data in the warehouse in various ways (Francett 1994). Data mining is the process of making a discovery from large amounts of detailed data (Barry 1995) by drilling down through detailed data in the warehouse. Data mining tools sift through large volumes of data to find patterns or similarities in the data. Data mining and online analytical processing tools translate warehouse contents into business intelligence by means of a variety of statistical analyses and data visualization methods (Brown 1995; Fogarty 1994). Table 1 provides a list of some of the common terms used within the data warehouse community.

A number of commercial products attempt to fulfill warehousing needs, including the SAS Institute's SAS/Warehouse Administrator, IBM's Visual Warehouse, Cognos's PowerPlay, Red Brick Systems' Red Brick Warehouse, and Information Builders' FOCUS. A discussion of these tools is beyond the scope of this chapter.

4. ENTERPRISE TOOLS

Since the early days of business computing, system designers have envisioned information system architectures that would allow the seamless sharing of information throughout the corporation. Each major advance in information technology spawned a new generation of applications such as CIM or MRP II that claimed to integrate large parts of the typical manufacturing enterprise. With the recent

TABLE 1 Common Terms Used by the Data Warehousing Community

Term	Description
Aggregates	Precalculated and prestored summaries that are stored in the data warehouse to improve query performance. For example, for every VAR there might be a prestored summary detailing the total number of licenses, purchased per VAR, each month.
Business Intelligence Tools	Those client products, which typically reside on a PC, that are the decision support systems (DSS) to the warehouse. These products provide the user with a method of looking and manipulating the data.
Data Extraction, Acquisition	The process of copying data from a legacy or production system in order to load it into a warehouse.
Data Mart	Separate, smaller warehouses typically defined along organization's departmental needs. This selectivity of information results in greater query performance and manageability of data. A collection of data marts (functional warehouses) for each of the organization's business functions can be considered an enterprise warehousing solution.
Data Mining	A collection of powerful analysis techniques for making sense out of very large datasets.
Data Modeling	The process of changing the format of production data to make it usable for heuristic business reporting. It also serves as a roadmap for the integration of data sources into the data warehouse.
Data Staging	The data staging area is the data warehouse workbench. It is the place where raw data are brought in, cleaned, combined, archived, and eventually exported to one or more data marts.
Data Transformation	Performed when data is extracted from the operational systems, including integrating dissimilar data types and processing calculations.
Data Warehouse	Architecture for putting data within reach of business intelligence systems. These are data from a production system that now resides on a different machine, to be used strictly for business analysis and querying, allowing the production machine to handle mostly data input.
Database Gateway	Used to extract or pass data between dissimilar databases or systems. This middleware component is the front-end component prior to the transformation tools.
Drill Down	The process of navigating from a top-level view of overall sales down through the sales territories, down to the individual sales person level. This is a more intuitive way to obtain information at the detail level.
DSS (Decision Support Systems)	Business intelligence tools that utilize data to form the systems that support the business decision-making process of the organization.
EIS (Executive Information Systems)	Business intelligence tools that are aimed at less sophisticated users, who want to look at complex information without the need to have complete manipulative control of the information presented.
Metadata	Data that define or describes the warehouse data. These are separate from the actual warehouse data, which are used to maintain, manage, and support the actual warehouse data.
OLAP (Online Analytical Processing)	Describes the systems used not for application delivery, but for analyzing the business, e.g., sales forecasting, market trends analysis, etc. These systems are also move conducive to heuristic reporting and often involve multidimensional data analysis capabilities.
OLTP (Online Transactional Processing)	Describes the activities and systems associated with a company's day-to-day operational processing and data (order entry, invoicing, general ledger, etc.).
Query Tools and Queries	Queries can either browse the contents of a single table or, using the databases SQL engine, they can perform join conditioned queries that produce result sets involving data from multiple tables that meet specific selection criteria.

TABLE 1 (*Continued*)

Term	Description
Scrubbing/Transformation	The processes of altering data from its original form into a format suitable for business analysis by nontechnical staff.
Star Schema	A method of database design used by relational databases to model multidimensional data. A star schema usually contains two types of tables: fact and dimension. The fact table contains the measurement data, e.g., the salary paid, vacation earned, etc. The dimensions hold descriptive data, e.g., name, address, etc.

growth in client/server applications and more powerful relational databases, a new breed of ''enterprise'' tools has emerged in the last 10 years. These enterprise tools enable the design of information systems, commonly called ''enterprise resource planning'' or ''ERP systems,'' that are truly enterprise-wide. They have become popular in large part because they are replacing older legacy systems that are often complex, redundant, and notoriously inflexible, with a suite of integrated application modules and integrated data that is accessible throughout the enterprise. Another common feature among the various enterprise tools is that they are intended to develop a set of integrated applications, typically for large business enterprises. These business enterprises may be global in nature, and these systems are typically running on a distributed client/server platform with a common, integrated database.

4.1. Enterprise Tools Defined

Today, when software vendors and consultants talk about enterprise tools they are referring to a class of tools that are frequently referred to as enterprise resource planning (ERP) software. *ERP* is a term used by the industry to refer to a collection of applications or modules that can be used to manage the whole business. ERP software describes the class of prepackaged software applications that can be configured and modified for a specific business to create an ERP system. Here we use the term *ERP* interchangeably with the term *enterprise tools*. ERP software or packages are defined here as:

An integrated set of software modules designed to support and automate core business processes that may include logistics, sales, manufacturing, finance, and human resources.

A key point to note in this definition is that the software is integrated around a common user interface and around a shared database so that data can easily be shared between applications such as production and finance. Also of importance is the notion that the software is designed to be modular, so users can implement whichever modules they choose, and that it is oriented towards supporting business processes rather than the traditional functional view of the organization. It is this process view that is often the most misunderstood part of implementing an ERP system, since it can entail substantial reengineering of core business processes.

4.2. Evolution of Enterprise Resource Planning

The idea that a large corporate enterprise could be supported by a completely integrated information system is certainly not new. It has been envisioned since the early days of computers. These visions began to become more concrete in the manufacturing area when computer integrated manufacturing (CIM) systems became popular in the early 1980s. The goal of CIM was to link up the CAD output from the engineering function to marketing personnel so it could be reviewed and also be shared with the production planning and production departments. A part of the product's design was the bill of materials (BOM), which contained the material specifications and quantities to produce a single unit of the particular product. In later versions of CIM, this BOM might also be used create an electronic order and send it to approved vendors via an electronic data interchange (EDI) connection. The basic benefits were that it would decrease product development time and production costs while allowing designs to be changed rapidly and smaller production runs to be made. Because of its inherent complexity, few firms developed these in-house and most chose to purchase generic CIM software. However, CIM did not achieve all the anticipated gains in integration and gave way to another manufacturing integration concept, MRP II.

The lessons learned in trying to implement CIM in the manufacturing environment were used in the development of the newer material requirements planning (MRP) systems in the mid-1980s. The functionality of these new planning systems continued its expansion as companies tried to take a more holistic view of manufacturing processes. Improvements in new relational database technology made it possible to integrate the different departments involved in the manufacturing processed around a central database. MRP II systems extended the integrated manufacturing concept by linking the production schedule to other organizational systems in finance, marketing, and human resources. For instance, changes in customer demand would have a ripple effect on cash flow planning, and personnel planning, in addition to purchasing. The goals again were to cut costs of production, reduce inventory, improve customer service, and integrate production decisions better with finance and marketing functions.

4.2.1. The Appearance of Enterprise Resource Planning

While MRP II remained focused on shortening the ordering and production cycles in a manufacturing environment, several advances in technology in the late 1980s led to the development of ERP tools and applications. Beyond the manufacturing processes, ERP encompasses business planning, sales planning, forecasting, production planning, master scheduling, material planning, shop floor control, purchasing and financial planning, routing and bill of material management, inventory control, and other functions. As with MRP II, a centralized relational database is key to the success of an ERP application. Other new technologies that enabled the spread of ERP tools include the rapidly maturing client/server distributed architecture and object-oriented programming (OOP) development practices. Both of these factors make the ERP system more scalable both in terms of the hardware and also the modularity of the software. This scalability in turn lends itself to continuous expansion of their functionality, as can be seen by the rise of new ERP modules that extend to supply chain and customer relationship concepts, generating tighter linkages with both the customer's and supplier's own planning systems.

As the scope of these integrated applications has grown, it is clear that the complexity has also grown. Horror stories about runaway ERP implementations appear frequently in the popular press. Two examples occurred in 1999 with the much-publicized problems of Hershey's ERP implementation and also a similar set of problems at Whirlpool. After more than $100 million had been spent on Hershey's ERP implementation, Hershey's inventory system did not work and they reportedly lost more than $100 million during the busy Halloween season (Stedman 1999b). ERP costs for large multinational firms can easily take several years and cost several hundred million dollars. Corning Inc. reported in 1998 that they expected their ERP project to take between five and eight years to roll out to all 10 of its manufacturing divisions (Deutsch 1998). Common complaints about ERP products are that they are inflexible and if anything special is desired, expensive consultants must be hired to design "tack on" modules that often must be programmed in some obscure proprietary language. This "one size fits all" mentality makes it difficult for smaller companies to justify the expense of an ERP systems. It may be that ERP will not fit with certain types of organizations that require extensive flexibility. Experience has shown that ERP systems seem to work best in those organizations with a strong top-down structure. Also, the very complexity of an ERP implementation frightens off many potential adopters because it may involve shifting to a client/server environment combined with an often painful process of reengineering.

With all the problems and risks associated with ERP, why would anyone even want to consider an ERP system? Despite these problems, from 1995 to 1999 there was a tremendous growth in corporate implementation of ERP systems. Over 70% of the Fortune 500 firms (including computer giants Microsoft, IBM, and Apple) have implemented an ERP system. Though market growth slowed a bit in 1999 due to corporate focus on the Y2K problem, during this same five-year period, ERP vendors averaged about 35% annual growth (Curran and Ladd, 2000). Globally, the ERP market is projected to grow at a compound annual rate of 37% over the next few years, reaching $52 billion by 2002.

The reality is that if a company today wants an enterprise-wide IS, there are few alternatives to ERP tools. Though the risks are very high, the potential benefits include Y2K and Euro compliance, standardization of systems and business processes, and improved ability to analyze data (due to improved data integration and consistency). A vice president at AlliedSignal Turbocharging Systems said the company was replacing 110 old applications with an ERP system and this would help the $1 billion unit do a much better job of filling orders and meeting delivery commitments (Davenport 1998). Prior to the year 2000, the most common expected benefits of implementing an ERP solution cited in interviews of information technology executives included:

- Solved potential Y2K and Euro problems
- Helped standardize their systems across the enterprise
- Improved business processes
- Improved efficiency and lowered costs
- Needed to support growth and market demands
- Global system capabilities
- Ability to integrate data (Bancroft et al., 1998)

4.2.2. The Enterprise Tools Market

Next to electronic commerce, ERP is one of the most dynamic and rapidly growing areas of business computing. The market for ERP tools has grown at an average annual rate of 30–40% since 1994. ERP revenues for 1998 were approximately $23 billion and were expected to grow to $52B by 2002. Because it grew out of the MRP II market, the early focus of ERP tools was on the manufacturing complex, and it expanded from there, to service and even government and educational institutions. Penetration of this market by ERP tools has reached 70%, and it is said that almost all of the remaining companies are considering installing an ERP solution. The ERP tools market was really defined and is still dominated by a German software company, SAP AG, which currently controls 33% of the ERP market. SAP had revenues in 1998 of $5.05 billion, a 41% increase from the year before (Curran and Ladd 2000). Oracle is the next-leading ERP tools vendor, at 10% of the market, then J.D. Edwards Inc. at 7%, PeopleSoft, Inc. at 6%, and Baan, NV rounding out the top 5 ERP vendors at 5%.

Begun in 1972 by four IBM German engineers, SAP virtually defined the ERP tools market. Its early product, R/2, was a mainframe-based extension to MRP II. However, its top leadership made the strategic decision in 1988 to focus all R&D efforts on developing enterprise solutions for the client/server platform. SAP was one of the earliest software vendors to recognize this fundamental shift and is now the leading client/server application software vendor and the fourth-largest software company in the world. The list of their clients includes 9 of the 10 companies with the top market value in the world and includes IBM, Microsoft, and Apple within the computer industry itself. Their product, R/3, is available in 14 languages and is used by over 10,000 customers in more than 90 countries (Curran and Ladd 2000). SAP has expanded beyond its traditional base in manufacturing industries into service industries, education, and governmental agencies. Service industry clients include Burger King, Sothebys, Lufthansa, and Deutsche Banke. Among governments, the Canadian government standardized on SAP and has one of the largest public installations. MIT and several other institutions of higher learning have also implemented SAP to run their fundamental operations.

The cost of SAP's R/3 for a mid-sized company averages about $750,000, but the implementation costs can be many times the software costs. Typical costs for a full-fledged Fortune 500 firm can typically run about $30 million in license fees and $100–200 million in consulting fees (Kirkpatrick 1998). SAP was the first to institute a University Alliance, which fosters the teaching of business courses that use R/3. SAP invests about 20–25% of its revenues in R&D. This represents an amount greater than all the other ERP vendors combined and ensures that it should be able to respond rapidly to changes in the ERP market.

Of the other ERP vendors, Oracle is in the odd position of being the second-largest software company in the world next to Microsoft and being second to SAP in the area of ERP tools. One reason for this is the fact that Oracle is the preferred database engine for SAP and so is often even shipped with R/3 to new clients. However, Oracle has begun to compete more aggressively with SAP with respect to its ERP tools. Initially, it focused on developing financial application modules and targeted the smaller to mid-sized corporate market. In the last several years, it has either developed application modules in-house or purchased smaller software development firms, so that they now can compete with SAP on the entire ERP spectrum. In addition to its enterprise-wide financial applications, Oracle has recently rolled out an Internet Procurement Suite, Oracle Exchange Suite, and the Oracle Customer Relationship Management (CRM) package. Of all the ERP vendors, it was the first to aggressively extend the ERP functions to e-commerce. As such, 19 of the top 20 e-businesses (e.g., Amazon.com, eBay, Barnes & Noble, and CDNow) use Oracle ERP software. One indicator of this strategy is that in 1988 it was the first to allow users to access their enterprise ISs from a standard web browser. Many venture capitalists reportedly refuse to fund an Internet start-up unless Oracle is a part of the business plan (Stroud, 1999).

PeopleSoft Inc. made its niche in the ERP market by being the company that disaffected SAP users could turn to as an alternative. When it was founded in 1987, it was a vendor of software for the human resource (HR) function. It has since focused on this application in the ERP market, while SAP has focused on inventory management and logistics processes. The quality of PeopleSoft's HR modules was such that some companies would adopt SAP for part of their processes and PeopleSoft specifically for their HR processes. Because it was more flexible in its implementations, PeopleSoft won over customers who just wanted a partial ERP installation that they could integrate more easily than with their existing systems. PeopleSoft was perceived as being more flexible and cost-effective because it developed its whole ERP software suite by collaborating with firms like Hyperion, Inc. for data warehousing, data mining, and workflow management applications. Once PeopleSoft got into a company with its HR applications, it then began offering the financial and other ERP software modules. This led to rapid growth; 1992 revenues of $33 million grew to $1.4 billion in 1998 (Kirkpatrick 1998). Along with the rest of the ERP software industry, PeopleSoft has been scrambling to extend its ERP offerings to the Internet. In 1999, it began development of eStore and eProcurement, and it acquired CRM vendor Vantive (Davis 1999).

Baan Software, a Dutch firm, was an early player in the ERP software market, shipping its first product in 1982. It was the first ERP vendor to focus on an open UNIX computing environment in 1987. Its early strengths were in the areas of procurement and supply chain management. Through internal development and acquisitions, Baan offers one of the most complete suites of ERP tools, including accounting, financial, and HR applications in one comprehensive package or separately for functional IS development. Shifting toward an Internet focus, Baan has recently acquired a number of tool development firms and entered into an alliance with Sun for its Java-based software. It is also moving aggressively to integrate XML-based middleware into its next release of BaanERP (Stedman 1999a).

J.D. Edwards began in 1977 as a CASE tool vendor but developed its own suite of ERP applications, primarily for the AS/400 platform. Its modules now include manufacturing, distribution, finance, and human resources. The company emphasizes the flexibility of its tools and has risen quickly to be fourth-largest ERP vendor, with nearly $1 billion in 1998 sales (Kirkpatrick 1998). It has been an aggressive ERP vendor in moving to the ERP Outsourcing model, where firms access the J.D. Edwards software over a secure network link to an offsite computer. This means firms' pay a fixed monthly fee per user and don't have large up-front investments in expensive implementations and software and skills that are rapidly out of date. This network-distribution-based outsourcing model for ERP vendors has yet to be validated, though, and presents many potential technical problems.

4.3. Basic Concepts of Enterprise Tools

As one reads the literature on ERP tools, it quickly becomes apparent that very little academic research has been performed to in this rapidly emerging area. The field, for the most part, is defined by a few case studies, and research consists primarily of industry publications (Brown and Vessey 1999). Articles appearing in industry publications generate many new terms, which the authors seldom define, and they also tend to over-hype new technology. Various ERP-related terms have been put forth, including *ERP tools, ERP integration tools, ERP network tools, ERP security tools, ERP software,* and *ERP system*. Some vendors are also trying to move away from the term *ERP* to the term *enterprise planning* (EP) and even *extended resource planning* (XRP) systems (Jetly, 1999). As mentioned earlier, *enterprise resource planning* (ERP) is a term used by the industry to refer to prepackaged software tools that can be configured and modified for a specific business to create an enterprise IS. Common enterprise-wide processes are integrated around a shared database. Although mainframe versions of ERP tools exist, for the most part ERP tools assume a client/server platform organized around a centralized database or data warehouse. The essential concepts that generally define ERP tools include integration of application modules, standard user interfaces, "process view" support, integrated database with real-time access, and use of an "open" system architecture usually built around a client/server platform.

4.3.1. ERP and the "Process View"

Another common feature of ERP tools/systems is that they usually involve the extensive application of business process reengineering (Curran and Ladd 2000). This process is often very painful because it may require an organization to reexamine all of its core processes and redesign whole departments. The ERP tool chosen actively supports this "process view." In this sense, a process can be thought of specified series of activities or a way of working to accomplish an organizational goal. To be properly supported by an ERP tool, processes have a defined structure and order and identifiable inputs and outputs. This process view is in contrast to the more traditional functional view, where processes are broken down into activities that lack integration. ERP tools offer further support for the process view of an organization by building into the system standard business processes such as order management or shipping scenarios. These scenarios are then used as the basis for configuring and customizing an ERP system for the specific company. Certain vendors go so far as to try and capture the "best practices" within whole industries, such as the pharmaceutical industry, and use these as models for implementing ERP systems within that industry. Benefits include streamlined business processes, better integration among business units, and greater access to real-time information by organizational members. For many organizations, movement to an ERP-based IS is seen as one way to use computer technology to provide substantial gains in productivity and speed of operations, leading to competitive advantage (Davenport 1998).

4.3.2. ERP and the Open Systems Architecture

It is clear that one of the reasons ERP tools became so successful was that they were able to take advantage of improvements in the client/server architecture and in middleware. The general movement to the client/server platform in the late 1980s meant that firms could reduce the cost of a highly centralized configuration in favor of a more flexible client/server setup. The C/S configuration preferred by most ERP vendors is that of the standard three-tier C/S platform, which consists of a presentation server, connected to the application server, which is in turn connected to the database server. The "3" in SAP's R/3 product name even refers to this three-tier platform, as opposed to the old R/2 mainframe product.

ERP systems take advantage of new developments in middleware that make it feasible to distribute applications across multiple, heterogeneous platforms. This means that ERP applications can be running on different operating systems and can be accessing data stored in various formats on different database servers worldwide. The middle-ware built into such ERP packages such as SAP and PeopleSoft contains all the communication and transaction protocols needed to move the data back and forth from the database server to the presentation server. The ultimate goal of this is that the movement of data across the varied platforms be transparent to the user, who comes into the ERP applications via a common graphical user interface or even a standard browser interface for e-commerce extensions to the enterprise ISs.

4.3.3. ERP Application and Data Integration

ERP packages include software for a wide range of applications, ranging from purchase orders to accounting and procurement to warehousing. They grew out of the need to plan, manage, and account for resources in manufacturing environments. In recent years, their functionality has grown in both breadth and depth. As companies integrate business units through consolidation or global operations, their information technology's ability to support these changes is often challenged. The broad functionality of the ERP applications allows companies to replace much of their systems with ERPs, providing better support for new business structures and strategies. The great appeal of ERPs is that employees enter information only once and that information is then available to all applications company-wide. Activities tracked or triggered by the ERP software are also automatically captured in the enterprise's accounting general ledger, without the risk of reentry errors. This means that IS reports throughout a company are based on accurate, real-time information. The value of this is borne out by the large amounts of money that multinational firms are willing to pay to implement ERPs.

An ERP tool provides companies with a standard interface for all of a firm's IS applications, a feature that may be lacking with the older legacy application systems. This has become even more important with rise of CRM and the movement of ERP toward Web-enabled enterprise applications. Although standard, SAP's R/3 interface is famous for some confusing and complicated aspects. Research on an improved user interface is continuing, and the new versions are much improved and more intuitive. For e-commerce extensions to ERP-based ISs, users will soon be able to navigate using a standard web browser such as Microsoft's Internet Explorer. Soon users will also have the option of accessing their ERP-based ISs using wireless, pen-based, hand-held computers (Marion 1999b).

4.4. Standard Enterprise Tool Applications

When companies look to implement an IS with an ERP tool, they typically analyze their functional requirements, technical integration issues, costs, and strategic issues. With respect to analyzing the functional requirements of a company and comparing them to the functionality of the various ERP tools available, this task is clouded somewhat by several factors. First, with the ERP emphasis on supporting core business processes, designers must be more knowledgeable about how all the corporate processes flow into each other, from creating a sales document (i.e., report) all the way to maintaining records of employee profiles. Beyond this, with the new emphasis on business plans that integrate alliance partners and customers more closely, a company must move its focus away from a strictly intraorganizational one to an increasing awareness of transorganizational needs. Both of these needs are further complicated by the fact that each vendor defines the standard business processes in different ways and hence the component architectures of their software reflect a different set of configurable components. Also, within the different vendor packages there are varying degrees of overlap in the functionality of each of the modules. This makes it very difficult to compare the functionality of specific modules from different vendors and how much of them will be able to be integrated into existing systems. The functionality varies, making it difficult to do a proper comparison of the purchase cost and implementation cost of the modules. This problem is further complicated when one considers the different sets of "industry solutions" that ERP vendors push as ways of smoothing the implementation process.

One way to get around some of the problems in making comparisons of ERP tool functionality is to try to translate a particular company's needs into a generic set of core processes, such as order management, inventory management, and so forth. Companies may then go through several iterations of attempting to map each vendor's modules to those requirements until the desired level of functionality has been met. In this way, some of the gaps and overlaps will become more apparent. Further requirements, such as those related to technical, cost, support, and data integration can be overlaid onto the matrix to refine the comparison further. This section examines the four standard core business processes that almost all ERP tools are designed to support. These processes include the sales and distribution processes, manufacturing and procurement, financial management, and human resources. Each ERP package defines these processes slightly differently and has varying degrees of functionality, but all tend to support the basic processes presented here with various combinations of modules.

4.4.1. Sales and Distribution Applications

With the advent of newer, more customer-oriented business models such as Dell's "build to order" computer manufacturing model, it is increasingly important for companies to take customer needs into account and to do it in a timely fashion. ERP tools for supporting sales and distribution are designed to give customers a tighter linkage to internal business operations and help to optimize the order management process. The processes that are most typically supported by the sales and distribution (sales logistics) ERP modules include order entry, sales support, shipping, quotation generation, contract management, pricing, delivery, and inquiry processing. Because the data and functional applications are integrated in a single real-time environment, all the downstream processes, such as inventory management and production planning, will also see similar improvements in timeliness and efficiency.

When companies adopt a particular ERP vendor's package, they often begin with the sales and distribution model because this is the beginning of the business cycle. ERP customers are given a variety of different sales scenarios and can choose the ones that most closely mirror how they would like to run their sales processes. Then they can configure that module to reflect more accurately the way they want to support their sales processes. If they need to modify the module more substantially, they will have to employ programmers to customize the module and possibly link it to existing custom packages that they wish to keep. Whichever approach they choose, the generic sales processes that are chosen by the ERP customer can be used as a good starting point from which to conduct user requirements interview sessions with the end users within the company.

One example of a typical sales process that would be supported by an ERP sales and distribution module is the "standard order processing" scenario. In the standard scenario, the company must respond to inquiries, enter and modify orders, set up delivery, and generate a customer invoice (Curran and Ladd 2000). The system should also be able to help generate and manage quotes and contracts, do credit and inventory availability checks, and plan shipping and generate optimal transportation routes. One area that does distinguish some of the ERP packages is their level of support for international commerce. For example, SAP is very good in its support for multiple currencies and also international freight regulations. Their sales and distribution module will automatically perform a check of trade embargoes to see if any particular items are blocked or are under other trade limits for different countries. They will also maintain international calendars to check for national holidays when planning shipping schedules. This application is also where customer-related data are entered and corresponding records maintained. Pricing data that is specific to each customer's negotiated terms may also be a part of this module in some ERP packages (Welti 1999).

Other sales processes that might be considered a part of the sales and distribution module (depending on the particular ERP product) include support for direct mailing campaigns, maintenance of customer contact information, and special order scenarios such as third party and consignment orders. With the rapid growth of the newer CRM software packages, much of the support for these processes is now being moved to the Internet. E-mail and Web-based forms enabling customer input mean that ERP sales and distribution modules will have to evolve rapidly along CRM lines. Currently, the most advanced with respect to this type of web integration is the ERP package from Oracle, but the other vendors are moving quickly to enhance their current sales and distribution module functionality (Stroud 1999).

4.4.2. Manufacturing and Procurement Applications

Certainly the most complex of the core business processes supported is the large area generally called manufacturing and procurement. Because ERP tools grew out of MRP II and its precursors, these modules have been around the longest and have been fine-tuned the most. The suite of ERP modules for this application is designed to support processes involved in everything from the procurement of goods used in production to the accounting for the costs of production. These modules are closely integrated with the financial modules for cash flow and asset management and with the sales and distribution modules for demand flow and inventory management. Key records maintained by these application modules include material and vendor information.

Among the supply chain activities, the manufacturing and procurement applications handle material management, purchasing, inventory and warehouse management, vendor evaluation, and invoice verification (Bancroft et al. 1998). These activities are so complex that they are typically divided up among a whole suite of manufacturing modules, such as in SAP's R/3, which consists of materials management (MM), production planning (PP), quality management (QM), plant maintenance (PM), and project management (PM). These various modules can be configured in a wide variety of ways and can accommodate job shop, process-oriented, repetitive, and build-to-order manufacturing environments. The PM module is designed to support the management of large, complex projects, such as shipbuilding or special construction projects. Also, despite being oriented more towards large manufacturing enterprises, ERP vendors have made great strides in modifying these modules to accommodate the special needs of service industries such as education and government.

With respect to procurement processes, ERP tools can be used to configure the desired IS in many ways, but they are generally built around a standard procurement scenario. Whether a company must procure subcontract work, consumable goods, or material for their day-to-day production stock, the chain of procurement activities is somewhat similar. These activities involve recognizing a material need, generating a purchase requisition, sending out RFQs to approved vendors, choosing a vendor, receiving and inspecting the goods, and verifying the vendor invoice (Curran and Ladd, 2000). In this area, many companies already use EDI effectively to support their procurement processes, and so most ERP tools are equipped to support standard EDI transactions. The growth of the Internet and extended supply chain management means that ERP tools are increasingly linked directly to their preapproved vendors to further shorten the procurement cycles.

4.4.3. Accounting and Finance Applications

Like the other core business activities, accounting has been undergoing a total reorientation towards a more process-oriented view of the enterprise. Most ERP tools support the accounting and financial processes with another suite of modules and submodules. These include general ledger, AR/AP, asset management, controlling, taxation, and cash flow projection. The more robust ERP packages support accounting for multiple divisions, across multiple currencies, and also support consolidation of financial statements across countries and divisions. Again, because of the integrated nature of all the ERP applications and data, the output from these modules is generated in a much more efficient and timely manner than in the older, legacy systems. The key, of course, is to make sure that the data are highly accurate when it is entered into the system. The downside of data integration is that inaccurate data can be spread through many different applications' reports.

A large part of the accounting and financial activities in an enterprise is the efficient processing of vendor payments and the posting of payments in the correct order into the various ledger accounts. This can be done automatically via EFT or manually in an ERP system, depending on the preferences of the vendor or customer. One of the outputs from the procurement process is a vendor-produced invoice, and this input is the usual starting point for the external accounting activities. Once the invoice is entered either via EDI, manual entry, or even scanning, the invoice is posted to the appropriate account and the amount is verified by matching it with a purchase order and any other partial payments. Once verified, a payment release is issued and payment is issued either electronically or manually. On the customer side, the accounting module also maintains the customer credit records and generates dunning notices (i.e., reports) according to a predefined dunning procedure for each individual customer. The ERP application system will also automatically add on any interest or processing charges that are defined as part of the sales contract for that particular customer. As payments are received and entered into the system, it applies them to the appropriate invoice and moves them to the appropriate GL account.

Most ERP tools also support a variety of management accounting activities. The largest part of these internal accounting activities center on cost management and profitability analysis. Depending on the type of organization, costs can be collected as a byproduct of procurement and production and the system can be configured to generate cost estimates, simultaneous production costing, and also final costs for a period or a project. These costs can be further analyzed and broken down by organizational unit and by specific product or project. Profitability analysis can likewise be conducted at various points in the process and be broken down by organizational unit and product or project. One must bear in mind that with a properly configured suite of ERP financials, the data that are collected and analyzed for these controlling reports are very timely and generating the reports can be done much more efficiently than previously. Analysis can also be conducted across international divisions of the company and whole product lines.

4.4.4. Human Resource Applications

Typically the last set of business processes to be converted to an ERP system, HR applications have recently become very popular as companies look to areas where they might be able to gain a competitive advantage within their respective industries. Because of this, there has been substantial improvement in the HR modules of the leading ERP vendors. This improvement in turn has led to a substantial second wave of ERP implementations among those early adopters, who may have only adopted the financial and manufacturing or sales modules in the first wave of ERP implementations (Caldwell and Stein 1998).

In most organizations, HR too often has become a catchall for those functions that do not seem to fit elsewhere. As a result, HR typically houses such disparate functions as personnel management, recruiting, public relations, benefit plan management, training, and regulatory compliance. In ERP applications, the HR modules house the employee data and also data about the current organizational structure and the various standard job descriptions. The latest wave of ERP tools now supports most of these activities in a growing suite of HR modules and submodules. For example, personnel management includes the recruitment, training and development, and compensation of the corporate workforce. Typical recruitment activities include realizing the internal need for new personnel, generating a job description, posting advertisements, processing applicants, screening, and selecting the final choice for the position. Once the position has been filled, then training assessments can be done and a benefit and development plan can be implemented. Other core processes in the HR module include payroll processing, benefits, and salary administration; sometimes they include time and travel management to help employees streamline reimbursement for travel expenses.

4.5. Implementing an Enterprise System

Because of the large investment in time and money, the selection of the right enterprise tool has a tremendous impact on the firm's strategy for many years afterward. Choosing a particular ERP package means that a company is entrusting its most intimate core processes to that vendor's software. This is a decision that cannot be easily reversed after it has been made. Some firms have chosen to implement an ERP package not realizing the extent of the commitment involved and then had to stop the project in the middle (Stedman 1999b). This is often because these companies are unwilling to reexamine their core processes to take advantage of the integrated nature of an ERP package.

What makes choosing an ERP package different from other types of organizational computing decisions? As discussed above, the large scale and scope of an ERP system make it even more critical than local or functional IS applications for the well-being of the enterprise, apart from the high-cost issue. It is also different in that it is oriented towards supporting business processes rather than separate islands of automation and so the typical implementation involves business experts more than technology experts as opposed to traditional system implementations. The process-oriented nature of an ERP implementation also means that cross-functional teams are a large part of the design process. Because of the high potential impact on an organization due to the redesign of core business processes and reduction of the labor force needed for these processes, one of the more subtle differences is that end users may fight the changes brought about by an ERP system. Therefore, project leaders must also be skilled in "change management" (Davenport 1998).

4.5.1. Choosing an Enterprise Tool

Given that an ERP implementation is so risky to an enterprise, what should a firm consider when choosing an ERP tool? There are currently a handful of case studies in the academic literature and numerous anecdotal references in the trade literature. From these we can derive some factors that a firm should take into account. Probably the most important criterion is how well the software functionality fits with the requirements of the firm. A firm could develop a matrix of its core business process requirements and then try to map the modules from the various ERP packages to the matrix to see where obvious gaps and overlaps might exist (Jetly 1999). A byproduct of this analysis will be a better understanding of the level of data and application integration that the package offers. Increasingly, the ERP package's support for electronic commerce functions will be a major factor in choosing the right package.

Beyond an analysis of the ERP package functionality, a firm must also take technical factors into consideration. Each package should be evaluated for the flexibility of the architectures that it supports and its scalability. How well will this architecture fit with the existing corporate systems? How many concurrent users will it support and what is the maximum number of transactions? Will the module require extensive modifications to make it fit with our business processes and how difficult will it be to change them again if my business changes? Given the potential complexity of an ERP implementation, potential users should be concerned about the ease of use and amount of training that will be required for end users to get the most out of the new system. ERP vendor-related questions should center on the level of support that is offered and how much of that support can be found locally. They should also look into the overall financial health and stability of their prospective ERP vendors and the scope and frequency of their new product releases and updates.

With respect to cost, several factors should be kept in mind when choosing an ERP package. First is the cost of the software and hardware itself that will be required. Besides the required number of new database and application servers, hardware for backup and archiving may also be required. Software costs are typically on a per-module basis and may vary between modules, but a firm may also have to purchase the submodules in order to get the desired functionality. If these modules need to be customized or special interfaces with existing systems need to be programmed, firms should consider the amount of customization required. This cost is exaggerated for some ERP packages, such as SAP's R/3, which require knowledge of the rather exotic proprietary language ABAP/4 in order do extensive customization. Training and ongoing support for these packages will be a continuing expense that should also be considered.

4.5.2. Enterprise System Implementation Strategies

With the high cost of implementing an enterprise information system, developers are under pressure to build them as quickly as possible. Given that the actual system development costs can typically range from 2–10 times the actual cost of the software and hardware (Doane 1997), many ERP consulting firms have sprung up to service this market. Of course, each of them has its own methodology which it claims to be faster and more efficient than the other's. Most of these methodologies represent some variation of the traditional phased systems implementation, so this will be discussed here. Another current ERP debate is the relative merits of the ‘‘big bang’’ conversion strategy as opposed to a slower, phased conversion (Marion 1999a).

Because of the strategic nature of ERP decisions, they are usually made at the top executive level. To help coordinate an ERP implementation, firms often create a steering committee with some top executives and high-level management from each of the affected divisions of the firm. The committee will determine the initial scope of the ERP implementation and also which area of the firm will act as a test bed for debugging the initial IS, assuming the firm does not choose to do a big bang implementation. As a result, a firm may choose to begin with the sales and distribution module in its marketing department and expand from there. A project team consisting of IS professionals and end users should then be formed. This team will work in conjunction with the steering committee. Studies show that the importance of the project leader cannot be underestimated and that this person should have strong leadership skills and some experience in implementing an ERP-based IS (Bancroft et al. 1998).

It is the project team's job to analyze the current business processes that will be the focus of the ERP-based IS and look at the existing hardware and software along with the existing data and interfaces. In order to explore the ERP software functionality in more detail, firms sometimes map the initial business processes to the chosen ERP package and do a limited test installation. After exploring the ERP package, the project team will come up with a detailed new design of the business processes in question and a prototype system may be developed and tested by users. Once the prototype has been accepted by users, then the complete ERP-based IS application will be developed and implemented and the data migration will take place. Once the IS has been rolled out for the initial application, it will be expanded to include the rest of the enterprise applications that have been identified in the initial scope of the project. It should be noted that most ERP packages support this development process with built-in CASE tools, standard process scenarios and scripts, and possibly a set of recommended ‘‘best’’ practices for specific industries.

As an alternative to this phased approach to ERP implementation, some firms choose the big bang approach, which is riskier but less expensive, in which a firm shifts its entire application for the whole firm from its old legacy system to its new ERP application. This is a high-risk implementation method because the resulting system may have so many bugs that chaos may result, after which it may take months for the firm to recover. An example of the potential problems occurred in the fall of 1999 when Hershey's tried to implement its ERP inventory application using the big bang approach and lost over $100 million due to its failure to manage inventory properly (Stedman 1999b). However, there have been big bang success stories at McDonalds Corp. and Rayovac, and perhaps the benefits of this approach can outweigh the risks (Marion 1999a). Some of the benefits include lower cost because it requires less expensive consultant time. It also reduces the amount of interface development with the old legacy systems because there is no temporary connection to maintain with the new ERP applications. The other problem with the long phased approach is that when one application is ready, updated versions of the ERP software are available for other applications when they are ready to be rolled out, and then developers must deal with a whole new set of compatibility and conversion issues.

4.5.3. Critical Success Factors for ERP Implementations

From the various case studies reported in the trade literature, several key factors have been reported that seem critical for the success of an ERP implementation (Bancroft et al. 1998). These include:

1. Management support for making the hard decisions
2. Capability and readiness of the organization for making changes
3. A balanced project team with strong leadership
4. Reasonable project scope
5. The right ERP package
6. The right implementation method
7. A strong training plan for users and the project team

The most commonly cited of these factors is the importance of management support for the success of the project. This factor has been widely documented for other types of business computing projects, but it is especially important in implementing an ERP system because this may involve a radical redesign of some core processes and resulting major changes in the organization. In a project as complex as an ERP system, problems will occur. Leaders must maintain the strong vision to persevere through all the changes. One study showed that nearly 75% of new implementations resulted in some organizational change, and these changes could be very severe (Boudreau and Robey 1999). Project teams should also be balanced with respect to business area experts and IS professionals. As the software becomes easier to configure and has more complete functionality, it seems clear that the trend of shifting more responsibility away from the IS professionals to the business experts will continue (Bancroft et al., 1998). Because of the cross-functional nature of ERP-based ISs, it is also important that team members be drawn from multiple business units and assigned full-time to the project (Norris et al., 1998).

4.6 Future of Enterprise Tools

The new generation of enterprise tools is beginning to enable large corporations to reap significant gains in efficiency and productivity, but the investment costs of implementing an ERP system can be a huge hurdle for smaller companies. Enterprise tools will continue to improve in overall functionality and ease of use as modules are developed and refined. The functionality will continue to be extended to the customer, supplier, and other business partners in what some call the Extended Resource Planning (XRP) software generation (Jetly 1999). Most of these extensions are based on some form of e-business model using the Internet. A description of some of the principal newer enhancements to ERP systems follows. These are taking enterprise ISs into the transorganizational realm.

4.6.1. Enterprise Tools and Supply Chain Management (SCM)

One logical outgrowth of the increased integration of enterprise-wide data and support for core business processes is the current interest in supply chain management. In the 1980s, much of the focus in manufacturing was on increasing the efficiency of manufacturing processes using JIT, TQM, Kanban, and other such techniques. Given the large investment in these improvements in manufacturing strategies, companies managed to lower the cost of manufacturing goods as far as was practical. These same companies are finding that they can take a system view of the whole supply chain, from suppliers to customers, and leverage their investment in ERP systems to begin to optimize the efficiency of the whole logistics network. In this context, supply chain management is defined as follows:

> A set of approaches utilized to efficiently integrate suppliers, manufacturers, warehouses, and stores, so that merchandise is produced and distributed at the right quantities, to the right locations, and at the right time, in order to minimize system-wide costs while satisfying service level requirements (Simchi-Levi et al. 2000).

Until recently, ERP vendors have focused their efforts on providing real-time support for all the manufacturing transactions that form the backbone of the supply chain and left the more focused decision support applications to software companies such as i^2 Technologies and Manugistics. These companies have produced whole families of decision support systems to help analyze the data produced by the ERP-based IS and make better decisions with respect to the supply chain. The interfaces between the standard ERP tools and the new generation of DSSs is where some of the most interesting and valuable research and development is being conducted. As companies improve their support of specific supply chain activities, they can expand the level of integration to include supporting activities from financial operations and human resources.

When taking a supply chain view of its operations, a company must begin with the notion that there are three basic flows within their enterprise: materials, financials, and information. At defined points in the supply chain, data can be collected and stored. The goal of ERP-based ISs in this view is to enable the enterprise to collect and access this information in a timely fashion. This reduces inventory levels, reduces costs, and improves customer service and satisfaction, and each product produced by the enterprise will also have an accompanying information trail that can be tracked and used for planning and estimating lead times. It must be kept in mind that this integrated information flow is available in real time as a basis for decision making.

Much of the recent interest in supply chain management and ERP stems from the e-business models that have been emerging. These models now call for tighter linkages between suppliers and customers, and the ERP tools must support these linkages. This extended supply chain management makes use of the Internet to link with suppliers and customers via standard browsers and TCP/IP protocols. This means that not only will customers be able to track their orders, but companies will be sharing inventory and BOM information with their suppliers. In fact, one of the major unresolved issues in ERP research is the question of inter-ERP linkage. Most vendors have their own standard format, and until there is general agreement within the industry, inter-ERP data sharing will require the development of sophisticated middleware to share data between ISs at companies using different vendors' ERP tools (Simchi-Levi et al. 2000).

4.6.2. Enterprise Tools and Customer Relationship Management (CRM)

As ERP vendors continue to extend the functionality of their packages, one area where there has been tremendous growth has been in the ability of the software to support a company's links to its clients. Interest in better supporting these linkages has led to the development of a niche software market called customer relationship management (CRM) software. CRM supports business functions typically performed by marketing, sales, and service. CRM software is designed to support these functions by helping to identify, select, develop and retain customers. With the recent move to e-business, the traditional view of CRM has changed to look to how CRM software can be combined with electronic commerce technology to gain a competitive advantage. This means that new CRM software helps to move existing processes such as order entry to the Internet and supports new processes such as email-based customer support. Companies are now revising their business strategies to take advantage of Web-enabled CRM software to support Web marketing and customer interaction. Besides supporting sales staff, the new breed of CRM software supports a ''contact center,'' which is a focal point of phone, Web, e-mail, ATM, and kiosk customer interactions. These front-office activities then must be linked in real time to back office functions that fall into the domain of the traditional ERP package.

While the predictions for the maturing ERP market indicate that its growth rate is slowing, the market for CRM software is just starting to explode. One industry analyst has predicted that the market will grow at an annual growth rate of 49% over the period from 1999–2003, when it will reach $16.8 billion in sales (Bowen, 1999). Of this market, the top vendors in 1998 were Siebel Systems, Vantive, and Clarify, who together represent 40% of the CRM tool market. Siebel became the market leader by changing its focus from client/server sales force automation applications to developing an integrated, Web-based CRM suite, with sales, marketing, and customer modules. It has generated more than 50 partnerships with industry giants, such as the recent alliance with IBM. With strategy, their revenues increased from $36 million in 1996 to $329 million in 1998 (Zerega 1999).

Because of the immense potential of the CRM market and because it fits well with their attempts to turn their product lines toward an e-business orientation, all the major ERP vendors are now looking to expand into the front-office domain. ERP vendors are doing this by acquiring CRM vendors, developing alliances with pure CRM vendors, developing their own packages, or as in the case of SAP, a combination of these. In October of 1999, Peoplesoft announced that it was acquiring one of the leading CRM vendors, Vantive. Similarly, Baan has already purchased Aurum Software and has formed an alliance with Cap Gemini to integrate its CRM applications into BaanERP (Goldbaum 1999). Of the top five ERP vendors, Oracle is the only one currently offering a front-office system. It is ahead of the ERP market in developing a suite of Web-enabled CRM applications that integrate easily with its tools for back-office enterprise systems. It has aggressively pursued alliances with hardware vendors such as HP to allow Oracle and HP salespeople to collaborate and share information over the Internet (Stroud 1999). In this way, it cosells its CRM solutions and shares customer data. Oracle also announced in March of 1999 that it would integrate its CRM suite with SAP's ERP tools (Sykes 1999). In contrast, SAP does not currently offer a CRM module for its product, R/3.

SAP is the ERP market leader with the most comprehensive and integrated package of ERP applications. Yet it is seen as being late to move its focus to Web-enabled enterprise application systems. In the fall of 1999, it unveiled its long-awaited MySAP.com and is attempting to use its large client base of active installations for leverage into the CRM market. Given its huge investments in R&D, SAP has never been inclined to purchase niche players in related software markets but has preferred to develop its own tools in-house. With respect to CRM tools, its eventual plan is to package CRM modules as a part of its new MySAP.com architecture for e-business portals (Bowen 1999). It will be browser-based, using Microsoft's Internet Explorer, and will provide seamless access to R/3's enterprise-wide applications. Although SAP has yet to deliver its own CRM modules, it is predicting that CRM-related revenues will eventually outpace revenues from sales of ''pure'' ERP packages (Bowen 1999).

Because of its huge growth potential, the CRM market is very dynamic, with a stream of new entrants and the expansion of current CRM vendors' tools. With respect to ERP packages, the trend is toward the eventual blending of ERP with CRM modules. This will occur as the major ERP vendors continue to turn their focus toward e-business. It is expected that this new amalgam of front-office with back-office applications will lead to a seamless new set of IS development tools called ''e-business suites'' (Goldbaum 1999).

4.6.3. Enterprise Tools and Electronic Commerce (EC)

As the ERP market has matured, ERP vendors have rushed to be the first to redesign their products so that they Web-enable their enterprise tools. The rapidly evolving model of ERP and e-commerce means that users anywhere in the world will be able to access their specific enterprise-wide ISs via a standard, browser-based interface. In this model, the ERP platform defines a new suite of e-business modules that are distributed to suppliers and clients using the standards of the Internet. The ERP tool can thus be used to build an e-business ''portal'' that is designed specifically for everything from small ''e-tailers'' to large multinational corporations. This portal will streamline links with customers, suppliers, employees, and business partners. It will extend the linkage between the enterprise and transorganizational classes of ISs.

One way that this model is emerging is as a turnkey supplier of e-business function support. For example, SAP's new web portal model, MySAP.com, can be used to help a new e-tailer design and build its website. Using an innovative natural language interface, a business person can describe the features he or she wants for the company website and a basic web page will be automatically generated. The website will be further linked with whatever functionality is desired from the back-office enterprise IS, via the Web-enabled suite of modules in MySAP.com. These modules might include processing customer orders, creating and maintaining a corporate catalog for customers to browse, processing bank transactions, monitoring customer satisfaction, and allowing customers to check the status of their orders. The same platform can be used to maintain the company's business-to-business applications, such as eProcurement and information sharing with business partners and clients. For a fee, MySAP.com users can ''rent'' back-office decision support software for forecasting, data mining, and inventory planning and management. These services are delivered over the Internet and users pay on a per-transaction basis. This web portal also serves as the platform for business-employee transactions, helping employees communicate better, monitor their benefit plans, plan for and receive training, and possibly provide more advanced knowledge management support.

5. TOOLS FOR ANALYSIS AND DESIGN OF IS

Managing the development and implementation of new IS is vital to the long-term viability of the organization. In this section, we present and discuss some of the tools and methodologies that have emerged over the last 40 years to help facilitate and manage the development of information systems. The systems development life cycle (SDLC) is introduced, and the tools used within each step of the life cycle are explained. It is important to realize that although professional systems developers have employed these tools and methodologies for the last 20 years with some success, the tools do not guarantee a successful outcome. Utilization of the SDLC will increase the likelihood of development success, but other forces may affect the outcome. Careful attention must also be paid to the needs of users of the system and to the business context within which the system will be implemented. This attention will increase the likelihood of system acceptance and viability.

5.1. The Systems Development Life Cycle

The SDLC was developed as a methodology to aid in the development of information systems. By defining the major activities in the process, the SDLC ensures that all the required tasks are completed, and the time and resources necessary in the different phases can be more precisely estimated. The SDLC separates the tasks that have to be performed when developing an IS into five phases:

1. Systems planning
2. Systems analysis
3. Systems design
4. Systems implementation/testing
5. Systems use and maintenance

Some have referred to this process as a waterfall because one phase naturally leads to the next, in a descending cascade. In business, all information systems become obsolete over time, either functionally or operationally. When the system fails to achieve the desired results, a change in that system is necessary. The existing system may be in the use and maintenance phase; however, it is necessary to move into the systems planning phase in order to initiate a change in the existing system.

During the systems planning phase, an investigation is conducted into the desirability and feasibility of changing the existing system. At this point, the major purpose is to understand the problem and ascertain whether it is cost effective to change it. At the outset of this phase, a systems development team is formed, made up of members of user groups, members of the information technology group, and some members of management. This team typically works together through the entire SDLC. At the end of this phase, a feasibility report is produced. This report should outline the major business needs to be met by the new system, the system's major inputs and outputs, the scope of the actual problem, and an estimate of the time and resources necessary to solve the problem.

The tools used during this phase usually include automated tools that enable team members to model the existing system using a context diagram and an economic analysis using tools that enable the team to conduct traditional financial analysis, such as net present value and return on investment. A determination is made at the end of this phase whether to continue on with the development or terminate the project. Often the project is either too trivial or cannot be solved in a cost-effective manner to warrant continuation. In such cases, the project is terminated at the end of this phase without major expenditures. If the team decides that the project is worth pursuing, they will proceed to the system analysis phase.

In the system analysis phase, some of the planning activities are repeated in order to flush out the details. The analysts must fully understand the entire problem and begin to create a guide to solve it. This involves a much more detailed study of the existing system. The analysts will question users in much more depth to gather data on their exact requirements. The existing system is typically modeled using tools such as data flow diagrams, data dictionaries, and entity relationship diagrams. Each of these tools will be discussed in greater detail in a following section. The modeling tools allow the analyst team to gain a thorough understanding of the existing system. This understanding, when compared with the new system requirements, will generate the new system proposal. This proposal spells out the changes necessary to correct the deficiencies in the existing system. The proposal is typically presented to a management group that is responsible for funding the next step in the process, system design. At this point, management determines whether the project should continue. They will either provide funds for the project to move on to the system design phase or terminate the project. If the project is terminated at this point, only labor costs are incurred because no information technology expenditures have yet been expanded.

In the design phase, the team will determine how user requirements will be met, working from the outputs backward. A new physical model of the proposed system is created. If the analysts are able to determine precisely what outputs the users require, they can then work back through the processes needed to produce those outputs and the input data needed in the process. This provides a blueprint for the design: the formulation of the data files, the specifications for the software, and the configuration of the needed hardware. The data configuration and software and hardware requirements are determined during this phase. The rule of thumb in systems has been to work in this manner because the data and software design drive the purchase of the actual computing equipment. If the hardware were designed first, later it might either prove inadequate for the long-term requirements of the new system or overshoot those requirements so much as to be less cost effective. At this point, nothing has actually been purchased. The new physical design is summarized in a new system proposal, which is presented to management for funding. If management declines to fund the proposal, the design team disbands. If management funds the new proposal, then the project moves on to system implementation.

If the project reaches this phase, it has reached the point of no return. During the implementation phase, the new system design is used to produce a working physical system. The needed pieces of the system are either created or acquired, then installed and tested on the hardware that has been earmarked for this system. Generally, the team begins by creating the test data needed for the new system. The software specifications are then converted into computing code and tested with the test data. The users are trained in the procedures required to use the new system. The site where the system will be installed is prepared, and the new equipment is installed and tested. The entire system typically undergoes final tests before being released into the production environment.

After the system has been installed, the system enters the use and maintenance phase. In order for the system to deliver value to the organization, minor maintenance typically is necessary to correct errors, ensure system currency, enhance features, or adapt hardware. The system will be periodically tested and audited to increase the reliability of the system outputs. When the system ages to a point where it can no longer be easily modified, the SDLC will shift back to the planning phase and the life cycle will begin again.

During each phase of the SDLC, the organization should maintain detailed documentation of the system. This documentation is the accumulation of all the models, analysis and design specifications, and records during system development. Such documentation is invaluable to the next team charged with a development project because it explains all of the steps and assumptions made during the system's development. The team must keep meticulous records of all meetings, findings, and results, which together make up the system document. This document is maintained at the organization as long as the system is in production.

5.2. Systems Development Tools

This section examines the models and methodologies that are used during the different phases of the SDLC.

5.2.1. Feasibility Analysis

Information systems development projects are almost always done under tight budget and time constraints. This means that during the systems planning phase of the SDLC, a feasibility assessment of the project must be performed to ensure that the project is worth the resources invested and will generate positive returns for the organization. Typically, feasibility is assessed on several different dimensions, such as technical, operational, schedule, and economic. Technical feasibility ascertains whether the organization has the technical skills to develop the proposed system. It assesses the development group's understanding of the possible target technologies, such as hardware, software, and operating environments. During this assessment, the size of the project is considered, along with the systems complexity and the group's experience with the required technology. From these three dimensions, the proposed project's risk of failure can be estimated. All systems development projects involve some degree of failure risk; some are riskier than others. If system risk is not assessed and managed, the organization may fail to receive anticipated benefits from the technology, fail to complete the development in a timely manner, fail to achieve desired system performance levels, or fail to integrate the new system properly with the existing information system architecture. Reducing the scope of the system, acquiring the necessary development skills from outside, or unbundling some of the system components to make the development project smaller can reduce these risks.

Operational feasibility examines the manner in which the system will achieve desired objectives. The system must solve some business problem or take advantage of opportunities. The system should make an impact on value chain activities in such a way that it supports the overall mission of the organization or addresses new regulations or laws. The system should be consistent with the operational environment of the organization and should, at worst, be minimally disruptive to organizational procedures and policies.

Another dimension of feasibility, projected schedule feasibility, relates to project duration. The purpose of this assessment is to determine whether the timelines and due dates can be met and that meeting those dates will be sufficient to meet the objectives of the organization. For example, it could be that the system must be developed in time to meet a regulatory deadline or must be completed at a certain point in a business cycle, such as a holiday or a point in the year when new products are typically introduced to market.

Usually, the economic feasibility assessment is the one of overriding importance. In this case, the system must be able to generate sufficient benefits in either cost reductions or revenue enhancements to offset the cost of developing and operating the system. The benefits of the system could include reducing the occurrence of errors associated with a certain procedure; reducing the time involved in processing transactions; reducing the amount of human labor associated with a procedure; or providing a new sales opportunity. The benefits that can be directly measured in dollars are referred to as tangible benefits. The system may also have some benefits that, though not directly quantifiable, assist the organization in reaching its goals. These are referred to as intangible benefits. Only tangible benefits will be used in the final economic feasibility assessment, however. After benefits have been calculated, the tangible costs of the systems are estimated. From a development perspective, these costs include hardware costs, labor costs, and operational costs such as site renovation and employee training. The costs can be divided into two classes: one-time (or sunk) costs and those costs associated with the operation of the new system, or recurring costs. The benefits and costs are determined for a prespecified period of time, usually the depreciable life of the system. This time period could be 1 year, 5 years, or 10 years, depending on the extent of the investment and the financial guidelines used within the organization.

The tangible costs and benefits are then used to determine the viability of the project. Typically, this involves determining the net present value (NPV) of the system. Since the cost of developing the system is incurred before the systems begins to generate value for the organization over the life of the system, it is necessary to discount the future revenue streams to determine whether the project is worth undertaking. The present value of anticipated costs at a future point in time is determined by:

$$\mathrm{PV}_n = \left(\frac{Y_n}{(1 + i)^n}\right)$$

where PV_n is the present value of an anticipated cash inflow or outflow Y_n (Y_n is the monetary value ($) of the cost or benefit) in the nth year into the project's life and i is the interest rate at which future cash flows are to be discounted. The net present value (NPV) of the project is:

$$\mathrm{NPV} = -C_0 + \sum_{t=1}^{T} PV_t$$

where C_0 is the cost of developing the system and putting it into production and T is the useful life of the system.

Other financial analyses that provide useful information as one seeks to determine whether the organization should proceed with the project include return on investment (ROI) and break-even (BE) analysis. ROI is the ratio of the net returns from the project divided by the cash outlays for the project. BE analysis seeks to determine how long it will take before the early cost outlays can be recouped. If the ROI is high and the BE point is small (i.e., reached quickly), the project becomes more desirable because cash flow estimates far into the future are much more difficult to forecast accurately. Table 2 is a simple spreadsheet that was used to determine economic feasibility.

5.2.2. Process Modeling Using Data Flow Diagrams

During the analysis phase, it is critical that the development team gain a good understanding of the existing system so that they can be in a position to know how the current system produces information. This understanding must be at a very low level of detail, at the level of the processes used to produce information. This level of understanding is best obtained by developing models of the existing system. Typically, the existing system is first graphically represented using a diagramming method known as a data flow diagram (DFD). This method provides a simple way to describe the system graphically. The diagrams can be produced quickly by the development team yet can provide a rich level of detail as they are expanded on lower levels. They are also easy for novices to read, which means they can be powerful communication tools among the development team, managers, and users. Finally, they can be decomposed, that is, logically broken down, showing increasing levels of detail without losing their ability to convey meaning to users and management.

DFDs are drawn with just four symbols: data sources and sinks, processes, data flows and data stores (see Figure 5). Data sources are entities that provide data to the system (a source) or receive information from the system (a sink). These sources lie outside the boundary of the system and are not active participants in the processing that occurs within the system. A process is any activity that converts data into information. Generally, processes are named with a verb of the action in the process and are numbered at the top of the symbol. Numbers are useful for cross-referencing processes on lower-level diagrams to the activities in higher-level diagrams. Processes are interconnected with other processes by data flows. Data flows are symbols that represent data in motion. Each data flow is named with a noun that describes the data represented in the flow. A data flow must be attached to a process at some point, either as a source or as a sink. A data store is a representation of the data at rest. Examples of data stores could include electronic data files, a database table, or a physical file folder.

When the symbols are used in combination, they trace the movement of data from its origin, the source, through all of the steps that transform that data into information for the users, to its eventual destination, a sink. The feature of DFDs that make it so useful in the context of system development is that the diagrams are decomposable. Every system must be understood at a high level of detail.

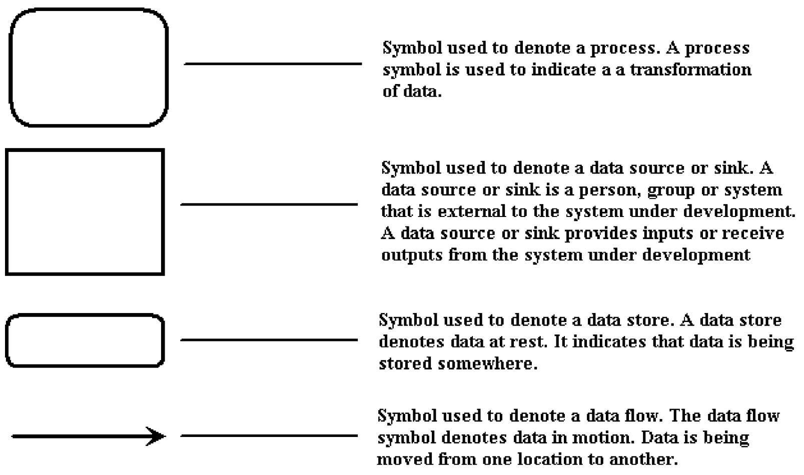

Symbol used to denote a process. A process symbol is used to indicate a a transformation of data.
Symbol used to denote a data source or sink. A data source or sink is a person, group or system that is external to the system under development. A data source or sink provides inputs or receive outputs from the system under development
Symbol used to denote a data store. A data store denotes data at rest. It indicates that data is being stored somewhere.
Symbol used to denote a data flow. The data flow symbol denotes data in motion. Data is being moved from one location to another.

Figure 5 Data Flow Diagram Symbols.

DFDs allow the same system to be examined at many levels of detail, from the context diagram level through whatever level the development team chooses to stop at.

The lowest level of detail in a DFD is called a context diagram, also called a "black box" diagram, because it shows only the sources and sinks and their connection to the system through aggregate data flows. The system itself is represented by only one process. That central process will be decomposed into many subsystems in the Level 0 diagram, which shows the system's major processes, data flows, and data stores at a higher level of detail. Figure 6 shows a level 0 diagram for a simple payroll processing system. Each process is numbered, and each represents a major information-processing activity. When the diagram moves to Level 1, each subsystem from Level 0 is individually decomposed. Thus, process 1.0 on the DFD may be functionally subdivided into processes 1.1, 1.2, 1.3, and so on for as many steps as necessary. In Figure 7, we show a Level 1 diagram that decomposes the third process from the Level 0 diagram in Figure 6. As the diagrams are decomposed further, at Level 2 the steps in process 1.1 may be decomposed into subprocesses 1.1.1, 1.1.2, 1.1.3, etc. Each Level 2 DFD will expand on a single process from Level 1. These diagrams can continue to be leveled down until the steps are so simple that they cannot be decomposed any further. This set of DFDs would be referred to as primitive diagrams because no further detail can be gained by decomposing further. The power of this tool is that while it is conceptually simple, with only four symbols, it is able to convey a great degree of detail from the nesting of the decompositions. For systems analysis, this detail is invaluable for an understanding of existing system functionality and to understand the processes.

5.2.3. *Structured English*

The next step in process modeling, usually performed after the creation of the DFDs, is to model process logic with Structured English, a modified form of the English language used to express information system process procedures. This tool uses the numbering convention from the DFDs to designate the process under examination, then semantically describes the steps in that DFD process. Structured English is used to represent the logic of the process in a manner that is easy for a programmer to turn into computer code. In fact, similar tools were called "pseudo-code" to indicate that they were closely related to the design of computer programs. At that point they were used

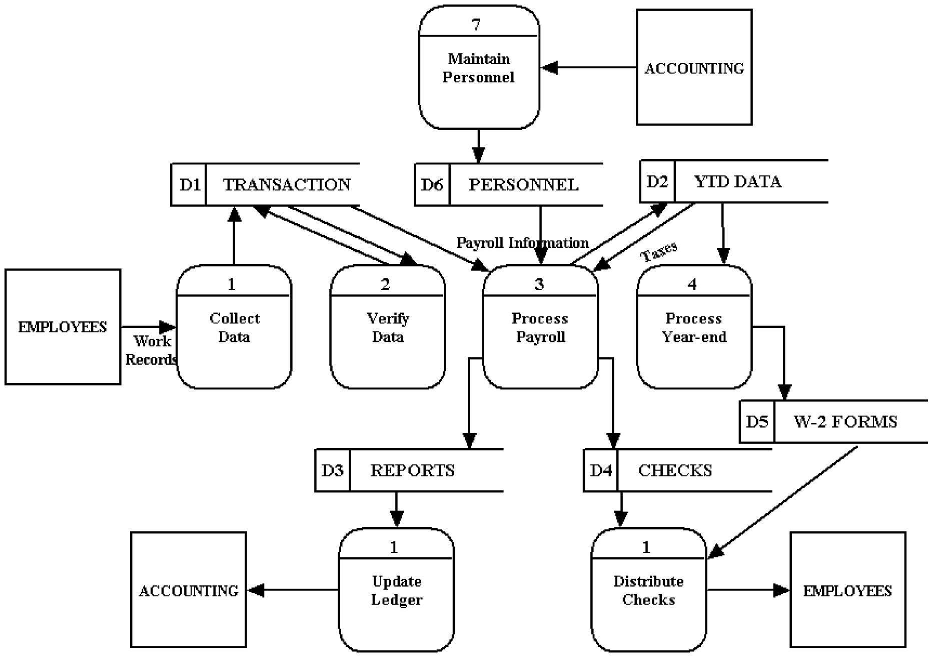


Figure 6 Data Flow Diagram for a Simple Payroll System.

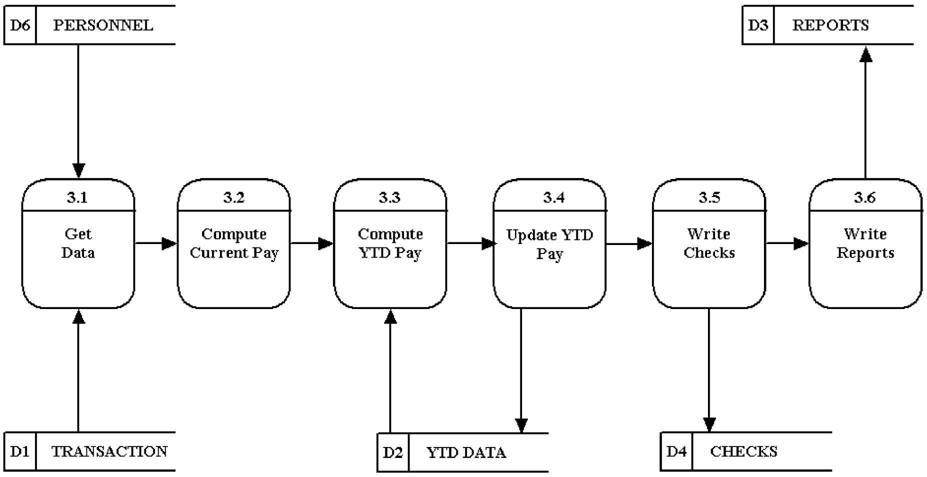


Figure 7 An Explosion of the Process Payroll Process.

to describe individual programs, not entire information systems. As a tool, Structured English representations of processes help in the implementation of the system, making it much easier to develop specifications that can be handed off to those who will develop the programs. Generally, Structured English is generic enough that it can used to develop programs in most programming languages.

The steps in developing Structured English versions of a process are fairly straightforward. First, describe the steps in the process being examined, beginning with a strong verb. The verb modifies a noun, which is generally a data structure, data element, or data flow. Any given statement should include no adjectives or adverbs. While there is no standard for Structured English, it represents basically three types of processes: sequence, conditional statements, and iterations. Sequences are lists of statements where one statement simply follows another. An example of a sequence is:

```
Process 1.1: Update Master File
DO
     OPEN Master File
     READ Record
     MATCH Key to Master File Key
     REPLACE Master record with New Master Record
     CLOSE Master File
END DO
```

The second type of Structured English process is a conditional statement. A case statement is declared, followed by a group of possible actions that will be executed after the evaluation of the case. Programmers refer to this as an "if/then" statement. An example of a conditional statement is:

```
Process 2.3: Order Inventory
BEGIN IF
     IF Quantity on hand is less than Reorder quantity
     THEN generate New order
     ELSE do nothing
END IF
```

The last type of Structured English process is an iteration. Iterations are special cases of the conditional statement, where steps are repeated until a stated condition is satisfied. This type of process is also sometimes referred to as a repetition. An example of an iteration is:

```
Read Order File
WHILE NOT End Of File DO
     BEGIN IF
          READ Order Address
          GENERATE Mailing Label
     END IF
END DO
```

Structured English has been found to be a good supplement to DFDs because, in addition to helping developers at the implementation stage, they have been found to be a useful way to communicate processing requirements with users and managers. They are also a good way to document user requirements for use in future SDLC activities.

5.2.4. ERD/Data Dictionaries

During the analysis phase of the SDLC, data are represented in the DFDs in the form of data flows and data stores. The data flows and data stores are depicted as DFD components in a processing context that will enable the system to do what is required. To produce a system, however, it is necessary to focus on the data, independent of the processes. Conceptual data modeling is used to examine the data "at rest" in the DFD to provide more precise knowledge about the data, such as definitions, structure, and relationships within the data. Without this knowledge, little is actually known about how the system will have to manipulate the data in the processes. Two conceptual data modeling tools are commonly used: entity relationship diagrams (ERDs), and data dictionaries (DDs). Each of these tools provides critical information to the development effort. ERDs and DDs also provide important documentation of the system.

An ERD is a graphical modeling tool that enables the team to depict the data and the relationships that exist among the data in the system under study. Entities are anything about which the organization stores data. Entities are not necessarily people; they can also be places, objects, events, or concepts. Examples of entities would be customers, regions, bills, orders, and divisions. Entity types are collections of entities that share a common property or characteristic, and entity instances are single occurrences of an entity type. For example, a CUSTOMER is an entity type, while Bill Smith is an entity instance. For each entity type, one needs to list all the attributes of the entity that one needs to store in the system. An attribute is a named property or characteristic of an entity type. For example, the attributes for a CUSTOMER could be customer number, customer name, customer address, and customer phone number. One or a combination of the attributes uniquely identifies an instance of the entity. Such an attribute is referred to as a *Candidate key,* or *identifier*. The candidate key must be isolated in the data: it is the means by which an entity instance is identified.

Associations between entity types are referred to as *relationships*. These are the links that exist between the data entities and tie together the system's data. An association usually means that an event has occurred or that some natural linkage exists between the entity types. For example, an order is associated with an invoice, or a customer is associated with accounts receivable instances. Usually the relationship is named with a meaningful verb phase that describes the relationship between the entities. The relationship can then be "read" from one entity to the other across the relationship. Figure 8 shows a simple ERD for a system in a university environment.

Once an ERD has been drawn and the attributes defined, the diagram is refined with the degree of the relationships between the entities. A unary relationship is a one-to-one or one-to-many relationship within the entity type—for example, a person is married to another person, or an employee manages many other employees. A binary relationship is a one-to-one, one-to-many, or many-to-many relationship between two entity types—for example, an accounts receivable instance is associated with one customer, or one order contains many items. A tenary relationship is a simultaneous relationship between three entity types—for example, a store, vendor, or part, all sharing stock-on-hand. Once the degree of the relationships have been determined, the ERD can be used to begin the process of refining the data model via normalization of the data. Normalization is necessary to increase the stability of the data so that there are fewer data updating and concurrency anomalies. The normalization process is described in detail in every database management textbook (e.g., McFadden et al., 1999).

The ERD is an excellent tool to help understand and explore the conceptual data model. It is less useful, however, for documentation purposes. To address the documentation requirements, a data dictionary (DD) is used. The DD is a repository of all data definitions for all the data in the system. A DD entry is required for each data flow and data store on every DFD and for each entity in the ERDs. Developing a DD entry typically begins with naming the data structure being described, either by flow name, file name, or entity type. The list of attributes follow, with a description of the attribute, the attribute type, and the maximum size of the attribute, in characters. The attribute type is specified (e.g., integer, text, date, or binary). In the case of a data file, the average size of the file is indicated.

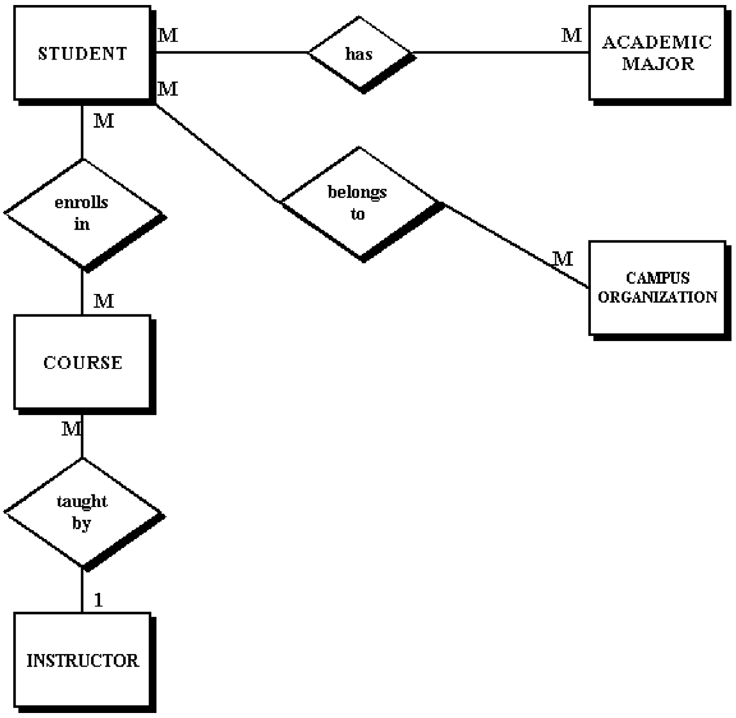


Figure 8 An Entity-Relationship Diagram.

For a data flow, the minimum, maximum, and average number of occurrences of that flow in a business day is determined and listed with the attributes. For the entity type, the entry will note the various associations that the entity type has with the other entities. An example of a DD entry is:

```
        Data Stores

Name:   Customer Mailing List

Key     Customer Number

Size    250,000
```

	Name	*Definition*	*Type*	*Size*
Attributes:	Cust_Num	Unique customer Number	Integer	5
	Cust_Name	Customer name	Text	25
	Cust_Add	Customer Address	Text	30

5.2.5. Gantt/PERT Diagram

Gantt charts and PERT diagrams are used in the planning phase of the SDLC to schedule and allocate time to key activities in the development process. Gantt charts are two-dimensional charts that show major activities on one axis and a time line on the other. The duration of the activities listed is designated with a solid bar under the time line, next to the name for that activity. The time lines for the bars can overlap because certain activities may overlap during the development process. An

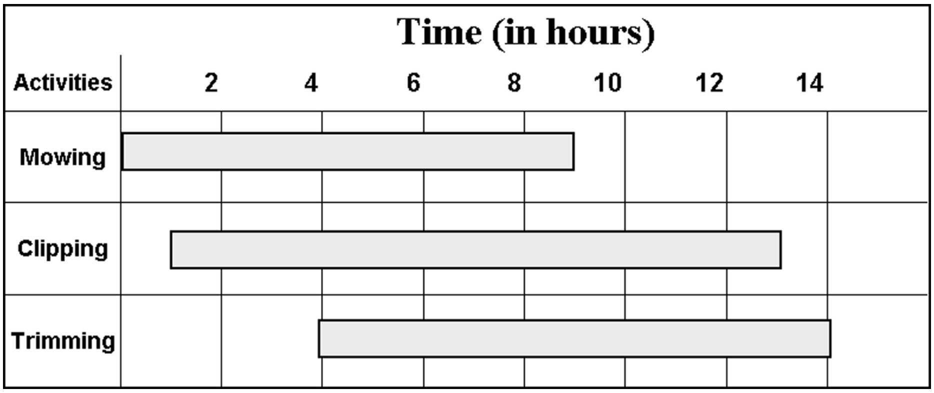


Figure 9 A Gantt Chart for a Simple Lawn-Maintenance Task.

example of this would be program code development and site preparation during system implementation. Figure 9 shows a Gantt chart for a lawn maintenance job with two workers assigned to each activity (Wu and Wu 1994).

PERT diagrams serve the same function as Gantt charts but are more complex and more detailed. They are essential tools for managing the development process. PERT is an acronym for Program Evaluation and Review Technique and was developed by the United States military during World War II to schedule the activities involved in the construction of naval vessels. Since then, PERT has found its way into production environments and structured system development because it allows one to show the causal relationships across activities, which a Gantt chart does not depict. PERT diagrams show the activities and then the relationships between the activities. They also show which activities must be completed before other activities can begin and which activities can be done in parallel. Figure 10 shows an example of a PERT diagram for the lawn maintenance job (Wu and Wu 1994).

Which planning tool is chosen depends upon the complexity of the system being developed. In the case of a systems development effort that is relatively small and simple and where the design can be completed quickly, the Gantt chart is preferable. Where the design is large, complex, and involves many activities that must be carefully coordinated, the PERT diagram is best.

5.2.6. JAD/RAD

Rapid application deployment (RAD), joint application deployment (JAD), and prototyping are approaches used in system development that bypass parts of the SDLC in an effort to speed up the

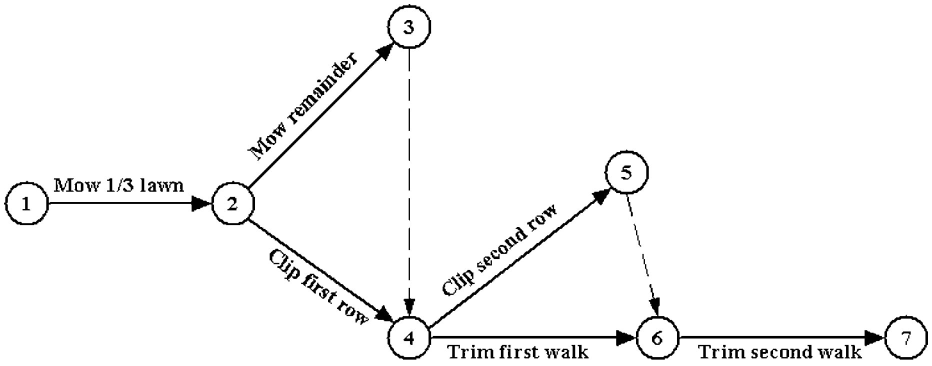


Figure 10 A PERT Diagram for a Simple Lawn-Maintenance Task.

development of the system. Traditionally, the development team will use the analysis phase of the SDLC to determine how the existing system functions and assess what the users of the new system would like to see in that system. RAD and JAD methods have emerged in an effort to remove or greatly reduce the time spent in the analysis phase of development. They are designed for development efforts that jump from planning directly to implementation, such as when software is acquired from a third party. RAD and JAD eliminate the detailed study required during the analysis phase and in doing so speed up user requirements analysis.

JAD is a structured approach whereby users, managers, and analysts work together for several days of intensive meetings to specify or review systems requirements. The coordination of these groups during these meetings determines whether the effort is successful. The groups must agree to work together to iron out any differences and have to work closely together to elicit requirements that might take months to emerge using traditional methods. RAD works by compressing the SDLC steps into four stages: planning, design, development, and cutover. Working only on the systems functions and user interface characteristics shortens planning and design. Then design and development work in an iterative fashion as the model is designed and implemented, then refined through further design and implementation until it produces the features that users require. RAD entails the use of a prototyping tool, which is a software tool that allows the rapid development of a new physical model that can be used and tested by the users. The distinction between prototyping and RAD as development approaches is that in many cases prototypes are never utilized in production environments. They are used to elicit requirements, but the actual prototype is then discarded in favor of a physical model produced with traditional SDLC methods. In RAD, the designed model will be implemented once it has reached a stage where no further refinement is necessary. That change would occur in the implementation phase.

The advantage of these approaches is in the speed with which systems can be developed. For smaller systems, or systems whose failure would not significantly affect the ability of the organization to engage in commerce, these approaches work very well. For systems that handle a significant volume of data, or where security and controls are a concern, these approaches are not advisable. The time saved using these approaches is usually gained at the expense of the attention paid to security, controls, and testing when the traditional development approach is used. Time is the usual hedge against the risk of systems failure: time in analysis and time in design. That time is sacrificed in these approaches.

5.2.7. CASE

Another way to speed systems development is to use computer aided software engineering (CASE) tools. In contrast to RAD or JAD, CASE tools are used with the traditional SDLC methodology. CASE aids the SDLC by using automation in place of manual methods. CASE tools support the activities in the SDLC by increasing productivity and improving the overall quality of the finished product. There are basically three different types of CASE tools. Upper CASE supports the planning, analysis and design phases of the SDLC. Upper CASE usually has planning and scheduling tools as well as DFD generators and data dictionary support. Lower CASE supports the latter parts of the SDLC, such as maintenance and implementation. Last come cross-life cycle CASE toolboxes, which support all the SDLC activities, from planning through maintenance.

There are many compelling reasons to use CASE tools. CASE tools can greatly shorten development times without sacrificing system quality. The tools usually enforce consistency across different phases by providing automated consistency checking between DFDs, ERDs, and the data dictionary to ensure that the attributes and data structures have been named and defined consistently across all of them. This, in turn, can increase the productivity of the SDLC team because a data store defined in a DFD can be automatically inserted as an entry in the data dictionary. This, in turn, can increase the consistency of the development effort across projects.

The disadvantages of CASE mostly revolve around cost. CASE tools tend to be very expensive, both in monetary and in resource terms. CASE tools cost a lot, and learning to use them can be very time consuming because of the number of features and the functionality that they provide. They also place great demands on the hardware in terms of processing needs and storage requirements at the server and workstation level. Essentially, all they do is automate manual processes, so the decision to use CASE hinges on the traditional tradeoff between labor costs and automation costs.

6. CONCLUSION

Over the past 35 years, tremendous strides have been made in management information systems, contributing greatly to organizations' abilities to grow, provide quality outputs, and operate efficiently. These strides have been fostered by continually improving hardware, advances in software tools for building ISs, formal methods for IS development, and progress in understanding the management of ISs. There is no sign that these trends are abating. Nor is there any sign that an organization's needs

TABLE 2 ABC Company Feasibility Analysis: Database Server Project

	Year 0	Year 1	Year 2	Year 3	Year 4	Year 5	Totals
Net Benefits	0	75,000	75,000	75,000	75,000	75,000	
Discount Rate	1.0000	0.90909091	0.82644628	0.7513148	0.68301346	0.62092132	
PV of Benefits	0	68,182	61,983	56,349	51,226	46,569	
NPV of all Benefits	0	68,182	130,165	186,514	237,740	284,309	$ 284,309
Sunk Costs	$ (100,000)						
	Year 0	Year 1	Year 2	Year 3	Year 4	Year 5	Totals
Net Costs	0	(22,000)	(21,000)	(23,000)	(24,000)	(25,000)	
Discount Rate	1.0000	0.90909091	0.82644628	0.7513148	0.68301346	0.62092132	
PV of Costs	0	(20,000)	(17,355)	(17,280)	(16,392)	(15,523)	
NPV of all Costs	0	(20,000)	(37,355)	(54,636)	(71,028)	(86,551)	$ (186,551)
Overall NPV							$ 97,758
Overall ROI							52%
Break Even Analysis							
Yearly NPV Cashflow	(100,000)	48,182	92,810	131,878	166,712	197,758	
Overall NPV Cashflow	(100,000)	(51,818)	40,992	172,870	339,582	537,340	

Break even is between year 1 and 2.

for ISs will disappear. Thus, even though their platforms, delivery methods, interfaces, and features will change, information systems will persist as indispensable elements of tomorrow's organizations, continuing to satisfy record keeping and reporting needs.

Integration will be a keynote as we look ahead. Integration is a collaborative interplay of components. It is operative at several levels. At the tool level, we should see a trend towards integration of traditionally distinct functions (Watabe et al., 1991). The resultant tools will enable more rapid IS development and the construction of systems having features not readily achieved with conventional means. At the IS level, we should expect to see a trend toward integration across IS classes. For instance, local ISs will increasingly be built to plug into (i.e., operate on the data of) functional and enterprise ISs; the modular ERP tools are designed for cross-functional IS integration; enterprise-wide ISs will increasingly exhibit transorganizational features. Within a very few years, all major ERP tools will be Internet-based and oriented toward collaborative activity.

At the level of business computing systems, the integration trend will manifest itself in efforts to incorporate decision support functionality into conventional management information systems. IS record keeping and reporting will still be essential for operational control, regulatory compliance, and legal purposes, but their records can be further leveraged for decision support. This can already be seen in ISs equipped with ad hoc query facilities. The use of IS records to build data warehouses that are subject to online analytical processing and data mining is another example in this direction. ERP vendors will increasingly offer decision support modules that work with their tools for building enterprise ISs (Holsapple and Sena 1999). Websites that handle transactions (create/update records and produce transaction reports) will increasingly incorporate facilities to support the decisions that underpin those transactions (Holsapple and Singh 2000).

At an organizational level, the rise of transorganizational ISs has been an important factor enabling and facilitating the appearance of network organizations and virtual corporations (Ching et al. 1996). Advances in supply chain, customer relationship, electronic commerce, and collaborative computing tools will add impetus to this trend. Its business driver is the increasingly dynamic and competitive nature of the global marketplace, in which collaboration and agility are vital. Dynamic, open business models are replacing those characterized by static trading partners.

Artificial intelligence technologies have long been recognized in the DSS realm, particularly in the guise of expert systems (Bonczek et al. 1980). It is likely that tools for building ISs will increasingly employ artificial intelligence technologies. This will lead to ISs that interact with users via speech (based on speech recognition/synthesis and natural language technologies), are self-maintaining (based on automated reasoning technologies), and can modify themselves in light of their experiences (based on machine learning and automatic program-generation technologies).

Application service providers will grow in importance as developers and hosts of information systems, particularly for small and mid-sized firms. These are firms that need an IS (e.g., for human resources functions) but cannot afford the infrastructure. For a rental fee, an application service provider (ASP) operates ISs for other companies on its own computing facilities (i.e., servers). ASP customers access their ISs via the Internet or private networks. ASPs specialize in various ways, such as hosting ISs for a particular industry (e.g., health care) or a particular class of ISs (e.g., transorganizational).

Information systems have become the centerpiece of the much-heralded Information Age. However, this 50-year era is rapidly being engulfed by the emerging knowledge economy, populated by knowledge workers and knowledge-based organizations (Holsapple and Whinston 1987). Information is being subsumed within the far richer notion of knowledge. The technology-driven aspect of this knowledge economy is variously known as the digital economy or the network economy, and its fabric is being defined by a multitude of electronic commerce initiatives (Holsapple et al. 2000). To be competitive in the knowledge economy, organizations (both traditional and virtual) strive to ensure that throughout their operations the right knowledge is available to the right processors (human and computer-based) at the right times in the right presentation formats for the right cost. Along with decision support systems, information systems built with tools such as those described here have a major role to play in making this happen.

REFERENCES

Bancroft, N., Seip, H. and Sprengel, A. (1998), *Implementing SAP R/3: How to Introduce a Large System into a Large Organization,* 2nd Ed., Manning, Greenwich, CT.

Barquín, R. C., and Edelstein, H. (1997), *Building, Using, and Managing the Data Warehouse,* Data Warehousing Institute Series, Prentice Hall PTR, Upper Saddle River, NJ.

Barry, M. (1995), "Getting a Grip on Data," *Progressive Grocer,* Vol. 74, No. 11, pp. 75–76.

Bonczek, R., Holsapple, C., and Whinston, A. (1980), "Future Directions for Decision Support Systems," *Decision Sciences,* Vol. 11, pp. 616–631.

Boudreau, M., and Robey, D. (1999), "Organizational Transition to Enterprise Resource Planning Systems: Theoretical Choices for Process Research," in *Proceedings of Twentieth International Conference on Information Systems* (Charlotte, NC), pp. 291–300.

Bowen, T. (1999), "SAP is Banking on the CRM Trend," *Infoworld,* November, pp. 74–76.

Brookshear, J. G. (1999), *Computer Science: An Overview,* Addison-Wesley, Reading, MA.

Brown, C., and Vessey, I. (1999), "ERP Implementation Approaches: Towards a Contingency Framework," in *Proceedings of Twentieth International Conference on Information Systems,* (Charlotte, NC), pp. 411–416.

Brown, S. (1995), "Try, Slice and Dice," *Computing Canada,* Vol. 21, No. 22, p. 44.

Caldwell, B., and Stein, T. (1998), "Beyond ERP: The New IT Agenda," *Information Week,* November 30, pp. 30–38.

Castro, E. (1999), *HTML 4 for the World Wide Web: Visual QuickStart Guide,* Peachpit Press, Berkeley, CA.

Ching, C., Holsapple, C., and Whinston, A. (1996), "Toward IT Support for Coordination in Network Organizations," *Information and Management,* Vol. 30, No. 4, pp. 179–199.

Curran, T., and Ladd, A. (2000), *SAP R/3 Business Blueprint: Understanding Enterprise Supply Chain Management,* 3rd Ed., Prentice Hall, PTR, Upper Saddle River, NJ.

Davenport, T. (1998), "Putting the Enterprise into the Enterprise System," *Harvard Business Review,* Vol. 76, July/August, pp. 121–131.

Davis, B. (1999), "PeopleSoft Fill out Analysis Suite," *Information Week,* Manhasset, RI, July 26.

Deutsch, C. (1998), "Software That Can Make a Grown Company Cry," *The New York Times,* November 8.

Doane, M. (1997), *In the Path of the Whirlwind: An Apprentice Guide to the World of SAP,* The Consulting Alliance, Sioux Falls, SD.

Eliason, A., and Malarkey, R. (1999), *Visual Basic 6: Environment, Programming, and Applications,* Que Education & Training, Indianapolis, IN.

Elmasri, R., and Navathe, S. B. (2000), *Fundamentals of Database Systems,* 3rd Ed., Addison-Wesley, Reading, MA.

Fogarty, K. (1994), "Data Mining Can Help to Extract Jewels of Data," *Network World,* Vol. 11, No. 23.

Francett, B. (1994), "Decisions, Decisions: Users Take Stock of Data warehouse Shelves," *Software Magazine,* Vol. 14, No. 8, pp. 63–70.

Goldbaum, L. (1999), "Another Day, Another Deal in CRM," *Forbes Magazine,* October, pp. 53–57.

Hackathorn, R. D. (1995), *Web Farming for the Data Warehouse,* Morgan Kaufmann Series in Data Management Systems, Vol. 41, Morgan Kaufmann, San Francisco, CA.

Hall, M. (1998), *Core Web Programming,* Prentice Hall PTR, Upper Saddle River, NJ.

Holsapple, C. (1995), "Knowledge Management in Decision Making and Decision Support," *Knowledge and Policy,* Vol. 8, No. 1, pp. 5–22.

Holsapple, C., and Sena, M. (1999), "Enterprise Systems for Organizational Decision Support," in *Proceedings of Americas Conference on Information Systems* (Milwaukee, WI), pp. 216–218.

Holsapple, C., and Singh, M. (2000), "Electronic Commerce: Definitional Taxonomy, Integration, and a Knowledge Management Link," *Journal of Organizational Computing and Electronic Commerce,* Vol. 10, No. 3.

Holsapple, C., and Whinston, A. (1987), "Knowledge-Based Organizations," *The Information Society,* Vol. 5, No. 2, pp. 77–90.

Holsapple, C. W., and Whinston, A. B. (1996), *Decision Support Systems: A Knowledge-Based Approach,* West, Mineapois/St. Paul.

Holsapple, C. W., Joshi, K. D., and Singh, M. (2000), "Decision Support Applications in Electronic Commerce," in *Handbook on Electronic Commerce,* M. Shawn, R. Blanning, T. Strader, and A. Whinston, Eds., Springer, Berlin, pp. 543–566.

Jetly, N. (1999), "ERP's Last Mile," *Intelligent Enterprise,* Vol. 2, No. 17, December, pp. 38–45.

Kirkpatrick, D. (1998), "The E-Ware War: Competition Comes to Enterprise Software," *Fortune,* December 7, pp. 102–112.

Main, M., and Savitch, W. (1997), *Data Structures and Other Objects Using C++,* Addison-Wesley, Chicago.

Marion, L. (1999a), "Big Bang's Return," *Datamation,* October, pp. 43–45.

Marion, L. (1999b), "ERP Transactions from Anywhere," *Datamation,* August, pp. 65–69.

Mattison, R. (1996), "Warehousing Wherewithal," *CIO,* Vol. 9, No. 12, pp. 58–60.

McFadden, F. R., Hoffer, J. A., and Prescott, M. B. (1999), *Modern Database Management,* 5th Ed., Addison-Wesley, Reading, MA.

McLeod, R. J. (1998), *Management Information Systems,* 7th Ed., Prentice Hall, Upper Saddle River, NJ.

Norris, G., Wright, I., Hurley, J. R., Dunleavy, J., and Gibson, A. (1998), *SAP: An Executive's Comprehensive Guide,* John Wiley & Sons, New York.

Rob, P., and Coronel, C. (1997), *Database Systems: Design, Implementation, and Management,* Course Technology, Cambridge, MA.

Simchi-Levi, D., Kaminsky, P., and Simchi-Levi, E. (2000), *Designing and Managing the Supply Chain,* Irwin/McGraw-Hill, Boston.

Singh, H. (1998), *Data Warehousing: Concepts, Technologies, Implementations, and Management,* Prentice Hall PTR, Upper Saddle River, NJ.

Stedman, C. (1999a), "ERP Guide: Vendor Strategies, Future Plans," *Computerworld,* July 19, pp. 62–64.

Stedman, C. (1999b), "Update: Failed ERP Gamble Haunts Hershey," *Computerworld,* November 18, pp. 34–37.

Stroud, J. (1999), "Oracle Suite Integrates ERP, E-Business Apps," *Internetweek,* October 4, pp. 1–2.

Stroustrup, B. (1994), *The Design and Evolution of C++,* Addison-Wesley, Reading, MA.

Stroustrup, B. (1997), *The C++ Programming Language,* Addison-Wesley, Reading, MA.

Sykes, R. (1999), "Oracle Readies Customer Apps for SAP R/3," IDG News Service.

Watabe, K., Holsapple, C., and Whinston, A. (1991), "Solving Complex Problems via Software Integration," *Journal of Computer Information Systems,* Vol. 31, No. 3, pp. 2–8.

Welti, N. (1999), *Successful SAP R/3 Implementation: Practical Management of ERP Projects,* Addison-Wesley Longman, Reading, MA.

Wu, S., and Wu, M. (1994), *Systems Analysis and Design,* West, Minneapolis/St. Paul.

Zerega, B. (1999), "CRM Rise to the Top," *Red Herring Magazine,* July, pp. 41–44.