CHAPTER

14

# Construct, Deliver, and Maintain Systems Project*

This chapter covers the final three phases of the systems development life cycle (SDLC). The chapter first examines the many activities associated with in-house development. These activities fall conceptually into two categories: (1) construct the system and (2) deliver the system. Through these activities, systems selected in the project initiation phase (discussed in Chapter 13) are designed in detail and implemented. This involves creating input screen formats, output report layouts, database structures, and application logic. Finally, the completed system is tested, documented, and rolled out to the user.

The chapter then examines the increasingly important option of using commercial software packages. The majority of companies today, particularly smaller firms and large firms with standardized information needs, employ prewritten software systems rather than develop in-house systems from scratch. Conceptually, the commercial software approach also consists of construct and delivery activities. In this section, we examine the pros, cons, and issues involved in selecting off-the-shelf systems.

Finally, the chapter addresses the important activities associated with systems maintenance. This stage in the SDLC carries significant financial and operational risks that are of particular importance to management, accountants, and auditors. A trend in systems development, maintenance, and operation within organizations is to outsource part of or all system activities. The SDLC does not change in such a case, but its component parts are provided by an outsourcing provider. Therefore, special controls should be introduced to prevent increasing costs, decreasing functionality of systems, and loss of strategic advantage.

## *Learning Objectives*

After studying this chapter, you should:

- Be able to identify the sequence of events that constitute the in-house development phase of the SDLC.
- Be familiar with the tools used to improve the success of system construction and delivery activities, including prototyping, CASE tools, and the use of PERT and Gantt charts.
- Understand the distinction between the structured and object-oriented design approaches.
- Understand the use of multilevel DFDs in the design of business processes.
- Be familiar with the different types of system documentation and the purposes they serve.
- Recognize the role of accountants in the construct and delivery of systems.
- Understand the advantages and disadvantages of the commercial software option and be able to discuss the decision-making process used to select commercial software.

*This chapter was co-authored by Jiri Polak, PhD, Deloitte & Touche, and Vojtech Merunka, PhD, Deloitte & Touche.

# In-House Systems Development

Organizations usually acquire information systems in two ways: (1) they develop customized systems in-house through formal systems development activities, and/or (2) they purchase commercial systems from software vendors. Numerous commercial vendors offer high-quality, general-purpose information systems. These vendors primarily serve organizations with generic information needs. Typically, their client firms have business practices so standardized that they can purchase predesigned information systems and employ them with little or no modifications. However, many organizations require systems that are highly tuned to their unique operations. These firms design their own information systems through in-house systems development activities.

Although each approach has advantages and disadvantages, they are not mutually exclusive options. A firm may satisfy some of its information systems needs by purchasing commercial software and developing other systems in-house. This section is concerned with the in-house systems development component of Figure 14-1.

## TOOLS FOR IMPROVING SYSTEMS DEVELOPMENT

Systems development projects are not always success stories. In fact, by the time they are implemented, some systems are obsolete or defective and must be replaced. Estimates hold that up to 25 percent of all systems projects fail. That is, they are terminated prematurely and never implemented, or they must be redesigned within six months of implementation. Historically, three problems that account for most systems failures have plagued the SDLC.

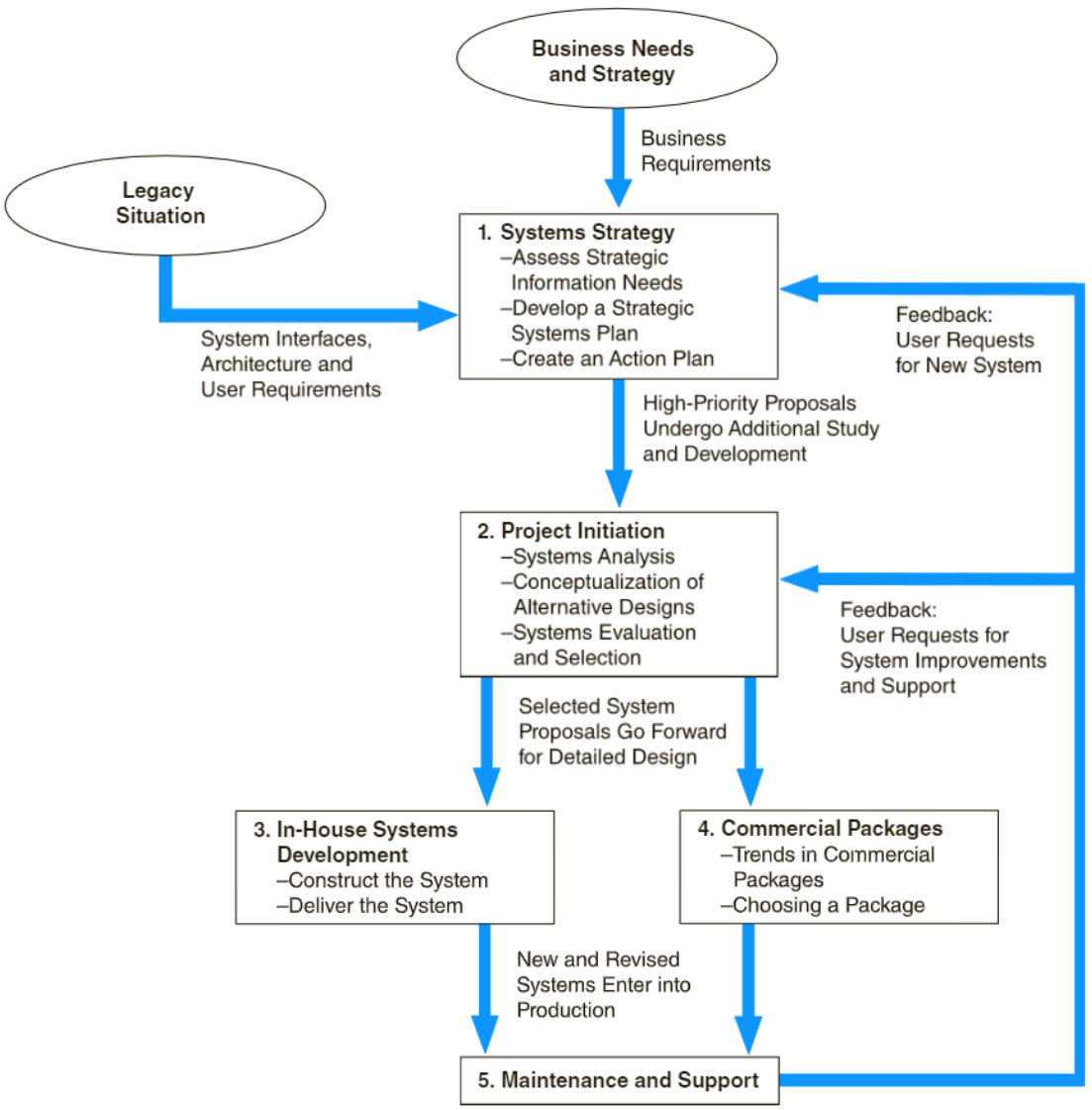
1. *Poorly specified systems requirements.* Systems development is not a precise science. The process involves human communications and the sharing of ideas between users and systems professionals. This information exchange is often imperfect. Mistakes are made in identifying problems and needs, new ideas emerge as the true nature of the problem unfolds, and people simply change their minds about what they really want and need from the system.

   Because of this uncertainty, the SDLC tends not to be a smooth, linear process, where one stage is completed before the next one begins. In reality, the process is iterative or cyclical. For example, it is not uncommon for a systems designer to return to the analysis stage from the construct stage to gather additional information as his or her perception of the problem changes.

   The cyclical nature of this process results in time-consuming false starts, much repeated work, and pressure from all fronts to get the job done. Too often, the result is a system that is poorly designed, over budget, and behind schedule.
2. *Ineffective development techniques.* The problems cited above are amplified by ineffective techniques for presenting, documenting, and modifying systems specifications. In the worst-case scenario, systems development tools are simply paper, pencils, rulers, templates, and erasers. The use of computer-based graphics software that permits original designs and changes to be made electronically considerably improves the situation. Nevertheless, days or even weeks of work may need to be redone because of a change in a system's specifications.
3. *Lack of user involvement in systems development.* The major cause of systems failure is the lack of end-user involvement during critical development stages. At one time, computer systems development was thought to be the exclusive domain of the systems professionals. During this period, users (including accountants) abdicated their traditional responsibility for systems design. Too often, this led to business problems because system designs reflected the analyst's perception of information needs rather than the perception of accountants and other users. Systems often lacked adequate controls and audit trails.

Today, we recognize that user involvement in a system's development is the key to its ultimate success. However, achieving competent user involvement is still difficult to accomplish. There are two reasons for this: (1) users tend to become discouraged when they discover the amount of time they must actually invest, and (2) communication between end users and systems professionals is generally not fluent. It is often said that these groups speak different languages. Each tends to

FIGURE 14-1 SYSTEMS DEVELOPMENT LIFE CYCLE





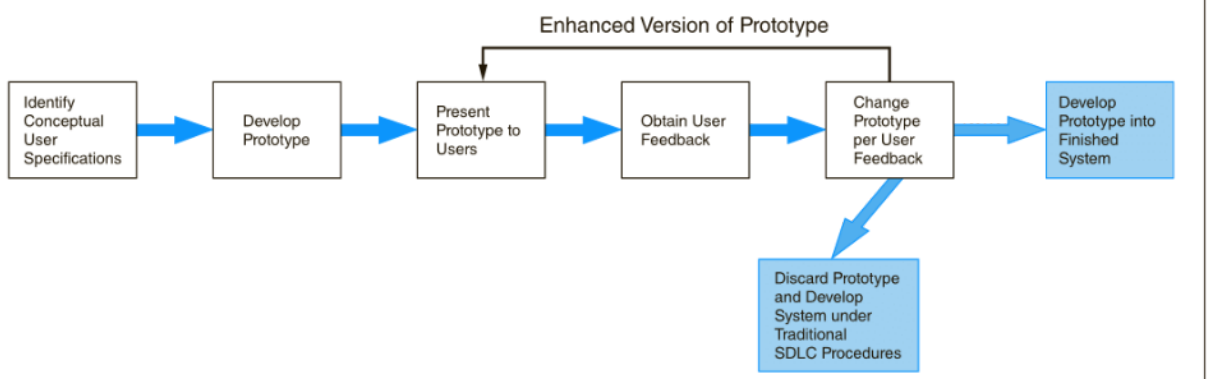

resort to its own jargon when communicating with the other. Therefore, much time is spent identifying user problems and needs and formulating acceptable solutions. Miscommunications between users and systems professionals lead to mistakes that, sometimes, are discovered too late.

These problems have led researchers to seek ways to improve the development process. The focus of this effort has been on techniques to reduce development time, facilitate better information transfer, encourage user involvement, and improve overall systems quality. Several widely used techniques for improving systems development are reviewed in the following section.

## Prototyping

**Prototyping** is a technique for providing users a preliminary working version of the system. The prototype is built quickly and inexpensively, with the intention that it will be modified. The objective of this technique is for the prototype to represent "an unambiguous functional specification,

FIGURE 14-2 PROTOTYPING TECHNIQUES




serve as a vehicle for organizing and learning, and evolve ultimately into a fully implemented" system.[1] As the users work with the prototype and make suggestions for changes, both they and the systems professional develop a better understanding of the true requirements of the system.

Reducing its features to the essential elements keeps the costs of a prototype model low. For example, the prototype system will not contain the complex code necessary to perform transaction validation, exception-handling capabilities, and internal controls. Typically, prototypes are limited to user input screens, output reports, and some principal functions.

When incorporated in the front-end stages of the SDLC, prototyping is an effective tool for establishing user requirements. Once these are obtained, the prototype is discarded. This throw-away prototyping is used for developing structured applications, such as accounting systems.[2] An alternative technique continues the prototyping process until the system is completed. This approach is used for developing decision support systems and expert systems. Figure 14-2 illustrates the prototyping model.

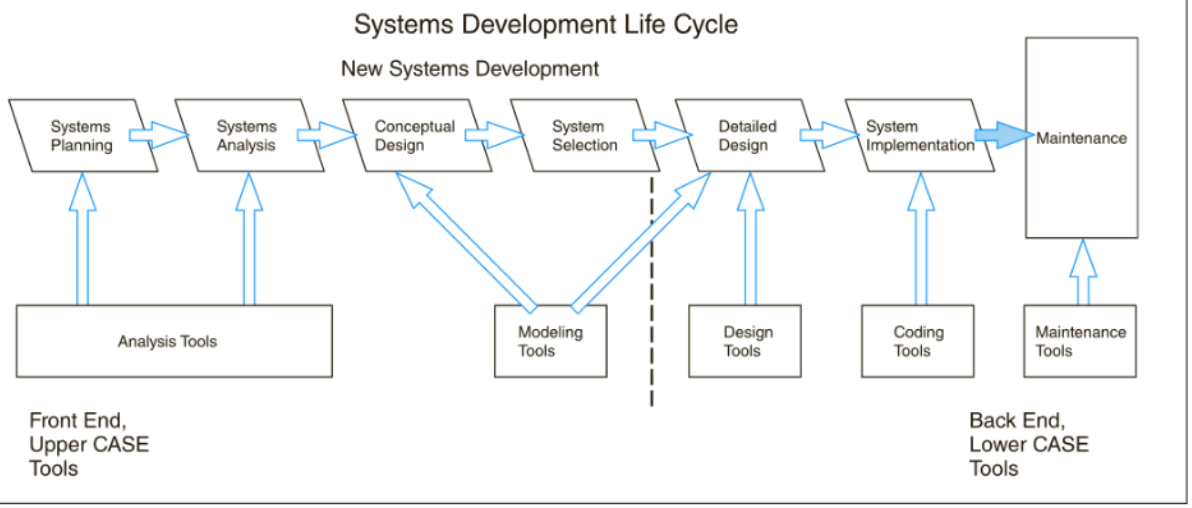
## The CASE Approach

**Computer-aided software engineering (CASE)** technology involves the use of computer systems to build computer systems. CASE tools are commercial software products consisting of highly integrated applications that support a wide range of SDLC activities. This methodology was developed to increase the productivity of systems professionals, improve systems design quality, and expedite the SDLC.

Most CASE products comprise both upper and lower tools or applications. Upper CASE tools support the conceptual activities of analysis and design. Lower CASE tools support the physical activities associated with application programming and system maintenance. CASE tools are used to define user requirements, create physical databases from conceptual entity relationship (ER) diagrams, produce system design specifications, automatically generate computer program code, and facilitate the maintenance of programs that both CASE and non-CASE techniques create. Figure 14-3 illustrates the CASE spectrum of tools as they apply to various stages in the SDLC. The appendix to this chapter presents more detailed discussion of CASE features.

1. J. C. Emery, *Management Information Systems: The Critical Strategic Resource* (New York: Oxford University Press, 1987), 325.
2. V. Zwass, *Management Information Systems* (Dubuque, Iowa: Wm. C. Brown, 1992), 740.

FIGURE 14-3 CASE SPECTRUM OF SUPPORT TOOLS FOR THE SDLC






## PERT Chart

The project evaluation and review technique (PERT) is a tool for showing the relationship among key activities that constitute the construct and delivery process. Figure 14-4 presents a **PERT chart** for a hypothetical project. The principal features of this diagram are:

1. *Activities*—the tasks to be completed in the project. These are labeled (and lettered A through L) on the lines, along with the time estimate for their completion. For example, the process design activity (C) is estimated to take 4 weeks.
2. *Events*—mark the completion of one activity and the beginning of the next. The events in this diagram are numbered 1 through 9.
3. *Paths*—routes through the diagram that connect the events from the first to the last.
4. *Critical path*—the path with the greatest overall time. The critical path in this project is C-F-G-J-L, with a total time of 20 (4+5+3+4+4) weeks. Any time delays in the activities along this path will extend the overall project time, which is why this path is critical.

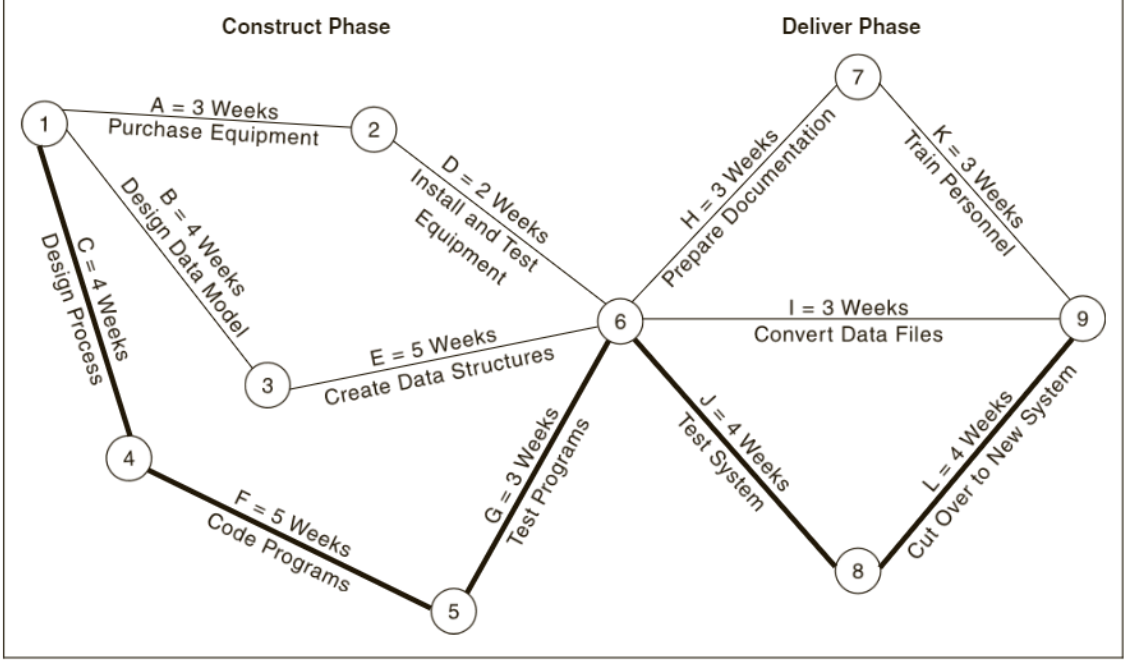
## Gantt Chart

The **Gantt chart** is a horizontal bar chart that presents time on a horizontal plane and activities on a vertical plane. Figure 14-5 illustrates a Gantt chart for the same project that the PERT chart represents in Figure 14-4. A bar marking its starting and ending dates represents the time associated with each activity. The Gantt chart is popular because it can show the current status of the project at a glance. By comparing projected time and work completed to date, we can see which projects are on, ahead of, or behind schedule.

# CONSTRUCT THE SYSTEM

The main goal of the **construct** phase is to design and build working software that is ready to be tested and delivered to its user community. This phase involves modeling the system, programming the applications, and application testing. The design and programming of modern systems follow one of two basic approaches: the structured approach and the object-oriented approach. We begin this section with a review of these competing methodologies. We then examine construct issues related to system design, programming, and testing.

FIGURE 14-4 PERT CHART FOR IN-HOUSE DEVELOPMENT PROJECT






## THE STRUCTURED DESIGN APPROACH

The **structured design** approach is a disciplined way of designing systems from the top down. It consists of starting with the big picture of the proposed system that is gradually decomposed into more and more detail until it is fully understood. Under this approach, the business process under design is usually documented by data flow and structure diagrams. Figure 14-6 shows the use of these techniques to depict the top-down decomposition of a hypothetical business process.

We can see from these diagrams how the systems designer follows a top-down approach. The designer starts with an abstract description of the system and, through successive steps, redefines this view to produce a more detailed description. In our example, Process 2.0 in the context diagram is decomposed into an intermediate-level data flow diagram (DFD). Process 2.3 in the intermediate DFD is further decomposed into an elementary DFD. This decomposition could involve several levels to obtain sufficient details. Let's assume that three levels are sufficient in this case. The final step transforms Process 2.3.3 into a structure diagram that defines the program modules that will constitute the process.

## THE OBJECT-ORIENTED DESIGN APPROACH

The **object-oriented design** approach is to build information systems from reusable standard components or objects. This approach may be equated to the process of building an automobile. Car manufacturers do not create each new model from scratch. New models are actually built from standard components that also go into other models. For example, each model of car that a particular manufacturer produces may use the same type of engine, gearbox, alternator, rear axle, radio, and so on. Some of the car's components will be industry-standard products that other manufacturers use. Such things as wheels, tires, spark plugs, and headlights fall into this category. In fact, it may be that the only component actually created from scratch for a new car model is the body.

The automobile industry operates in this fashion to stay competitive. By using standard components, car manufacturers minimize production and maintenance costs. At the same time, they can remain responsive to consumer demands for new products and preserve manufacturing flexibility by mixing and matching components according to the customer's specification.

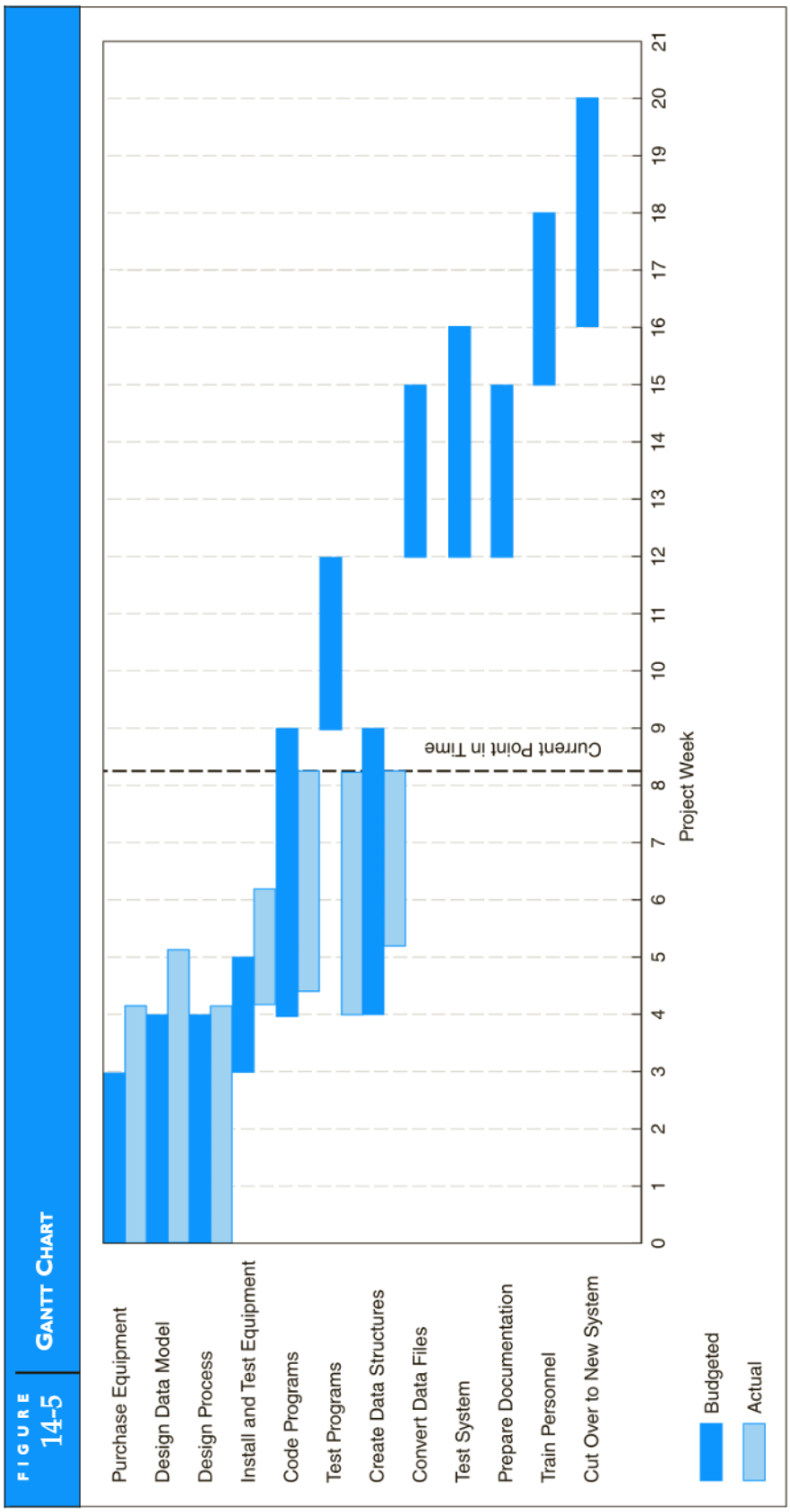
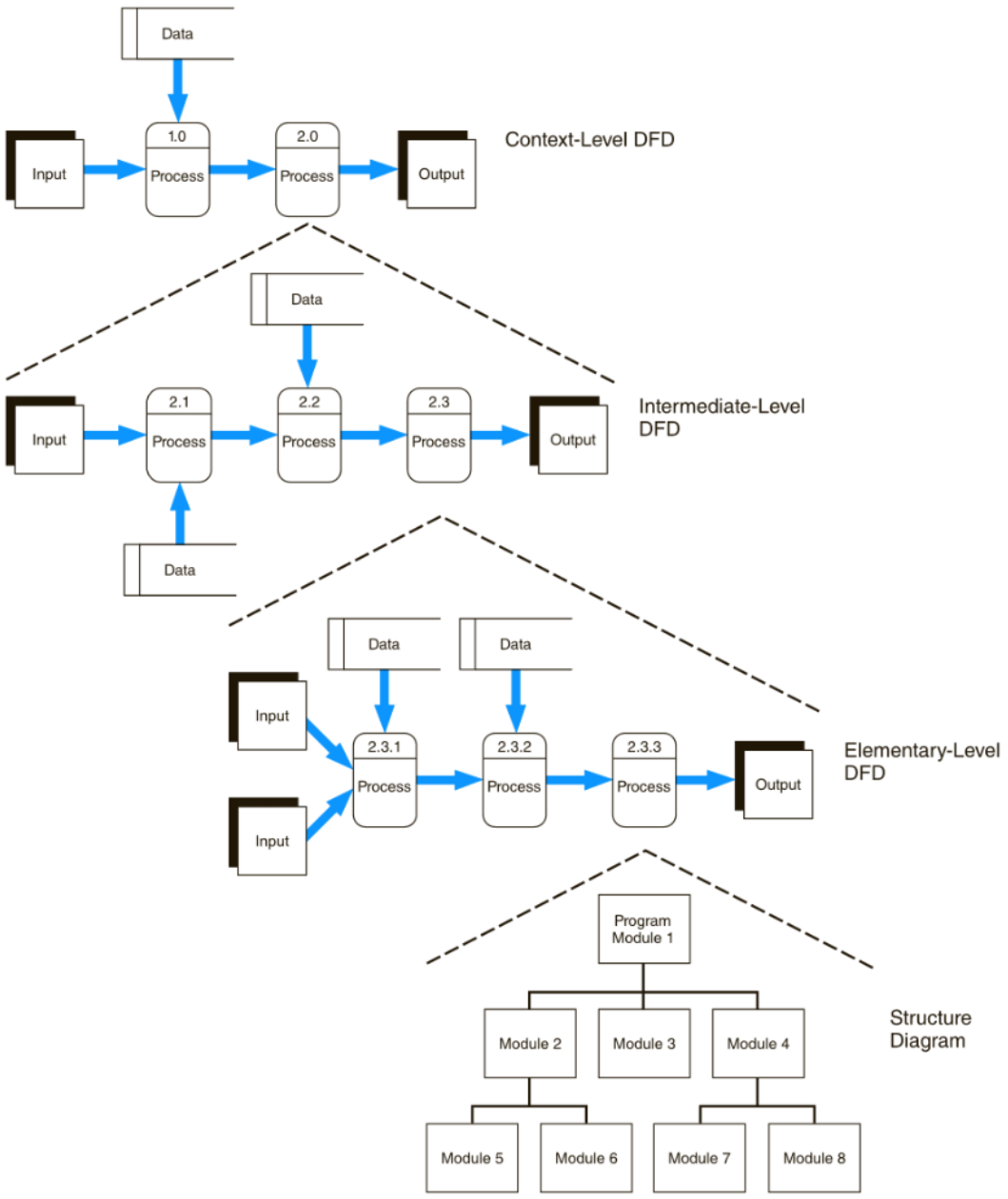



FIGURE 14-5 GANTT CHART

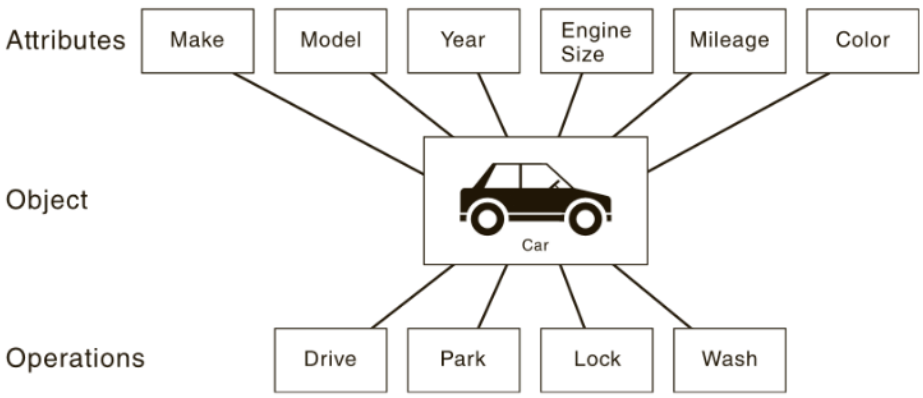
FIGURE 14-6 TOP-DOWN DECOMPOSITION OF THE STRUCTURED DESIGN APPROACH






The concept of reusability is central to the object-oriented design approach to systems design. Once created, standard modules can be used in other systems with similar needs. Ideally, the systems professionals of the organization will create a library (inventory) of modules that other systems designers within the firm can use. The benefits of this approach are similar to those stated for the automobile example. They include reduced time and cost for development, maintenance, and testing, and improved user support and flexibility in the development process.

FIGURE 14-7 CHARACTERISTICS OF OBJECTS






## Elements of the Object-Oriented Design Approach

A distinctive characteristic of the object-oriented design approach is that both data and programming logic, such as integrity tests, accounting rules, and updating procedures, are encapsulated in modules to represent objects. The following discussion deals with the principal elements of the object-oriented approach.

**OBJECTS. Objects** are equivalent to nouns in the English language. For example, vendors, customers, inventory, and accounts are all objects. These objects possess two characteristics: attributes and methods.

**Attributes** are the data that describe the objects. **Methods** are the actions that are performed on or by objects that may change their attributes. Figure 14-7 illustrates these characteristics with a nonfinancial example. The object in this example is an automobile whose attributes are its make, model, year, engine size, mileage, and color. Methods that may be performed on this object include drive, park, lock, and wash. Note that if we perform a drive method on the object, the mileage attribute will be changed.

Figure 14-8 illustrates these points with an inventory accounting example. In this example, the object is inventory, and its attributes are part number, description, quantity on hand, reorder point, order quantity, and supplier number. The methods that may be performed on inventory are reduce inventory

FIGURE 14-8 CHARACTERISTICS OF AN INVENTORY OBJECT

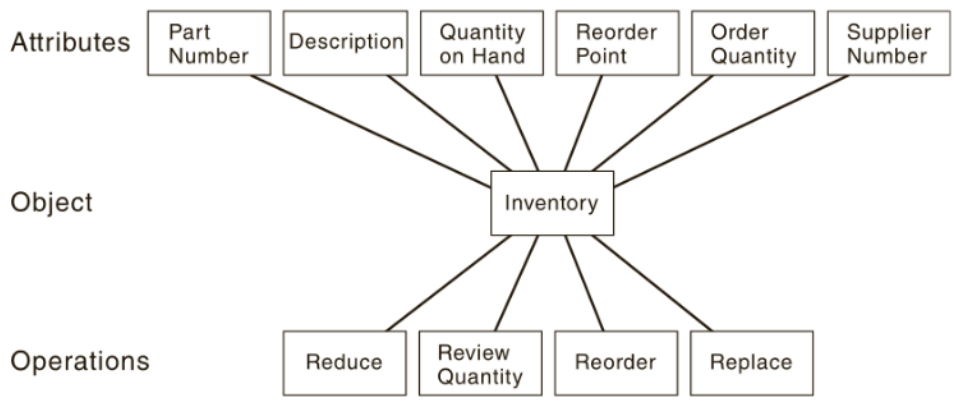
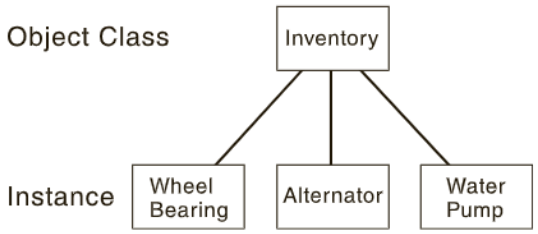


**FIGURE 14-9 RELATIONSHIP BETWEEN OBJECT CLASS AND INSTANCE**






(from product sales), review available quantity on hand, reorder inventory (when quantity on hand is less than the reorder point), and replace inventory (from inventory receipts). Again, note that performing any of the methods will change the attribute quantity on hand.

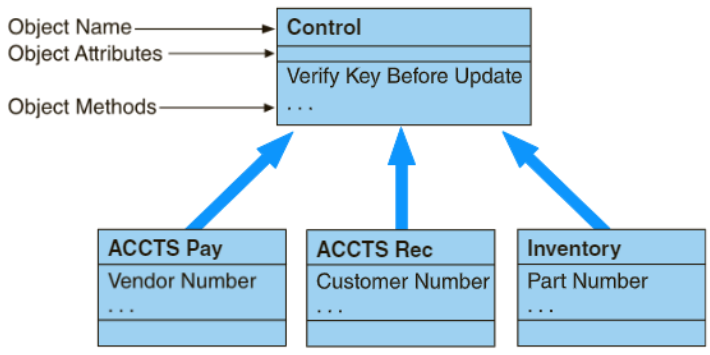
**CLASSES AND INSTANCES.** An **object class** is a logical grouping of individual objects that share the same attributes and methods. An **instance** is a single occurrence of an object within a class. For example, Figure 14-9 shows the inventory class consisting of several instances or specific inventory types.

**INHERITANCE. Inheritance** means that each object instance inherits the attributes and methods of the class to which it belongs. For example, all instances within the inventory class hierarchy share the attributes of part number, description, and quantity on hand. These attributes would be defined once and only once for the inventory object. Thus, the object instances of wheel bearing, water pump, and alternator will inherit these attributes. Likewise, these instances will inherit the methods (reduce, review, reorder, and replace) defined for the class.

Object classes can also inherit from other object classes. For example, Figure 14-10 shows an object hierarchy made up of an object class called control and three subclasses called accounts payable, accounts receivable, and inventory. This diagramming technique is an example of unified modeling language (UML). The object is represented as a rectangle with three levels: name, attributes, and methods.

The three object subclasses have certain control methods in common. For example, no account should be updated without first verifying the values Vendor Number, Customer Number, or Part

**FIGURE 14-10 OBJECTS, CLASSES, AND INHERITANCE**



Number. This method (and others) may be specified for the control object (once and only once), and then all the subclass objects to which this method applies inherit it.

Because object-oriented designs support the objective of reusability, portions of systems, or entire systems, can be created from modules that already exist. For example, any future system that requires the attributes and methods that the existing control module specifies can inherit them by being designated a subclass object.

Finally, the object-oriented approach offers the potential of increased security over the structured model. Each object's collection of methods, which creates an impenetrable **wall of code** around the data, determines its functionality (behavior). This means that the internal data of the object can be manipulated only by its methods. Direct access to the object's internal structure is not permitted.

## SYSTEM DESIGN

The purpose of the **design phase** is to produce a detailed description of the proposed system that both satisfies the system requirements identified during systems analysis and is in accordance with the conceptual design. In this phase, all system components—user views, database tables, processes, and controls—are meticulously specified. At the end of this phase, these components are presented formally in a detailed design report. This report constitutes a set of blueprints that specify input screen formats, output report layouts, database structures, and process logic. These completed plans then proceed to the final phase in the SDLC—system implementation—in which the system is physically constructed.

### The Design Sequence

The **systems design** phase of the SDLC follows a logical sequence of events: create a data model of the business process, define conceptual user views, design the normalized database tables, design the physical user views (output and input views), develop the process modules, specify the system controls, and perform a system walk-through. In this section, each of the design steps is examined in detail.

### An Iterative Approach

Typically, the design sequence listed in the previous section is not a purely linear process. Inevitably, system requirements change during the detailed design phase, causing the designer to revisit previous steps. For example, a last-minute change in the process design may influence data collection requirements that, in turn, changes the user view and requires alterations to the database tables.

To deal with this material as concisely and clearly as possible, the detailed design phase is presented here as a neat linear process. However, the reader should recognize its circular nature. This characteristic has control implications for both accountants and management. For example, a control issue that was previously resolved may need to be revisited as a result of modifications to the design.

## DATA MODELING, CONCEPTUAL VIEWS, AND NORMALIZED TABLES

**Data modeling** is the task of formalizing the data requirements of the business process as a conceptual model. The primary documentation instrument used for data modeling is the entity relationship (ER) diagram. This technique is used to depict the entities or data objects in the system. Once the entities have been represented in the data model, the data attributes that define each entity can then be described. They should be determined by careful analysis of user needs and may include both financial and nonfinancial data. These attributes represent the **conceptual user views** that must be supported by normalized database tables. To the extent that the data requirements of all users have been properly specified in the data model, the resulting databases will support multiple user views. We described data modeling, defining user views, and designing normalized base tables in Chapter 9 and its appendix.

## DESIGN PHYSICAL USER VIEWS

The physical views are the media for conveying and presenting data. These include output reports, documents, and input screens. The remainder of this section deals with a number of issues related to the design of physical user views. The discussion examines output and input views separately.

## Design Output Views

Output is the information the system produces to support user tasks and decisions. Table 14-1 presents examples of output that several AIS subsystems produce. At the transaction processing level, output tends to be extremely detailed. Revenue and expenditure cycle systems produce control reports for lower-level management and operational documents to support daily activities. Conversion cycle systems produce reports for scheduling production, managing inventory, and cost management. These systems also produce documents for controlling the manufacturing process.

The general ledger/financial reporting system (GL/FRS) and the management reporting system (MRS) produce output that is more summarized. The intended users of these systems are management, stockholders, and other interested parties outside the firm. The GL/FRS is a nondiscretionary reporting system that produces formal reports required by law. These include financial

**TABLE 14-1 EXAMPLES OF SYSTEM OUTPUTS**

| System | Output |
|---|---|
| Expenditure cycle | Purchase orders |
| | Cash disbursement voucher |
| | Payment check |
| | Purchases summary report |
| | Cash disbursements summary |
| Revenue cycle | Sales invoice |
| | Remittance advice |
| | Bill of lading |
| | Packing slip |
| | Customer statements |
| | Deposit slips |
| | Cash receipts prelist |
| | Sales summary |
| | Cash receipts summary |
| Conversion cycle | Purchase requisitions |
| | Work orders |
| | Move tickets |
| | Materials requisitions |
| | Production schedules |
| | Job tickets |
| | Employee time cards |
| | Work-in-process status reports |
| | Summary of changes to finish goods |
| General ledger and financial reporting system | Financial statements |
| | Comparative financial statements |
| | Tax returns |
| | Reports to regulatory agencies |
| Management reporting system | Various status and analysis reports such as: |
| | Inventory turnover reports |
| | Inventory status reports |
| | Vendor analysis reports |
| | Budget and performance reports |

statements, tax returns, and other reports that regulatory agencies demand. The output requirements of the GL/FRS tend to be predictable and stable over time and between organizations.

The management reporting system (MRS) serves the needs of internal management users. MRS applications may be stand-alone systems, or they may be integrated in the revenue, conversion, and expenditure cycles to produce output that contains both financial and nonfinancial information. The MRS produces problem-specific reports that vary considerably between business entities.

**OUTPUT ATTRIBUTES.** Regardless of their physical form, whether operational documents, financial statements, or discretionary reports, output views should possess the following attributes: relevance, summarization, exceptions orientation, timeliness, accuracy, completeness, and conciseness.

**RELEVANCE.** Each element of information output must support the user' decision or task. Irrelevant facts waste resources and detract attention from the information content of the output. Output documents that contain unnecessary facts tend to be cluttered, take time to process, cause bottlenecks, and promote errors.

**SUMMARIZATION.** Reports should be summarized according to the level of the user in the organization. The degree of summarization increases as information flows upward from lower-level managers to top management. We see this characteristic clearly in the responsibility reports represented in Figure 14-11.

**EXCEPTION ORIENTATION. Operations control reports** should identify activities that are about to go out of control and ignore those that are functioning within normal limits. This allows managers to focus their attention on areas of greatest need. An example of this is illustrated with the inventory reorder report presented in Figure 14-12. Only the items that need to be ordered are listed on the report.

**TIMELINESS.** Timely information that is reasonably accurate and complete is more valuable than perfect information that comes too late to be useful. Therefore, the system must provide the user with information that is timely enough to support the desired action.

**FIGURE 14-11 RESPONSIBILITY REPORTS SHOWING CONSOLIDATION OF INFORMATION**

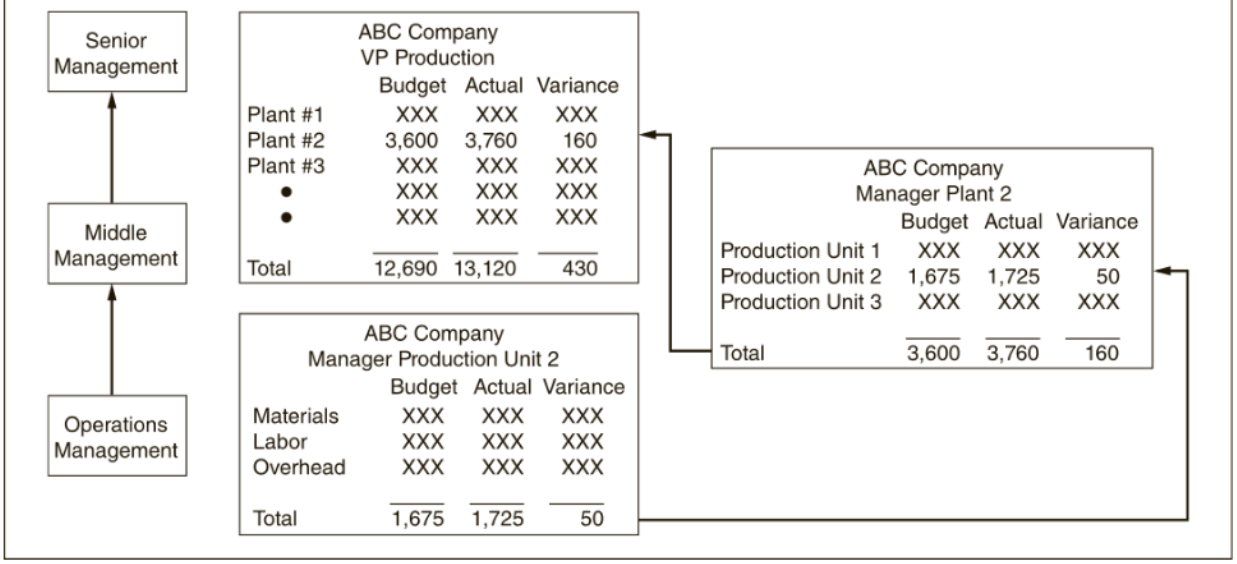


**FIGURE 14-12 INVENTORY REORDER REPORT**

# 12975

Sports Car Factory
Inventory Reorder Report

Date: 11/22/15

The following inventory items have fallen below normal levels:

| Part | Description | Primary Vendor | Order Quantity | Quantity On Hand | Average Daily Usage |
|---|---|---|---|---|---|
| 47782 | Exhaust Header | 2378 | 10 | 2 | 1 |
| 6671 | Wheel Bearing | 2378 | 500 | 25 | 20 |
| 9981 | Ball Joint | 2401 | 200 | 10 | 20 |



**ACCURACY.** Information output must be free of material errors. A material error is one that causes the user to take an incorrect action or to fail to take the correct action. Operational documents and low-level control reports usually require a high degree of accuracy. However, for certain planning reports and reports that support rapid decision making, the system designer may need to sacrifice accuracy to produce information that is timely. Managers cannot always wait until they have all the facts before they must act. The designer must seek a balance between the competing needs for accuracy and timeliness when designing output reports.

**COMPLETENESS.** Information must be as complete as possible. Ideally, no piece of information essential to the task or decision should be missing from the output. As with the accuracy attribute, the designer must sometimes sacrifice completeness in favor of timely information.

**CONCISENESS.** Information output should be presented as concisely as possible within the report or document. Output should use coding schemes to represent complex data classifications. Also, information should be clearly presented with titles for all values. Reports should be visually pleasing and logically organized.

**OUTPUT REPORTING TECHNIQUES.** While recognizing that differences in cognitive styles exist among managers, systems designers must determine the output type and format most useful to the user. Some managers prefer output that presents information in tables and matrices. Others prefer information that is visually oriented in the form of graphs and charts. The issue of whether the output should be hard copy (paper) or electronic must also be addressed.

Despite predictions for two decades or more, we have not yet achieved a paperless society. On the contrary, trees continue to be harvested, and paper mills continue to be productive. In some firms, top management receives hundreds of pages of paper output each day. Paper documents also continue to flow at lower organizational levels.

On the other hand, many firms are moving to paperless audit trails and support daily tasks with electronic documents. Insurance companies, law firms, and mortgage companies make extensive use of electronic documents. The use of electronic output greatly reduces or eliminates the

problems associated with paper documents (purchasing, handling, storage, and disposal). However, the use of electronic output has obvious implications for accounting and auditing.

The query and report-generating features of modern database management systems permit the manager to quickly create standard and customized output reports. Custom reports can present information in different formats, including text, matrices, tables, and graphs. Section 1 in the chapter appendix examines these formats and provides some examples.

## Design Input Views

Data input views are used to capture the relevant facts about the resources, events, and agents involved in business process transactions. In this section, we divide input into two classes: hard-copy input and electronic input.

**DESIGN HARD-COPY INPUT.** Businesses today still make extensive use of paper input documents. In designing **hard-copy** documents, the system designer must keep in mind several aspects of the physical business process.

**HANDLING.** How will the document be handled? Will it be on the shop floor around grease and oil? How many hands must it go through? Is it likely to get folded, creased, or torn? Input forms are part of the audit trail and must be preserved in legible form. If they are to be subjected to physical abuse, they must be made of high-quality paper.

**STORAGE.** How long will the form be stored? What is the storage environment? Length of storage time and environmental conditions will influence the appearance of the form. Data entered onto poor-quality paper may fade under extreme conditions. Again, this may have audit trail implications. A related consideration is the need to protect the form against erasures.

**NUMBER OF COPIES.** Source documents are often created in multiple copies to trigger multiple activities simultaneously and provide a basis for reconciliation. For example, the system may require that individual copies of sales orders go to the warehouse, the shipping department, billing, and accounts receivable. Manifold forms are often used in such cases. A manifold form produces several carbon copies from a single writing. The copies are normally color-coded to facilitate distribution to the correct users.

**FORM SIZE.** The average number of facts captured for each transaction affects the size of the form. For example, if the average number of items received from the supplier for each purchase is 20, the receiving report should be long enough to record them all. Otherwise, additional copies will be needed, which will add to the clerical work, clog the system, and promote processing errors.

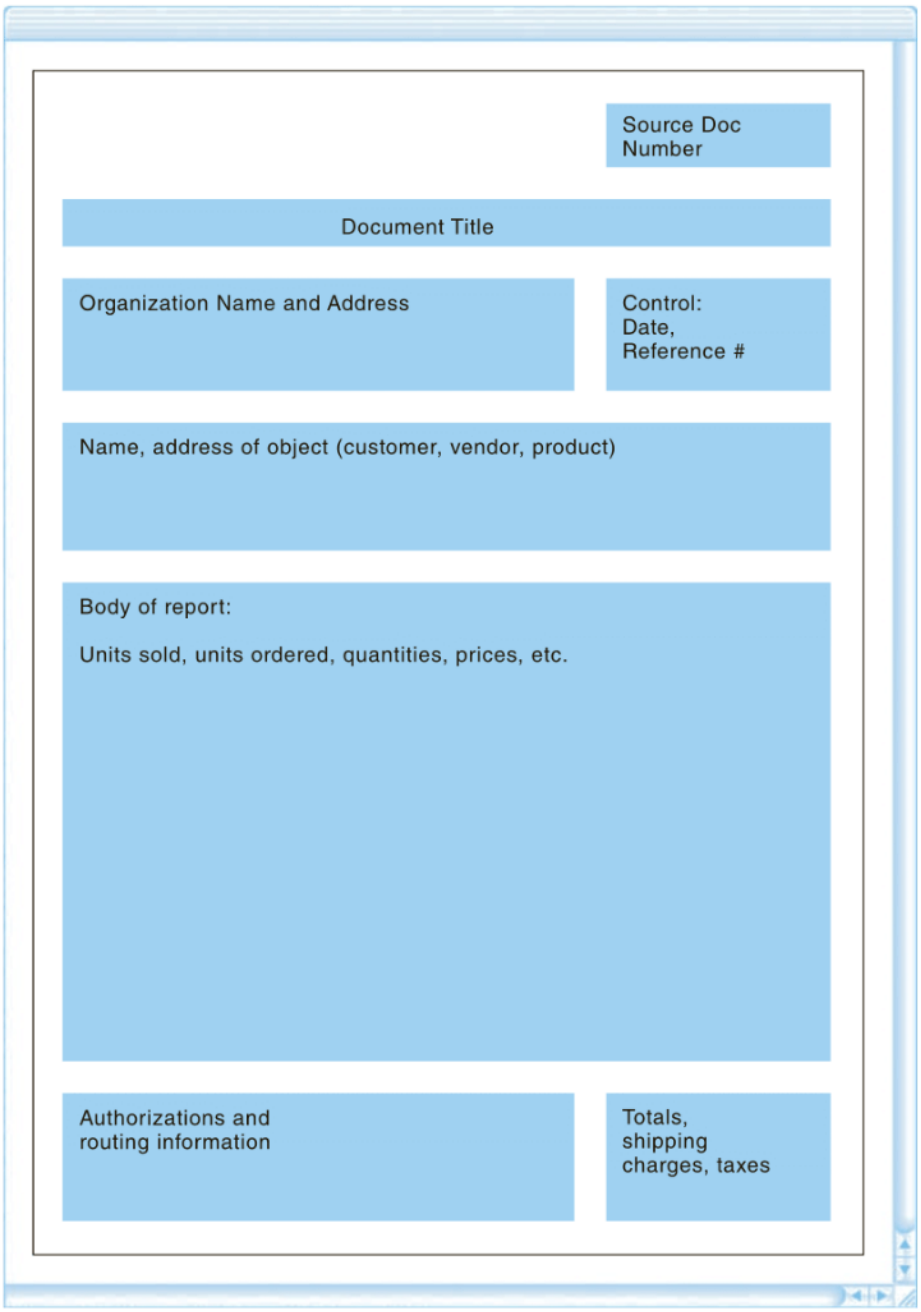
Standard forms sizes are full-size, 8½ by 11 inches; and half-size, 8½ by 5½ inches. Card form standards are 8 by 10 inches and 8 by 5 inches. The use of nonstandard forms can cause handling and storage problems and should be avoided.

**FORM DESIGN.** Clerical errors and omissions can cause serious processing problems. Input forms must be designed to be easy to use and collect the data as efficiently and effectively as possible. This requires that forms be logically organized and visually comfortable to the user. Two techniques used in well-designed forms are zones and embedded instructions.

**ZONES. Zones** are areas on the form that contain related data. Figure 14-13 provides an example of a form divided into zones. Each zone should be constructed of lines, captions, or boxes that guide the user's eye to avoid errors and omissions.

**EMBEDDED INSTRUCTIONS. Embedded instructions** are contained within the body of the form itself rather than on a separate sheet. It is important to place instructions directly in the zone to which they pertain. If an instruction pertains to the entire form, it should be placed at the top of the form. Instructions should be brief and unambiguous. As an instructive technique, active voice is

FIGURE 14-13 ZONES OF A FORM



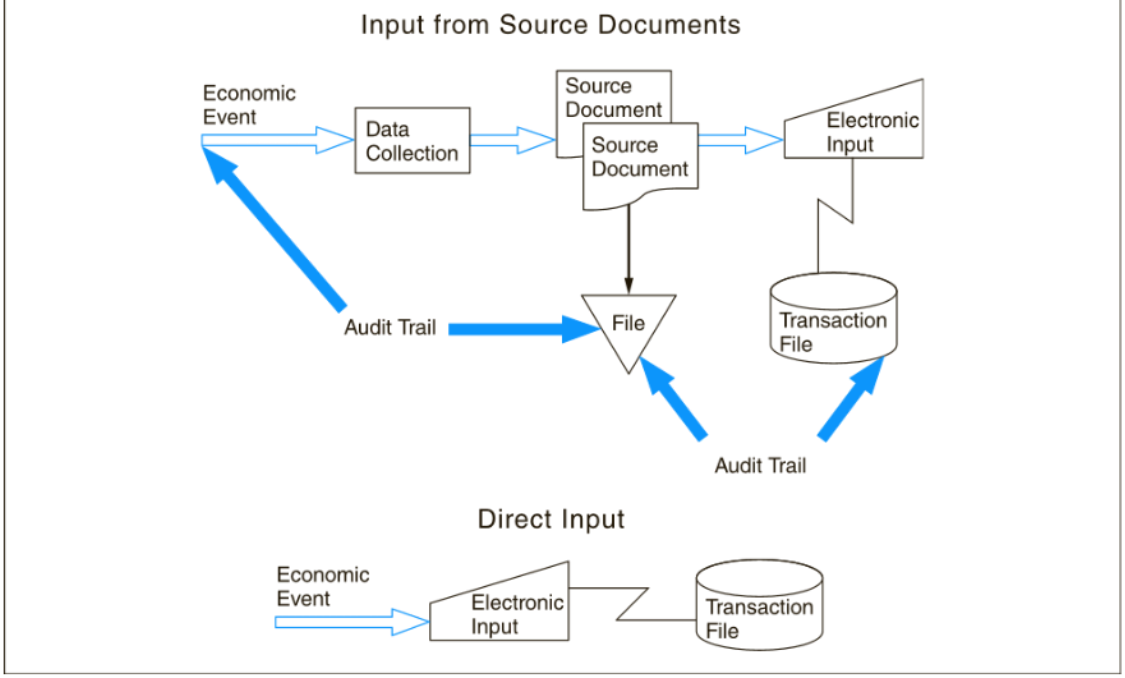



stronger, more efficient (needing fewer words), and less ambiguous than passive voice. For example, the first instruction that follows is written in passive voice. The second is in active voice.

1. This form should be completed in ink.
2. Complete this form in ink.

Notice the difference: the second sentence is stronger, shorter, and clearer than the first; it is an instruction rather than a suggestion.

**FIGURE 14-14** INPUT FROM SOURCE DOCUMENT AND DIRECT INPUT




**DESIGN ELECTRONIC INPUT. Electronic input techniques** fall into two basic types: input from source documents and direct input. Figure 14-14 illustrates the difference in these techniques. Input from source documents involves the collection of data on paper forms that are then transcribed to electronic forms in a separate operation. Direct input procedures capture data directly in electronic form, via terminals at the source of the transaction.

**INPUT FROM SOURCE DOCUMENTS.** Firms use paper source documents for a number of reasons. Some firms prefer to maintain a paper audit trail that goes back to the source of an economic event. Some companies capture data onto paper documents because direct input procedures may be inconvenient or impossible. Other firms achieve economies of scale by centralizing electronic data collection from paper documents.

An important aspect of this approach is to design input screens that visually reflect the source document. The captions and data fields should be arranged on the electronic form exactly as they are on the source document. This minimizes eye movement between the source and the screen and maximizes throughput of work.

**DIRECT INPUT.** Direct data input requires that data collection technology be distributed to the source of the transaction. A very common example of this is the point-of-sale terminal in a department store.

An advantage of direct input is the reduction of input errors that plague downstream processing. By collecting data once, at the source, clerical errors are reduced because the subsequent transcription step associated with paper documents is eliminated. The more times a transaction is manually transcribed, the greater the potential for error.

Direct data collection uses **intelligent forms** for online editing that help the user complete the form and make calculations automatically. The input screen is attached to a computer that performs logical checks on the data being entered. This reduces input errors and improves the efficiency of the data collection procedures. During data entry, the intelligent form will detect transcription errors, such as illegal characters in a field, incorrect amounts, and invalid item numbers. A beep can be used to draw attention to an error, an illegal action, or a screen message. Thus, corrections to input can be made on the spot.

Given minimal input, an intelligent form can complete the input process automatically. For example, a sales clerk need enter in the terminal only the item numbers and quantities of products sold. The system will automatically provide the descriptions, prices, price extensions, taxes, and freight charges, and calculate the grand total. Many time-consuming and error-prone activities are eliminated through this technique. Modern relational database packages have a screen painting feature that allows the user to quickly and easily create intelligent input forms.

**DATA ENTRY DEVICES.** A number of data entry devices are used to support direct electronic input. These include point-of-sale terminals, magnetic ink character recognition devices, optical character recognition devices, automatic teller machines, and voice recognition devices.

## DESIGN THE SYSTEM PROCESS

Now that the database tables and user views for the system have been designed, we are ready to design the process component. This starts with the DFDs that were produced in the general design phase. Depending on the extent of the activities performed in the general design phase, the system may be specified at the context level or may be refined in lower-level DFDs. The first task is to decompose the existing DFDs to a degree of detail that will serve as the basis for creating structure diagrams. The structure diagrams will provide the blueprints for writing the actual program modules.

### Decompose High-Level DFDs

To demonstrate the decomposition process, we will use the intermediate DFD of the purchases and cash disbursements system illustrated in Figure 14-15. This DFD was decomposed from the

FIGURE 14-15 DFD FOR PURCHASES AND CASH DISBURSEMENTS SYSTEM

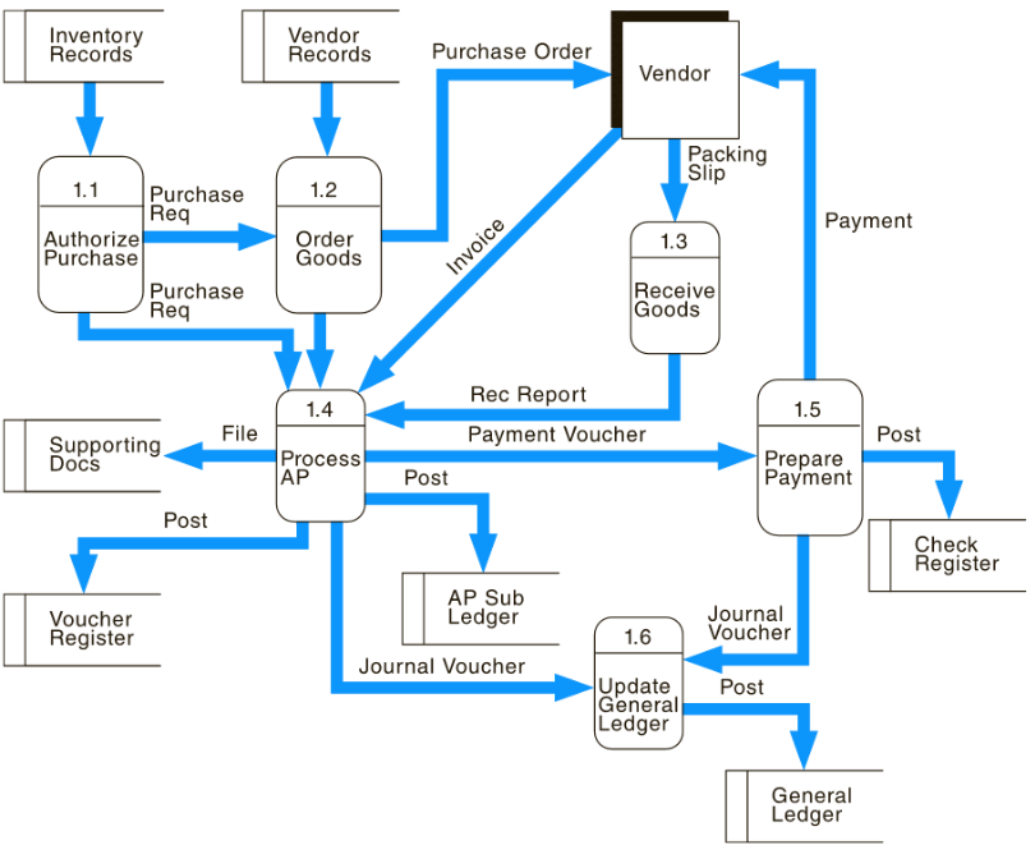
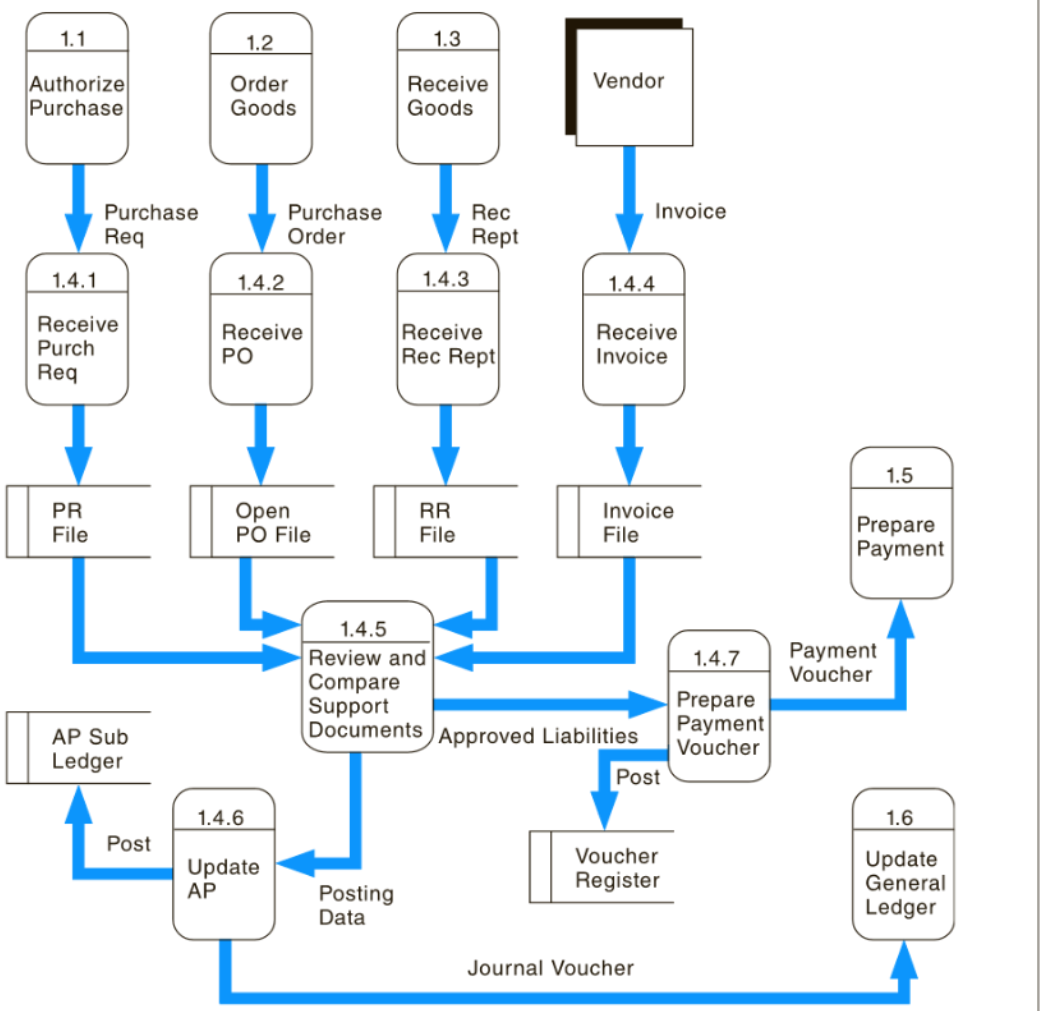


FIGURE 14-16 LOWER-LEVEL DFD FOR AP PROCESS 1.4






context-level DFD (Figure 13-6, Option A) originally prepared in the conceptual design phase. We will concentrate on the accounts payable process numbered 1.4 in the diagram. This process is not yet sufficiently detailed to produce program modules.

Figure 14-16 shows Process 1.4 decomposed into the next level of detail. Each of the resulting subprocesses is numbered with a third-level designator, such as 1.4.1, 1.4.2, 1.4.3, and so on. We will assume that this level of DFD provides sufficient detail to prepare a structure diagram of program modules. Many CASE tools will automatically convert DFDs to structure diagrams. However, to illustrate the concept, we will go through the process manually.

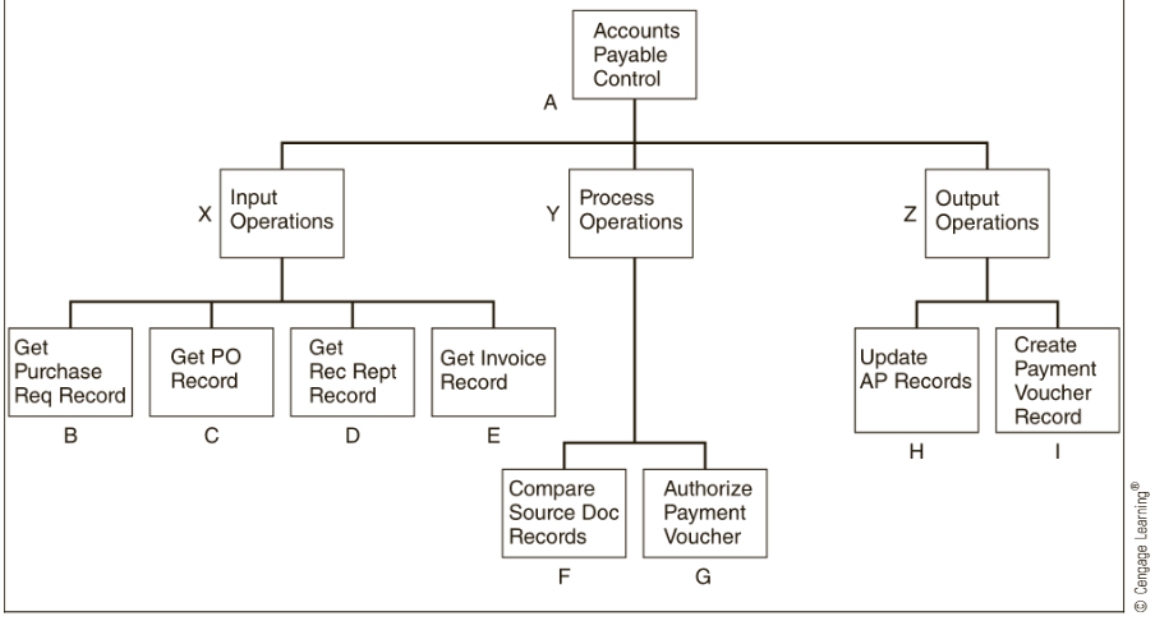
## Design Structure Diagrams

The creation of the **structure diagram** requires analysis of the DFD to divide its processes into input, process, and output functions. Figure 14-17 presents a structure diagram showing the program modules based on the DFD in Figure 14-16.

## The Modular Approach

The modular approach presented in Figure 14-17 involves arranging the system in a hierarchy of small, discrete modules, each of which performs a single task. Correctly designed modules possess two

**FIGURE 14-17 STRUCTURE DIAGRAM FOR AP PROCESS**





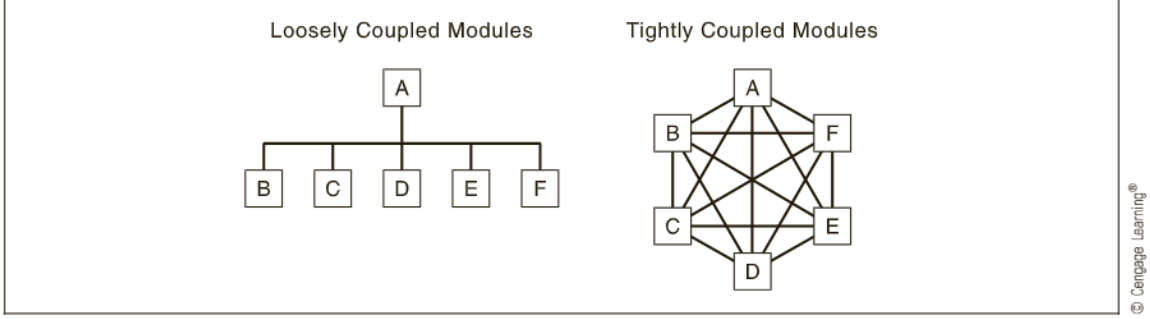

attributes: (1) they are loosely coupled, and (2) they have strong cohesion. **Coupling** measures the degree of interaction between modules. Interaction is the exchange of data between modules. A loosely coupled module is independent of the others. Modules with a great deal of interaction are tightly coupled. Figure 14-18 shows the relationship between modules in loosely coupled and tightly coupled designs.

In the loosely coupled system, the process starts with Module A. This module controls all the data flowing through the system. The other modules interact only with this module to send and receive data. Module A then redirects the data to other modules.

**Cohesion** refers to the number of tasks a module performs. Strong cohesion means that each module performs a single, well-defined task. Returning to Figure 14-17, Module D gets receiving reports and only that. It does not compare reports to open purchase orders, nor does it update accounts payable. Separate modules perform these tasks.

Modules that are loosely coupled and strongly cohesive are much easier to understand and easier to maintain. Maintenance is an error-prone process, and it is not uncommon for errors to be accidentally inserted into a module during maintenance. Thus, changing a single module within a

**FIGURE 14-18 LOOSELY AND TIGHTLY COUPLED MODULES**



tightly coupled structure can have an impact on the other modules with which it interacts. Look again at the tightly coupled structure in Figure 14-18. These interactions complicate maintenance by extending the process to the other modules. Similarly, modules with weak cohesion—those that perform several tasks—are more complex and difficult to maintain.

### Pseudocode the System Modules

Each module in Figure 14-17 represents a separate computer program. The higher-level programs will communicate with lower-level programs through call commands. System modules are coded in the implementation phase. When we get to that phase, we will examine programming language options. At this point, the designer must specify the functional characteristics of the modules through other techniques.

Next, we illustrate how **pseudocode** may be used to describe the function of Module F in Figure 14-17. This module authorizes payment of accounts payable by validating the supporting documents.

```
COMPARE-DOCS (Module F)
READ PR-RECORD FROM PR-FILE
READ PO-RECORD FROM PO-FILE
READ RR-RECORD FROM RR-FILE
READ INVOICE-RECORD FROM INVOICE-FILE
IF ITEM-NUM, QUANNTY-RECEIVED, TOTAL AMOUNT IS EQUAL FOR ALL
RECORDS
THEN PLACE "Y" IN AUTHORIZED FIELD OF PO-RECORD
ELSE READ ANOTHER RECORD
```

The use of pseudocode for specifying module functions has two advantages. First, the designer can express the detailed logic of the module, regardless of the programming language to be used. Second, although the end user may lack programming skills, he or she can be actively involved in this technical but crucial step.

## DESIGN SYSTEM CONTROLS

The last step in the design phase is the design of system controls. This includes computer processing controls, database controls, and manual controls over input to and output from the system, as well as controls over the operational environment (e.g., distributed data processing controls). In practice, many controls that are specific to a type of technology or technique will, at this point, have already been designed, along with the modules to which they relate. This step in the design phase allows the design team to review, modify, and evaluate controls with a system-wide perspective that did not exist when each module was being designed independently. Because of the extensive nature of computer-based system controls, treatment of this aspect of the systems design is deferred to Chapters 15, 16, and 17, where they can be covered in depth.

## PERFORM A SYSTEM DESIGN WALK-THROUGH

After completing the detailed design, the development team usually performs a system design **walk-through** to ensure the design is free from conceptual errors that could become programmed into the final system. Many firms have formal, structured walk-throughs that a **quality assurance group** conducts. This is an independent group of programmers, analysts, users, and internal auditors. The job of this group is to simulate the operation of the system to uncover errors, omissions, and ambiguities in the design. Most system errors emanate from poor designs rather than programming mistakes. Detecting and correcting errors in the design thus reduce costly reprogramming later.

### Review System Documentation

The **detailed design report** documents and describes the system to this point. This report includes:

- Designs of all screen outputs, reports, and operational documents.
- ER diagrams describing the data relations in the system.

- Third normal form designs for database tables specifying all data elements.
- An updated **data dictionary** describing each data element in the database.
- Designs for all screen inputs and source documents for the system.
- Context diagrams for the overall system.
- Low-level data flow diagrams of specific system processes.
- Structure diagrams for the program modules in the system, including a pseudocode description of each module.

The quality control group scrutinizes these documents, and any errors are recorded in a walkthrough report. Depending on the extent of the system errors, the quality assurance group will make a recommendation. The system design will be accepted without modification, accepted subject to modification of minor errors, or rejected because of material errors.

At this point, a decision is made either to return the system for additional design or to proceed to the next phase—systems implementation. Assuming the design goes forward, the documents just mentioned constitute the blueprints that guide programmers and system designers in constructing the physical system.

## PROGRAM APPLICATION SOFTWARE

The next stage of the in-house development is to select a programming language from among the various languages available and suitable to the application. These include procedural languages such as COBOL, event-driven languages such as Visual Basic, or object-oriented programming (OOP) languages such as Java or C++. This section presents a brief overview of various programming approaches. Systems professionals will make their decision based on the in-house standards, architecture, and user needs.

### Procedural Languages

A **procedural language** requires the programmer to specify the precise order in which the program logic is executed. Procedural languages are often called **third-generation languages** (3GLs). Examples of 3GLs include COBOL, FORTRAN, C, and PL1. In business (particularly in accounting) applications, COBOL was the dominant language for years. COBOL has great capability for performing highly detailed operations on individual data records and handles large files efficiently. On the other hand, it is an extremely wordy language that makes programming a time-consuming task. COBOL has survived as a viable language because many of the legacy systems written in the 1970s and 1980s, which were coded in COBOL, are still in operation today. Major retrofits and routine maintenance to these systems need to be coded in COBOL. More than 12 billion lines of COBOL code are executed daily in the United States.

### Event-Driven Languages

**Event-driven languages** are not procedural. Under this model, the program's code is not executed in a predefined sequence. Instead, external actions or events that the user initiates dictate the control flow of the program. For example, when the user presses a key or clicks on a computer icon on the screen, the program automatically executes code associated with that event. This is a fundamental shift from the 3GL era. Now, instead of designing applications that execute sequentially from top to bottom in accordance with the way the programmer thinks they should function, the user is in control. Microsoft's Visual Basic is the most popular example of an event-driven language. The syntax of the language is simple yet powerful. Visual Basic is used to create real-time and batch applications that can manipulate flat files or relational databases. It has a screen-painting feature that greatly facilitates the creation of sophisticated graphical user interfaces.

### Object-Oriented Languages

Central to achieving the benefits of this approach is developing software in an **object-oriented programming (OOP) language**. The most popular true OOP languages are Java and Smalltalk.

However, the learning curve of **OOP** languages is steep. The time and cost of retooling for OOP are the greatest impediments to the transition process. Most firms are not prepared to discard millions of lines of traditional COBOL code and retrain their programming staffs to implement object-oriented systems. Therefore, a compromise, intended to ease this transition, has been the development of hybrid languages, such as Object COBOL, Object Pascal, and C++.

## Programming the System

Regardless of the programming language used, modern programs should follow a modular approach. This technique produces small programs that perform narrowly defined tasks. The following three benefits are associated with modular programming.

**PROGRAMMING EFFICIENCY.** Modules can be coded and tested independently, which vastly reduces programming time. A firm can assign several programmers to a single system. Working in parallel, the programmers each design a few modules. These are then assembled into the completed system.

**MAINTENANCE EFFICIENCY.** Small modules are easier to analyze and change, which reduces the start-up time during program maintenance. Extensive changes can be parceled out to several programmers simultaneously to shorten maintenance time.

**CONTROL.** By keeping modules small, they are less likely to contain material errors of fraudulent logic. Because each module is independent of the others, errors are contained within the module.

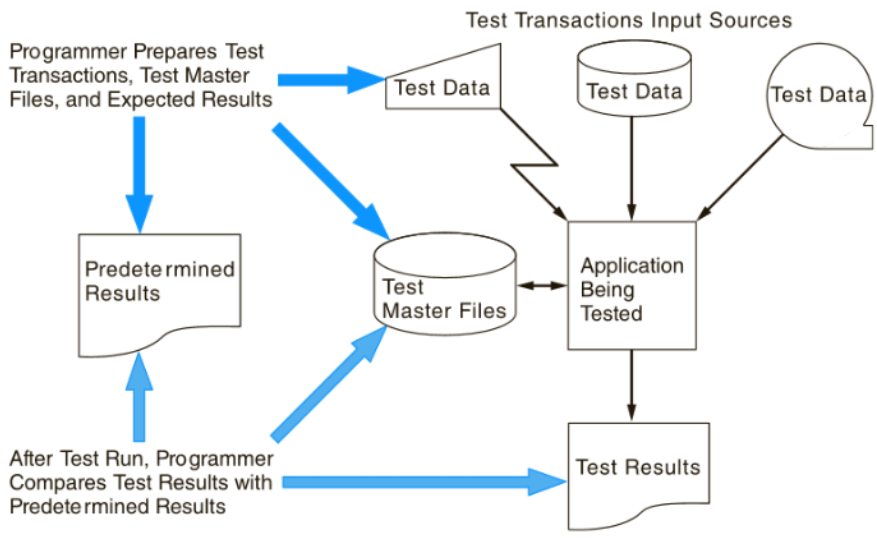
# SOFTWARE TESTING

Programs must be thoroughly tested before they are implemented. Program testing issues of direct concern to accountants are discussed in this section.

## Testing Individual Modules

Programmers should test completed modules independently before implementing them. This usually involves the creation of test data. Depending on the nature of the application, this could include test transaction files, test master files, or both. Figure 14-19 illustrates the test data approach. We examine this and several other testing techniques in detail in Chapter 17.

**FIGURE 14-19 THE TEST DATA TECHNIQUE**



Assume the module under test is the Update AP Records program (Module H) represented in Figure 14-17. The approach taken is to test the application thoroughly within its range of functions. To do this, the programmer must create some test accounts payable master file records and test transactions. The transactions should contain a range of data values adequate to test the logic of the application, including both good and bad data. For example, the programmer may create a transaction with an incorrect account number to see how the application handles such errors. The programmer will then compare the amounts posted to accounts payable records to see if they tally with precalculated results. Tests of all aspects of the logic are performed in this way, and the test results are used to identify and correct errors in the logic of the module.

## DELIVER THE SYSTEM

The system is now ready to be implemented. In this phase, database structures are populated with data, equipment is purchased and installed, employees are trained, and the system is documented. This phase concludes with the roll-out of the new system and the termination of the old system.

The implementation process engages the efforts of designers, programmers, database administrators, users, and accountants. All the steps in this stage warrant careful management. Nevertheless, not all steps are part of every system's implementation, and not all are of direct concern to accountants. For example, the implementation activities of ordering equipment from vendors, preparing the site, installing equipment, and training employees are not performed with each new system. Moreover, these are technical tasks that do not usually involve the accounting function. This section focuses on those activities that have the greatest direct implications for accountants and auditors.

## TESTING THE ENTIRE SYSTEM

When all modules have been coded and tested, they must be brought together and tested as a whole. User personnel should direct system-wide testing as a prelude to the formal system cutover. The procedure involves using the system to process hypothetical data. The outputs of the system are then reconciled with predetermined results, and the test is documented to provide evidence of the system's performance. Finally, when those conducting the tests are satisfied with the results, a formal acceptance document should be completed. This is an explicit acknowledgment by the user that the system in question meets stated requirements. The user acceptance document becomes important in reconciling differences and assigning responsibilities during the post-implementation review of the system.

### Saving Test Data

The preparation of test data is a tedious, time-consuming activity. The auditor should save these data during system reviews for future use. By preserving the test data, we create what is called a base case, which documents how the system performed at a point in time. At any future point, the base case data should generate the same results. System changes (maintenance) that have occurred since system implementation will explain the differences between base case results and current test results. Hence, the base case provides a reference point for analyzing the effects of system changes and eases the burden of creating test data.

## DOCUMENTING THE SYSTEM

The system's **documentation** describes how the system works. In this section, we consider the documentation requirements of four groups: systems designers and programmers, computer operators, end users, and accountants.

### Designer and Programmer Documentation

Systems designers and programmers need documentation to debug errors and perform maintenance on the system. This group is involved with the system on a highly technical level, which requires both general and detailed information. Some of this is provided through DFDs, ER diagrams, and structure diagrams. In addition, system flowcharts, program flowcharts, and listings of program code are important

forms of documentation. The system flowchart shows the relationship of input files, programs, and output files. However, it does not reveal the logic of individual programs that constitute the system. The program flowchart provides a detailed description of the sequential and logical operation of the program. A separate program flowchart represents each program in the system's flowchart. From these, the programmer can visually review and evaluate the program's logic. The program code should itself be documented with comments that describe each major program segment.

## Operator Documentation

Computer operators use documentation describing how to run the system, called a **run manual**. The typical contents of a run manual include:

- The name of the system, such as Purchases System.
- The run schedule (daily, weekly, time of day, etc.).
- Required hardware devices (tapes, disks, printers, or special hardware).
- File requirements specifying all the transaction (input) files, master files, and output files used in the system.
- Run-time instructions describing the error messages that may appear, actions to be taken, and the name and telephone number of the programmer on call, should the system fail.
- A list of users who receive the output from the run.

For security and control reasons, system flowcharts, logic flowcharts, and program code listings are not part of the operator documentation. Operators should not have access to the details of a system's internal logic. We will discuss this point more fully in Chapter 15.

## User Documentation

Users need documentation describing how to use the system. User tasks include such things as entering input for transactions, making inquiries of account balances, updating accounts, and generating output reports. The nature of user documentation will depend on the user's degree of sophistication with computers and technology. Thus, before designing user documentation, the systems professional must assess and classify the user's skill level. The following is one classification scheme:

- Novices
- Occasional users
- Frequent light users
- Frequent power users

Novices have little or no experience with computers and may be embarrassed to ask questions. Novices also know little about their assigned tasks. User training and documentation for novices must be extensive and detailed.

Occasional users once understood the system but have forgotten some essential commands and procedures. They require less training and documentation than novices.

Frequent light users are familiar with limited aspects of the system. Although they understand the basic functions, they tend not to explore beneath the surface and lack depth of knowledge. This group knows only what it needs to know and requires training and documentation for unfamiliar areas.

Frequent power users understand the existing system and will readily adapt to new systems. They are intolerant of detailed instructions that waste their time. They like to find shortcuts and use macro commands to improve performance. This group requires only abbreviated documentation.

With these classes in mind, user documentation often takes the form of a user handbook, as well as online documentation. The typical **user handbook** will contain the following items:

- An overview of the system and its major functions
- Instructions for getting started
- Descriptions of procedures with step-by-step visual references

- Examples of input screens and instructions for entering data
- A complete list of error message codes and descriptions
- A reference manual of commands to run the system
- A glossary of key terms
- Service and support information

**Online documentation** will guide the user interactively in the use of the system. Some commonly found online features include tutorials and help features.

**TUTORIALS.** Online tutorials can be used to train the novice or the occasional user. The success of this technique is based on the tutorial's degree of realism. Tutorials should not restrict the user from access to legitimate functions.

**HELP FEATURES.** Online help features range from simple to sophisticated. A simple help feature may be nothing more than an error message displayed on the screen. The user must walk through the screens in search of the solution to the problem. More sophisticated help is context related. When the user makes an error, the system will send the message, "Do you need help?" The help feature analyzes the context of what the user is doing at the time of the error and provides help with that specific function (or command).

### Accountant (Auditor) Documentation

With responsibility for the design of certain security procedures, accounting controls, and audit trails, accountants are stakeholders in all AIS applications. For these tasks, accountants may draw upon all of the documentation described previously. As internal and external auditors, accountants also require document flowcharts of manual procedures. We have encountered numerous examples of document flowcharts in the chapters dealing with the revenue, expenditure, and conversion cycles. Document flowcharts differ from DFDs in an important way. DFDs describe the overall logic of the system. Document flowcharts show explicitly the flow of information between departments, the departments in which tasks are actually performed, and the specific types and number of documents that carry information. A physical view such as this is needed to understand the segregation of duties, the adequacy of source documents, and the location of files that support the audit trail. Document flowcharts are not always included as part of the system's documentation. When not provided, auditors must create their own during the audit process.

## CONVERTING THE DATABASES

**Database conversion** is a critical step in the implementation phase. This is the transfer of data from its current form to the format or medium the new system requires. The degree of conversion depends on the technology leap from the old system to the new one. Some conversion activities are very labor-intensive, requiring data to be entered into new databases manually. For example, the move from a manual system to a computer system will require converting files from paper to magnetic disk or tape. In other situations, writing special conversion programs may accomplish data transfer. A case in point is changing the file structure of the databases from sequential direct access files. In any case, data conversion is risky and must be carefully controlled. The following precautions should be taken:

1. *Validation.* The old database must be validated before conversion. This requires analyzing each class of data to determine whether it should be reproduced in the new database.
2. *Reconciliation.* After the conversion action, the new database must be reconciled against the original. Sometimes this must be done manually, record by record and field by field. In many instances, writing a program that will compare the two sets of data can automate this process.
3. *Backup.* Copies of the original files must be kept as backup against discrepancies in the converted data. If the current files are already in magnetic form, a backup of these files can be

conveniently made and stored. However, paper documents can create storage problems. When the user feels confident about the accuracy and completeness of the new databases, the paper documents may be destroyed.

## CONVERTING TO THE NEW SYSTEM

The process of converting from the old system to the new one is called the **cutover**. A system cut-over will usually follow one of three approaches: cold turkey, phased, or parallel operation.

### Cold Turkey Cutover

Under the **cold turkey cutover** approach (also called the big bang approach), the firm switches to the new system and simultaneously terminates the old system. When implementing simple systems, this is often the easiest and least costly approach. With more complex systems, it is the riskiest. Cold turkey cutover is akin to skydiving without a reserve parachute. As long as the main parachute functions properly, there is no problem. But things don't always work the way they are supposed to. System errors that were not detected during the walk-through and testing steps may materialize unexpectedly. Without a backup system, an organization can find itself in serious trouble.

### Phased Cutover

Sometimes an entire system cannot, or need not, be cut over at once. The **phased cutover** begins operating the new system in modules. For example, Figure 14-20 shows how we might implement a system, starting with the sales subsystem, followed by the inventory control subsystem, and finally the purchases subsystem. By phasing in the new system in modules, we reduce the risk of a devastating system failure. However, the phased approach can create incompatibilities between new subsystems and yet-to-be-replaced old subsystems. Implementing special conversion systems that provide temporary interfaces during the cutover period may alleviate this problem.

### Parallel Operation Cutover

**Parallel operation cutover** involves running the old system and the new system simultaneously for a period of time. Figure 14-21 illustrates this approach, which is the most time-consuming and costly of the three. Running two systems in parallel essentially doubles resource consumption. During the cutover period, the two systems require twice the source documents, twice the processing time, twice the databases, and twice the output production.

The advantage of parallel cutover is the reduction in risk. By running two systems, the user can reconcile outputs to identify and debug errors before running the new system solo. Parallel

**FIGURE 14-20 PHASED CUTOVER**

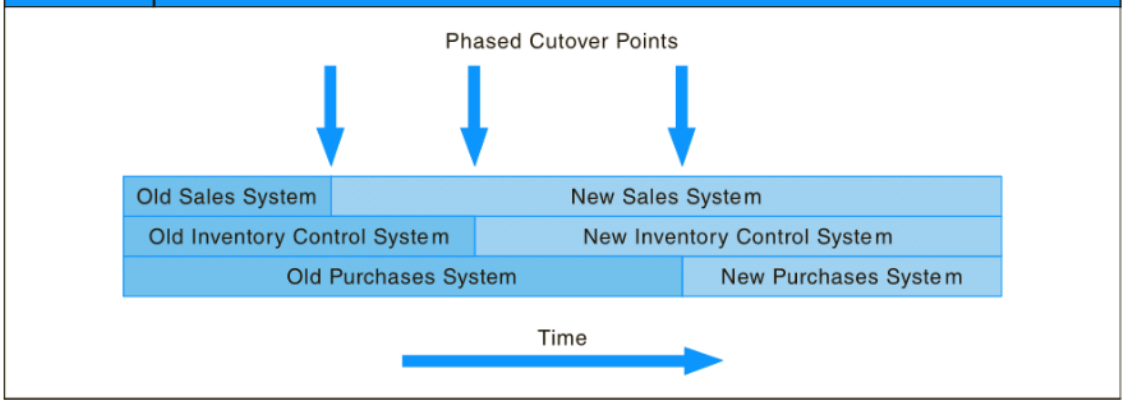
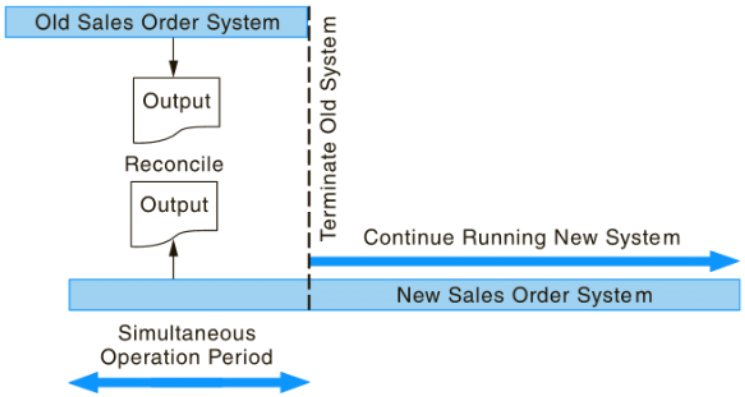


FIGURE 14-21 PARALLEL OPERATION CUTOVER






operation should usually extend for one business cycle, such as one month. This allows the user to reconcile the two outputs at the end of the cycle as a final test of the system's functionality.

## POST-IMPLEMENTATION REVIEW

The final step in the implementation phase actually takes place some months later in a post-implementation review. The objective is to measure the success of the system and of the process after the dust has settled. Although systems professionals strive to produce systems that are on budget, on time, and meet user needs, this does not always happen. The post-implementation review of the newly installed system can provide insight into ways to improve the process for future systems. The areas discussed in the following section are of particular concern.

### System Design Adequacy

The physical features of the system should be reviewed to see if they meet user needs. The reviewer should seek answers to the following types of questions:

1. Does the output from the system possess such characteristics of information as relevance, timeliness, completeness, accuracy, and so on?
2. Is the output in the format most useful and desired by the user (such as tables, graphs, electronic, hard copy, etc.)?
3. Are the databases accurate, complete, and accessible?
4. Did the conversion process lose, corrupt, or duplicate data?
5. Are input forms and screens properly designed and meeting users' needs?
6. Are the users using the system properly?
7. Does the processing appear to be correct?
8. Can all program modules be accessed and executed properly, or does the user ever get stuck in a loop?
9. Is user documentation accurate, complete, and easy to follow?
10. Does the system provide the user adequate help and tutorials?

### Accuracy of Time, Cost, and Benefit Estimates

Uncertainty complicates the task of estimating time, costs, and benefits for a proposed system. This is particularly true for large projects involving many activities and long time frames.

The more variables in the process, the greater the likelihood for material error in the estimates. History is often the best teacher for decisions of this sort. Therefore, a review of actual performance compared to budgeted amounts provides critical input for future budgeting decisions. From such information, we can learn where mistakes were made and how to avoid them the next time. The following questions provide some insight:

1. Were PERT and Gantt chart estimates accurate to within 10 percent?
2. What were the areas of significant departures from budget?
3. Were departures from the budget controllable (internal) in the short run or noncontrollable (e.g., supplier problems)?
4. Were estimates of the number of lines of program code accurate?
5. Was the degree of rework due to design and coding errors acceptable?
6. Were actual costs in line with budgeted costs?
7. Are users receiving the expected benefits from the system?
8. Do the benefits seem to have been fairly valued?

## THE ROLE OF ACCOUNTANTS

The role of accountants in the construct and deliver phases of the SDLC should be significant. Most system failures are due to poor designs and improper implementation. Being a major stakeholder in all financial systems, accountants must apply their expertise in this process to guide and shape the finished system. Specifically, accountants should get involved in the following ways.

### Provide Technical Expertise

The detailed design phase involves precise specifications of procedures, rules, and conventions to be used in the system. In the case of an AIS, these specifications must comply with GAAP, GAAS, SEC regulations, and IRS codes. Failure to so comply can lead to legal exposure for the firm. For example, choosing the correct depreciation method or asset valuation technique requires a technical background that systems professionals don't necessarily possess. The accountant must provide this expertise to the systems design process.

### Specify Documentation Standards

In the implementation phase, the accountant plays a role in specifying system documentation. Because financial systems must periodically be audited, they must be adequately documented. The accountant must actively encourage adherence to effective documentation standards.

### Verify Control Adequacy

The applications that emerge from the SDLC must possess controls that are in accordance with the provisions of Statement on Auditing Standards No. 78. This requires the accountant's involvement at both the detailed design and implementation phases. Controls may be programmed or manual procedures. Some controls are part of the daily operation of the system, while others are special actions that precede, follow, or oversee routine processing. The extent of control techniques makes it impossible to treat them within this chapter. Instead, we have devoted the next two chapters to the study of control concepts and design.

# Commercial Packages

Thus far we have examined system construction and delivery activities pertaining to in-house development. Not all systems are acquired in this fashion; the trend today is toward purchased software. Faced with many competing packages, each with unique features and attributes, management must choose the system and the vendor that best serves the needs of the organization. Making the optimal choice requires that this be an informed decision.

Before moving on to the next phase of the SDLC, we will examine the issues surrounding the purchase of commercial software. Our discussion will focus primarily on a technique that can help structure and evaluate the many intangible factors that complicate the process of selecting commercial software.

## TRENDS IN COMMERCIAL PACKAGES

Four factors have stimulated the growth of the commercial software market: (1) the relatively low cost of general commercial software as compared to customized software, (2) the emergence of industry-specific vendors who target their software to the needs of particular types of businesses, (3) a growing demand from businesses that are too small to afford an in-house systems development staff, and (4) the trend toward downsizing of organizational units and the resulting move toward the distributed data processing environment, which has made the commercial software option more appealing to larger organizations.

Indeed, organizations that maintain their own in-house systems development staff will purchase commercial software when the nature of their need permits. Commercial software can be divided into a number of general groups, which are discussed in the following section.

### Turnkey Systems

**Turnkey systems** are completely finished and tested systems that are ready for implementation. Often these are general-purpose systems or systems customized to a specific industry. Turnkey systems are usually sold only as compiled program modules, and users have limited ability to customize such systems to their specific needs. Some turnkey systems have software options that allow the user to customize input, output, and some processing through menu choices. Other turnkey system vendors will sell their customers the source code if program changes are desired. For a fee, the user or the vendor can then reprogram the original source code to customize the system. Some examples of turnkey systems are described in the following section.

**GENERAL ACCOUNTING SYSTEMS.** General accounting systems are designed to serve a wide variety of user needs. By mass producing a standard system, the vendor is able to reduce the unit cost of these systems to a fraction of in-house development costs. Powerful systems of this sort can be obtained for under $2,000.

To provide as much flexibility as possible, general accounting systems are designed in modules. This allows users to purchase the modules that meet their specific needs. Typical modules include accounts payable, accounts receivable, payroll processing, inventory control, general ledger, financial reporting, and fixed asset.

**SPECIAL-PURPOSE SYSTEMS.** Some software vendors have targeted their systems to selected segments of the economy. For example, the medical field, the banking industry, and government agencies have unique accounting procedures, rules, and conventions that general-purpose accounting systems do not always accommodate. Software vendors have thus developed standardized systems to deal with industry-specific procedures.

**OFFICE AUTOMATION SYSTEMS.** Office automation is the use of computer systems to improve the productivity of office workers. Examples of office automation systems include word processing packages, database management systems, spreadsheet programs, and desktop publishing systems.

### Backbone Systems

As we learned in Chapter 1, **backbone systems** provide a basic system structure on which to build. Backbone systems come with all the primary processing modules programmed. The vendor designs and programs the user interface to suit the client's needs. This approach can produce highly customized systems. But customizing a system is expensive and time-consuming. Many vendors thus

employ object-oriented systems design, which takes advantage of reusable modules and thereby reduces the costs of tailoring the system to the user.

### Vendor-Supported Systems

Vendor-supported systems are hybrids of custom systems and commercial software. Under this approach, the vendor develops (and maintains) custom systems for its clients. The systems themselves are custom products, but the systems development service is commercially provided. This option is popular in the health care and legal services industries. Because the vendor serves as the organization's in-house systems development staff, the client organization must rely on the vendor to provide custom programming and on-site maintenance of systems. Much of each client's system may be developed from scratch, but by using an object-oriented approach, vendors can produce common modules that can be reused in other client systems. This approach helps to reduce development costs charged to the client firms.

### ERP Systems

Enterprise resource planning (ERP) systems are difficult to classify into a single category because they have characteristics of all of the previously discussed systems. They are prewritten systems, which in some cases are implemented as turnkey applications. On the other hand, they can be modified to meet user needs. An ERP may be installed as a backbone system that interfaces with other legacy systems, or it may constitute an entirely new system. Because of their complexity, ERP systems are most often vendor-supported packages that an outside service provider installs.

## ADVANTAGES OF COMMERCIAL PACKAGES

### Implementation Time

Custom systems often take a long time to develop. Months or even years may pass before a custom system can be developed through in-house procedures. Unless the organization successfully anticipates future information needs and schedules application development accordingly, it may experience long periods of unsatisfied need. On the other hand, small commercial software systems can be implemented almost immediately upon recognizing a need. The user does not need to wait. The implementation of a single module of larger systems such as PeopleSoft, SAP, or ORACLE-FIN, however, could take from several weeks to a few months. An entire ERP could take years, but this is still much quicker than in-house or outsourced development would take.

### Cost

A single user must wholly absorb in-house development costs. However, because the cost of commercial software is spread across many users, the unit cost is reduced to a fraction of the cost of a system developed in-house.

### Reliability

Most reputable commercial software packages are thoroughly tested before their release to the consumer market. Any system errors that were not discovered during testing that organizations likely uncover shortly after release are corrected. Although no system is certified as being free from errors, commercial software is less likely to have errors than an equivalent in-house system.

## DISADVANTAGES OF COMMERCIAL PACKAGES

### Independence

Purchasing a vendor-supported system makes the firm dependent on the vendor for maintenance. The user runs the risk that the vendor will cease to support the system or even go out of business. This is perhaps the greatest disadvantage of vendor-supported systems.

### The Need for Customized Systems

The prime advantage of in-house development is the ability to produce applications to exact specifications. This advantage also describes a disadvantage of commercial software. Sometimes, the user's needs are unique and complex, and commercially available software is either too general or too inflexible.

### Maintenance

Business information systems undergo frequent changes. If the user's needs change, it may be difficult or even impossible to modify commercial software. On the other hand, in-house development provides users with proprietary applications that can be maintained.

## CHOOSING A PACKAGE

Having made the decision to purchase commercial software, the systems development team is now faced with the task of choosing the package that best satisfies the organization's needs. On the surface, there may appear to be no clear-cut best choice from the many options available. The following four-step procedure can help structure this decision-making process by establishing decision criteria and identifying key differences between options.

### Step 1: Needs Analysis

As with in-house development, the commercial option begins with an analysis of user needs. These are formally presented in a statement of systems requirements that provides a basis for choosing between competing alternatives. For example, the stated requirement of the new system may be to:

1. Support the accounting and reporting requirements of federal, state, and local agencies.
2. Provide access to information in a timely and efficient manner.
3. Simultaneously support both accrual accounting and fund accounting systems.
4. Increase transaction processing capacity.
5. Reduce the cost of current operations.
6. Improve user productivity.
7. Reduce processing errors.
8. Support batch and real-time processing.
9. Provide automatic general ledger reconciliations.
10. Be expandable and flexible to accommodate growth and changes in future needs.

The systems requirements should be as detailed as the user's technical background permits. Detailed specifications enable users to narrow the search to only those packages most likely to satisfy their needs. Although computer literacy is a distinct advantage in this step, the technically inexperienced user can still compile a meaningful list of desirable features that the system should possess. For example, the user should address such items of importance as compliance with accounting conventions, special control and transaction volume requirements, and so on.

### Step 2: Send Out the Request for Proposals

Systems requirements are summarized in a document called a **request for proposal (RFP)**, which is sent to each prospective vendor. A letter of transmittal accompanies the RFP to explain to the vendor the nature of the problem, the objectives of the system, and the deadline for proposal submission.

The RFP provides a format for vendor responses and thus a comparative basis for initial screening. Some vendors will choose not to respond to the RFP, while others will propose packages that clearly do not meet the stated requirements. The reviewer should attempt to select from these responses those proposals that are feasible alternatives.

## Step 3: Gather Facts

In this next step in the selection process, the objective is to identify and capture relevant facts about each vendor's system. The following describes techniques for fact gathering.

**VENDOR PRESENTATIONS.** At some point during the review, vendors should be invited to make formal presentations of their systems at the user's premises. This provides the principal decision makers and users with an opportunity to observe the product firsthand.

Technical demonstrations are usually given at these presentations using modified versions of the packages that run on microcomputers. This provides an opportunity to obtain answers to detailed questions. Sufficient time should therefore be allotted for an in-depth demonstration followed by a question-and-answer period. If vendor representatives are unable or unwilling to demonstrate the full range of system capabilities or to deal with specific questions from the audience, this may indicate a functional deficiency of the system.

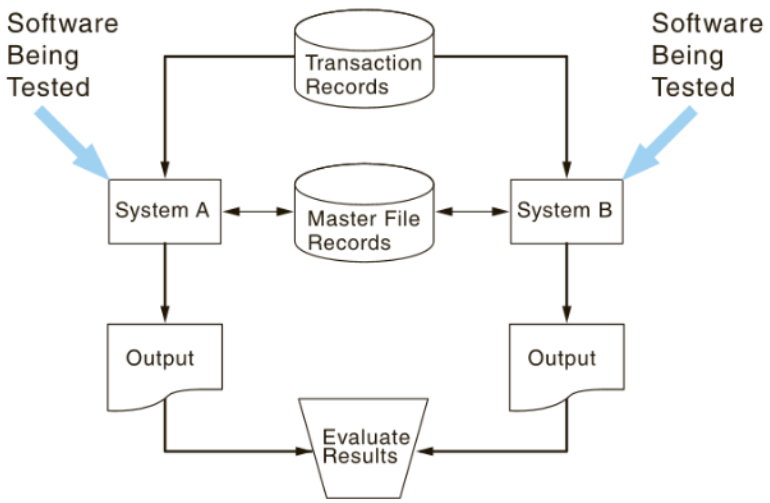
Failure to gain satisfactory responses from vendor representatives may also be a sign of their technical incompetence. The representatives either do not understand the user's problem or their own system and how it relates to the user's situation. In either case, the user has cause to question the vendor's ability to deliver a quality product and to provide adequate support.

**BENCHMARK PROBLEMS.** One often-used technique for measuring the relative performance of competing systems is to establish a benchmark problem for them to solve. The benchmark problem could consist of important transactions or tasks that key components of the system perform. In the benchmark example illustrated in Figure 14-22, both systems are given the same data and processing task. The results of processing are compared on criteria such as speed, accuracy, and efficiency in performing the task.

**VENDOR SUPPORT.** For some organizations, vendor support is an important criterion in systems selection. The desired level of support should be carefully considered. Organizations with competent in-house systems professionals may need less vendor support than firms without such internal resources. Support can vary greatly from vendor to vendor. Some vendors provide full-service support, including:

- Client training
- User and technical documentation
- Warranties

**FIGURE 14-22 BENCHMARK APPROACH TO TESTING COMPETING SOFTWARE**



- Maintenance programs to implement system enhancements
- Toll-free help numbers
- Annual seminars to obtain input from users and apprise them of the latest developments

At the other extreme, some vendors provide virtually no support. The buyer should be wary of a promise of support that seems too good to be true, because it probably is. The level of support the vendor provides can account for a large portion of its product's price. To avoid dump-and-run vendors, the buyer must be prepared to pay for support.

**CONTACT USER GROUPS.** A vendor's current user list is an important source of information. The prospective user, not the vendor, should select a representative sample of users with the latest version of the package and with similar computer configurations. A standard set of questions directed to these users will provide information for comparing packages. The following list provides examples of the types of questions to ask:

- When did you purchase the package?
- Which other vendors did you review?
- Why did you select this package?
- Are you satisfied with the package?
- Are you satisfied with the vendor support?
- Does the system perform as advertised?
- Were modifications required?
- What type of training did the vendor provide?
- What is the quality of the documentation?
- Have any major problems been encountered?
- Do you subscribe to the vendor's maintenance program?
- If so, do you receive enhancements?

To reiterate an important point, the prospective user, not the vendor, should select the user references. This provides for a more objective appraisal and reduces the possibility of receiving biased information from showcase installations.

## Step 4: Analyze the Findings and Make a Final Selection

The final step in the selection process is to analyze the facts and choose the best package. The principal problem is dealing with the many qualitative aspects of this decision. A popular technique for structuring and analyzing qualitative variables is the weighted factor matrix. The technique requires constructing a table similar to the one illustrated in Table 14-2. This table shows a comparison between only two proposals, those of Vendor A and Vendor B. In practice, the approach can be applied to all the vendors under consideration.

The table presents the relevant decision criteria under the heading *Factor*. Each decision factor is assigned a weight that implies its relative importance to the user. Two steps are critical to this analysis technique: (1) identify all relevant decision factors, and (2) assign realistic weights to each factor. Because these factors represent all of the relevant decision criteria, their weights should total 100 percent. These weights will likely vary among decision makers. One reviewer may consider vendor support an important factor and thus assign a high numeric value to its weight. Another decision maker may give this a low weight because support is not important in his or her firm, relative to other factors.

After assigning weights, each vendor package is evaluated according to its performance in each factor category. Based upon the facts gathered in the previous steps, each individual factor is scored on a scale of 1 to 5, where 1 is poor performance and 5 is excellent. The weighted scores are computed by multiplying the raw score by the weight for each factor. Using the previous example, a weight of 15 for vendor support is multiplied by a score of 4 for Vendor A and 3 for Vendor B, yielding the weighted scores of 60 and 45, respectively.

**TABLE 14-2 WEIGHTED FACTOR MATRIX**

| | | PROPOSAL A | | PROPOSAL B | |
|---|---|---|---|---|---|
| Factor | Weight | Raw Score | Weighted Score | Raw Score | Weighted Score |
| Response time | 10 | 5 | 50 | 4 | 40 |
| Compatibility | 9 | 3 | 27 | 5 | 45 |
| Reputation and experience | 5 | 3 | 15 | 5 | 25 |
| Ability to deliver on schedule | 7 | 4 | 28 | 5 | 35 |
| Range of capabilities | 15 | 4 | 60 | 4 | 60 |
| Modularity | 12 | 4 | 48 | 3 | 36 |
| User friendliness | 15 | 4 | 60 | 3 | 45 |
| Supports database | 9 | 2 | 18 | 5 | 45 |
| Supports networking | 3 | 2 | 6 | 5 | 15 |
| Vendor support | 15 | 4 | 60 | 3 | 45 |
| Total | 100 | | 372 | | 391 |



The weighted scores are then totaled, and each vendor is assigned a composite score. This is the vendor's overall performance index. Table 14-2 shows a score of 391 for Vendor B and 372 for Vendor A. This composite score suggests that Vendor B's product is rated slightly higher than Vendor A's.

This analysis must be taken a step further to include financial considerations. For example, assume Proposal A costs $150,000 and Proposal B costs $190,000. An overall performance/cost index is computed as follows.

$$\text{Proposal A}: \frac{372}{\$150{,}000} = 2.48 \text{ per } \$1{,}000$$

$$\text{Proposal B}: \frac{391}{\$190{,}000} = 2.06 \text{ per } \$1{,}000$$

This means that Proposal A provides 2.48 units of performance per $1,000 versus only 2.06 units per $1,000 from Proposal B. Therefore, Proposal A provides the greater value for the cost.

The option with the highest performance/cost ratio is the more economically feasible choice. Of course, this analysis rests on the user's ability to identify all relevant decision factors and assign to them weights that reflect their relative importance to the decision. If any relevant factors are omitted or if their weights are misstated, the results of the analysis will be misleading.

# Maintenance and Support

Maintenance involves both implementing the latest software versions of commercial packages and making in-house modifications to existing systems to accommodate changing user needs. Maintenance may be relatively trivial, such as modifying an application to produce a new report, or more extensive, such as programming new functionality into a system.

Some organizations view systems maintenance services as commodity activities that should be outsourced to third-party vendors on a low-cost bidder basis. The underlying justification for this is short-term economic benefit. By outsourcing maintenance and support, management can channel financial resources into the organization's core competencies. Unfortunately, isolating maintenance activities

from the organization also disrupts the flow of system-related knowledge that may be of strategic importance to the organization.

Some organizations take a strategic view of maintenance. Maintenance is an integral part of the SDLC. Rather than representing the end of the line, it can be an incubator for new ideas. If management captures the appropriate data, each currently running system can be the prototype for the next version. To ensure success, the organization needs to collect all relevant data from comments, requests, observed symptoms, and ideas for improvement from the user community.

## USER SUPPORT

Typically, the first point of contact for such data transfer is through the user support function. This includes help desk services, user training and education classes, and formally documented user feedback pertaining to problems and system errors. To facilitate data gathering and analysis, **knowledge management** systems are effective maintenance tools.

## KNOWLEDGE MANAGEMENT AND GROUP MEMORY

Knowledge management is a concept consisting of four basic processes: gathering, organizing, refining, and disseminating. **Gathering** brings data into the system. **Organizing** associates data items with subjects, giving them context. **Refining** adds value by discovering relationships between data, performing synthesis, and abstracting. **Disseminating** gets knowledge to the recipients in a usable form. The most difficult of these processes to automate is refining.

A knowledge management system can be used to create a **group memory**, which makes an organization more effective, just as human beings become more effective and mature with the accumulation of thoughts and memories. From a technology viewpoint, a knowledge management system is a database-oriented software tool that allows users, developers, and the operations community to contribute to the group memory. Contributors add their comments, suggestions, or complaints about a system or process into forms from their desktop PCs. The knowledge management software uses a parsing utility that takes incoming strings of data and infers relationships from them. A notable strength of the system is that it can deal with both historical and emerging data. The goal of the system is not simply to store information in a central repository for record keeping or archival recall. Rather, it analyzes the heterogeneous data and disseminates information to users and systems management. Group memory is thus a potentially valuable input to the organization's evolving systems strategy.

# Summary

The chapter dealt with the construction and delivery of information systems. The first section presented topics and issues related to in-house development. This began with a review of techniques used for improving systems construction, including prototyping, CASE technology, PERT charts, and Gantt charts. Next, we discussed two design approaches: the structured approach and the object-oriented approach. The discussion followed a design sequence that dealt with system components in the following order: create a data model of the business process, define conceptual user views, design the normalized database tables, design the physical user views (output and input views), develop the process modules, specify the system controls, and perform a system walkthrough. The delivery stage involves populating database structures, purchasing and installing equipment, employee training, and system documentation. This phase concludes with the roll-out of the new system and the termination of the old system. The section concluded with a discussion of the accountant's role in in-house development.

We next examined issues related to commercial software, an option that businesses are increasingly using. After briefly identifying the pros and cons of commercial software, we examined a four-step procedure that can be employed in the selection of commercial software packages. The chapter concluded with a brief discussion of the strategic role of system maintenance and the importance of group memory as a key input to systems strategy.