UNIT-FIVE
Implementation and Maintenance

- Implementation and maintenance are the last two phases of the systems development life cycle. **The purpose of implementation** is to build a properly working system, install it in the organization, replace old systems and work methods, finalize system and user documentation, train users, and prepare support systems to assist users. Implementation also involves closedown of the project, including evaluating personnel, reassigning staff, assessing the success of the project, and turning all resources over to those who will support and maintain the system. **The purpose of maintenance** is to fix and enhance the system to respond to problems and changing business conditions. Maintenance includes activities from all systems development phases. Maintenance also involves responding to requests to change the system, transforming requests into changes, designing the changes, and implementing them.

- **System Implementation (453)**

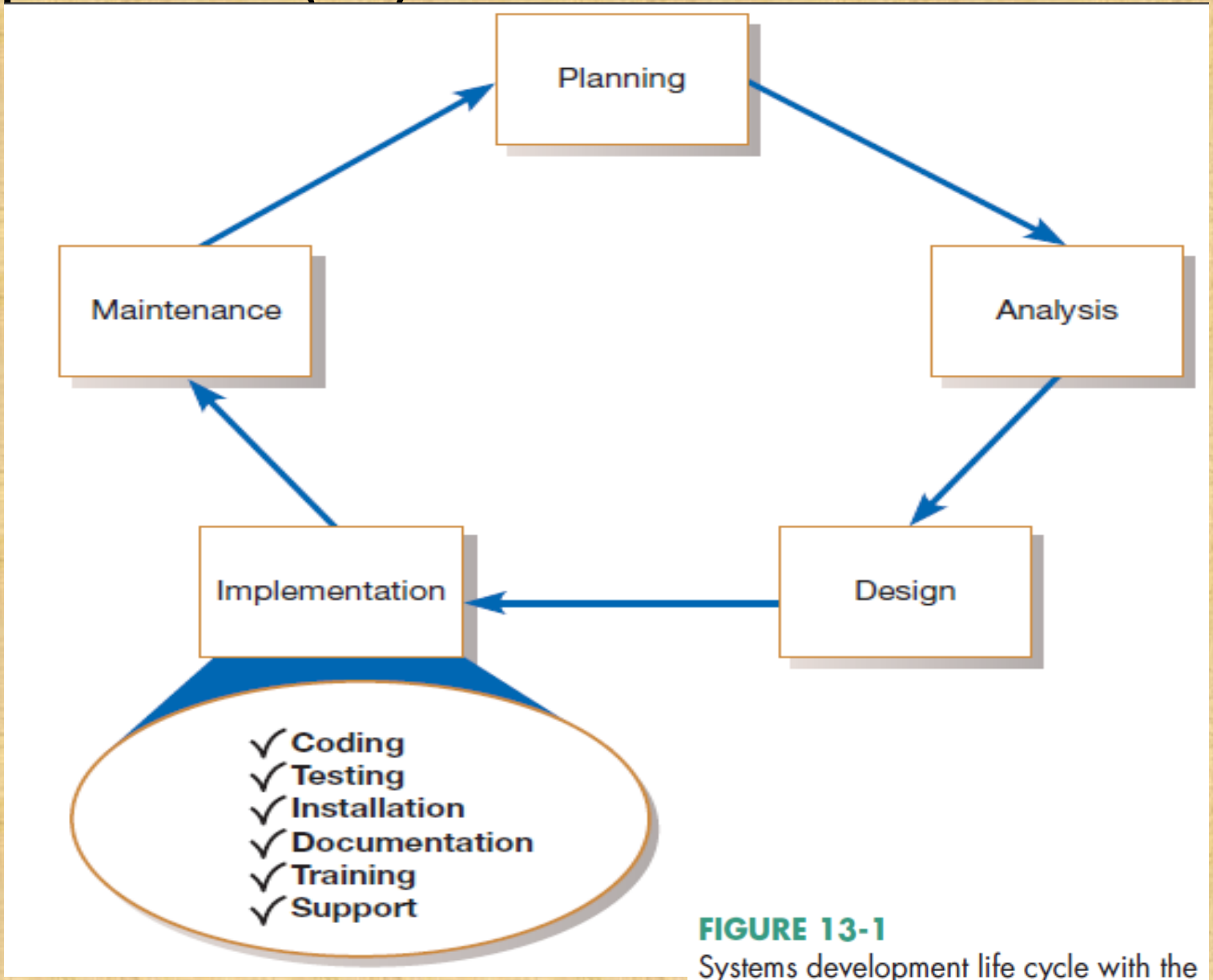

FIGURE 13-1
Systems development life cycle with the implementation phase highlighted

- **System implementation** is made up of many activities. The six major activities we are concerned with this are **coding, testing, installation**, **documentation, training, and support** (see Figure 13-1). The purpose of these steps is to convert the physical system specifications into working and reliable software and hardware, document the work that has been done, and provide help for current and future users and caretakers of the system. Coding and testing may have already been completed by this point if Agile Methodologies have been followed. Using a plan-driven methodology, coding and testing are often done by other project team members besides analysts, although analysts may do some programming. In any case, analysts are responsible for ensuring that all of these various activities are properly planned and executed. Next, we will briefly discuss these activities in two groups:

  (1) **coding, testing, and installation** and (2) **documenting the system and training and supporting users.**

- **Coding, testing, and Installation processes**
- **Coding** is the process whereby the physical design specifications

  created by the analysis team are turned into working computer code by the programming team. Depending on the size and complexity of the system, coding can be an involved, intensive activity. once coding has begun, the testing process can begin and proceed in parallel. As each program module is produced, it can be tested individually, then as part of a larger program, and then as part of a larger system.
- **Installation** is the process during which the current system is replaced by the new system. This includes conversion of existing data, software, documentation, and work procedures to those consistent with the new system. Users will sometimes resist these changes, and you must help them adjust. However, you cannot control all the dynamics of user–system interaction involved in the installation process.

- **The Processes of Documenting the System, Training Users, and Supporting Users:** Although the process of documentation proceeds throughout the life cycle, it receives formal attention during the implementation phase because the end of implementation largely marks the end of the analysis team's involvement in systems development. As the team is getting ready to move on to new projects, you and the other analysts need to prepare documents that reveal (**give information**) all of the important information you have accumulated about this system during its development and implementation. There are two audiences for this final documentation:

  (1) the information systems personnel who will maintain the system throughout its productive life, and **(2)** the people who will use the system as part of their daily lives. The analysis team in a large organization can get help in preparing documentation from specialized staff in the information systems department.

- Larger organizations also tend to provide training and support to computer users throughout the organization. Some of the training and support is very specific to particular application systems, whereas the rest is general to particular operating systems or off-the-shelf software packages.

- **Software application testing**
- Software testing can be stated as the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by its design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases. Software testing is method of assessing the functionality of a software program. The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy and usability. It mainly aims at measuring specification, functionality and performance of a software program or application.
- **Seven Different types of tests**
- **Static testing means** that the code being tested is not executed. The results of running the code are not an issue for that particular test. **Dynamic testing**, on the other hand, involves execution of the code. **Automated testing** means the computer conducts the test, whereas **manual testing means** that people complete the test.

- **1. Inspections:** A testing technique in which participants examine program code for predictable language-specific errors **that is** participants manually examine code for occurrences of well-known errors. Syntax, grammar, and some other routine errors can be checked by automated inspection software, so manual inspection checks are used for more subtle (small) errors. Each programming language lends itself to certain types of errors that programmers make when coding, and these common errors are well-known and documented. **Exactly what the code does is not investigated in an inspection**. It has been estimated that code inspections detect from **60 to 90 percent** of all software defects as well as provide programmers with feedback that enables them to avoid making the same types of errors in future work (Fagan, 1986).
- **2. Desk checking :** A testing technique in which the program code is sequentially executed manually by the reviewer. It is informal process in which the programmer or someone else who understands the logic of the program works **through the code with a paper and pencil**. The programmer executes each instruction, **using test cases** that may or may not be written down. In one sense, **the reviewer acts as the computer, mentally checking each step** and its results for the entire set of computer instructions.

- **3. Unit testing:** sometimes called module testing, is an automated technique whereby **each module** is tested alone in an attempt to discover any errors that may exist in the module's code. But because modules coexist and work with other modules in programs and the system, they must also be tested together in larger groups.
- **4. Integration testing:** Combining modules and testing them is called integration testing**.** Integration testing is gradual. First you test the coordinating module and only one of its subordinate modules. After the first test, you add one or two other subordinate modules from the same level. Once the program has been tested with the coordinating module and all of its immediately subordinate modules, you add modules from the next level and then test the program. You continue this procedure until the entire program has been tested as a unit.
- **5. System testing:** System testing is a similar process, but instead of integrating modules into programs for testing, you integrate programs into systems. System testing follows the same incremental logic that integration testing does. Under both integration and system testing, not only do individual modules and programs get tested many times, so do the interfaces between modules and programs.

- Current practice calls for a top-down approach to writing and testing modules. Under a **top-down approach**, the **coordinating module** is written first. Then the modules at the next level in the structure chart are written, followed by the modules at the next level, and so on, until all of the modules in the system are done. Each module is tested as it is written. Because top-level modules contain many calls to subordinate modules. System testing is more than simply expanded integration testing where you are testing the interfaces between programs in a system rather than testing the interfaces between modules in a program. System testing can be performed in two ways:

  5.1 Black box testing: It is defined as a testing technique in which functionality of the application under test is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications. In this testing, we just focus on inputs and output of the software system without bothering about internal knowledge of the software program. In black box test (also called functional test) internal code of the program are tested. It is called black box testing because the test cases are totally hidden for the general users.

- **5.2 White box testing:** White box testing is a testing technique that examines the program structure and derives test data from the program logic/code. It is a software testing methodology that uses a program's source code to design tests and test cases for quality assurance (QA). The code structure is known and understood by the tester in white box testing. In white box test (also called glass box test) structure of the program is tested. It is called white box testing because the test cases are totally visible to the general users and they can also make test cases.
- **6. Stub testing**: Stubs are two or three lines of code written by a programmer to stand in for the missing modules. During testing, the coordinating module calls the stub instead of the subordinate module. The stub accepts control and then returns it to the coordinating module.
- **7. User acceptance testing:** Once the system tests have been satisfactorily completed, the system is ready for acceptance testing, which is testing the system in the environment where it will eventually be used.

- **Installation (464)**

  The process of moving from the current information system to the new one is called **installation.** All employees who use a system, whether they were consulted during the development process or not, must give up their reliance on the current system and begin to rely on the new system. Four different approaches to installation have emerged over the years: **direct, parallel, single-location, and phased** (Figure 13-5). The approach an organization decides to use will depend on the scope and complexity of the change associated with the new system and the organization's risk aversion.

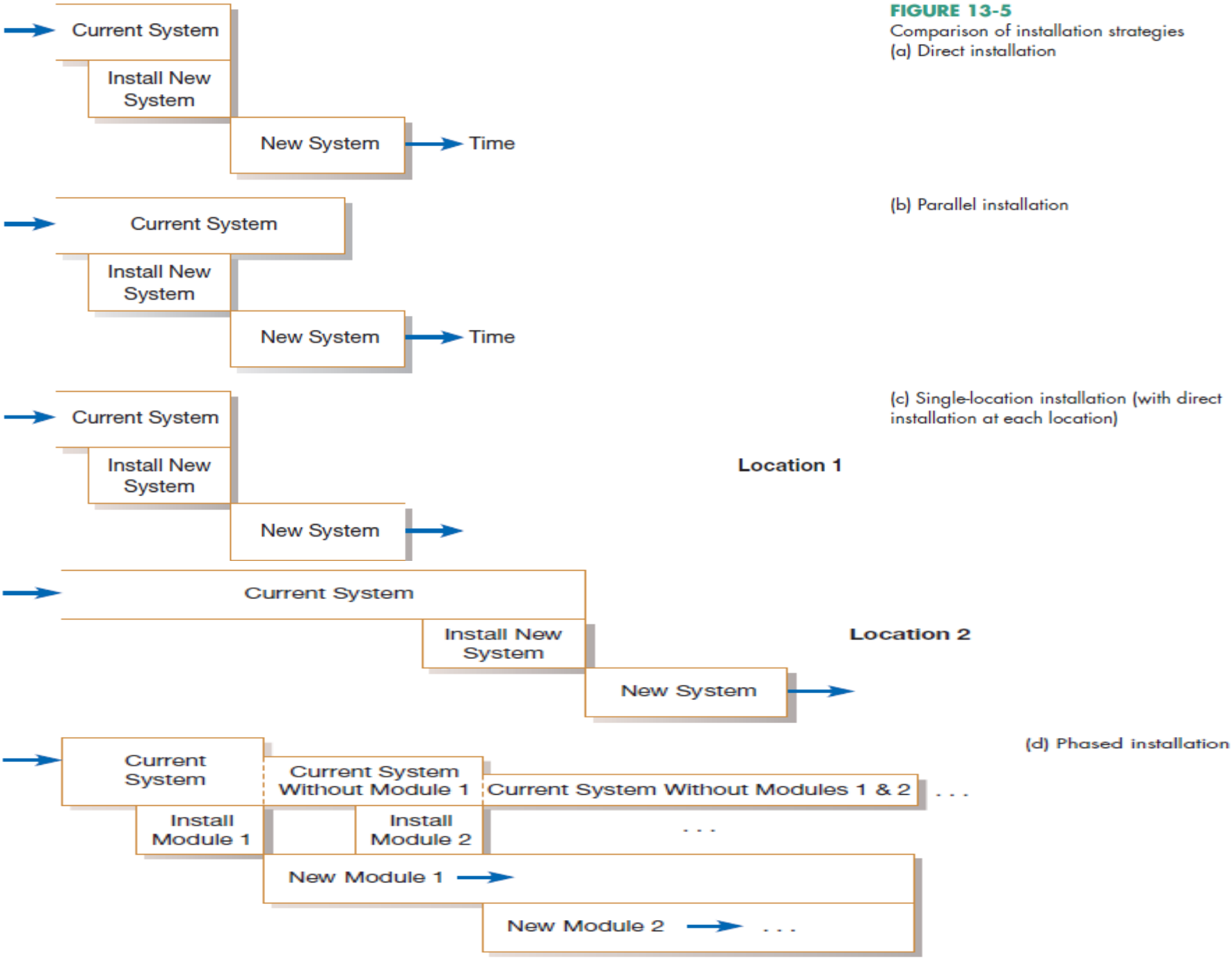

FIGURE 13-5
Comparison of installation strategies
(a) Direct installation

(b) Parallel installation

(c) Single-location installation (with direct installation at each location)

(d) Phased installation

- **Direct Installation**
- The direct, or abrupt, approach to installation (also called "**cold turkey**") is as sudden as the name indicates: The old system is turned off and the new system is turned on (Figure 13-5a). Under **direct installation, users are at the mercy of the** new system. Any errors resulting from the new system will have a direct impact on the users and how they do their jobs and, in some cases—depending on the centrality of the system to the organization—on how the organization performs its business. If the new system fails, considerable delay may occur until the old system can again be made operational and business transactions are reentered to make the database up to date**. For these reasons, direct installation can be very risky**. Further, direct installation requires a complete installation of the whole system. For a large system, this may mean a long time until the new system can be installed, thus delaying system benefits or even missing the opportunities that motivated the system request. On the other hand, it is the least expensive installation method, and it creates considerable interest in making the installation a success. Sometimes, a direct installation is the only possible strategy if there is no way for the current and new systems to coexist, which they must do in some way in each of the other installation approaches.

- **Parallel installation** is as riskless as direct installation is risky. Under parallel installation, the old system continues to run alongside the new system until users and management are satisfied that the new system is effectively performing its duties and the old system can be turned off (Figure 13-5b). All of the work done by the old system is concurrently performed by the new system. Outputs are compared (to the greatest extent possible) to help determine whether the new system is performing as well as the old. Errors discovered in the new system do not cost the organization much, if anything, because errors can be isolated and the business can be supported with the old system. Because all work is essentially done twice, a parallel installation can be very expensive; running two systems implies employing (and paying) two staffs to not only operate both systems, but also to maintain them. A parallel approach can also be confusing to users because they must deal with both systems. As with direct installation, there can be a considerable delay until the new system is completely ready for installation. A parallel approach may not be feasible, especially if the users of the system (such as customers) cannot tolerate redundant effort or if the size of the system (number of users or extent of features) is large.

- **Single-location installation,** also known as location **or pilot installation**, is a middleof- the-road approach compared with direct and parallel installation. Rather than convert all of the organization at once, single-location installation involves **changing from the current to the new system in only one place or in a series of separate sites over time**. (Figure 13-5c depicts this approach for a simple situation of two locations.) The single location may be a branch office, a single factory, or one department, and the actual approach used for installation in that location may be any of the other approaches. The key advantage to single-location installation is that it limits potential damage and potential cost by limiting the effects to a single site. Once management has determined that installation has been successful at one location, the new system may be deployed in the rest of the organization, possibly continuing with installation at one location at a time. Success at the pilot site can be used to convince reluctant personnel at other sites that the system can be worthwhile for them as well. Problems with the system (the actual software as well as documentation, training, and support) can be resolved before deployment to other sites.

Even though the single-location approach may be simpler for users, it still places a large burden on information systems (IS) staff to support two versions of the system. On the other hand, because problems are isolated at one site at a time, IS staff members can devote all of their efforts to success at the pilot site. Also, if different locations require sharing of data, extra programs will need to be written to synchronize the current and new systems; although this will happen transparently to users, it is extra work for IS staff. As with each of the other approaches (except phased installation), the whole system is installed; however, some parts of the organization will not get the benefits of the new system until the pilot installation has been completely tested.

- **Phased installation,** also called **staged installation**, is an incremental approach**.** With phased installation, the new system is brought online in functional components; different parts of the old and new systems are used in cooperation until the whole new system is installed. (Figure 13-5d shows the phase-in of the first two modules of a new system.) Phased installation, like single-location installation, is an attempt to limit the organization's exposure to risk, whether in terms of cost or disruption of the business. By converting gradually, the organization's risk is spread out over time and place. Also, a phased installation allows for some benefits from the new system before the whole system is ready. For example, new data-capture methods can be used before all reporting modules are ready. For a phased installation, the new and replaced systems must be able to coexist and probably share data. Thus, bridge programs connecting old and new databases and programs often must be built. Sometimes, the new and old systems are so incompatible (built using totally different structures) that pieces of the old system cannot be incrementally replaced, so this strategy is not feasible. A phased installation is akin to bringing out a sequence of releases of the system. Thus, a phased approach requires careful version control, repeated conversions at each phase, and a long period of change, which may be frustrating and confusing to users. On the other hand, each phase of change is smaller and more manageable for all involved.

- **Documenting the System (468)**
- Documentation is the process of collecting, organizing, storing and maintaining a complete record of system and other documents used or prepared during the different phases of the life cycle of system. System cannot be considered to be complete, until it is properly documented. Proper documentation of system is necessary due to the following reasons:
- 1. It solves the problem of indispensability (not willing) of an individual for an organization. Even if the person, who has designed or developed the system, leaves the organization, the documented knowledge remains with the organization, which can be used for the continuity of that software.
- 2. It makes system easier to modify and maintain in the future. The key to maintenance is proper and dynamic documentation. It is easier to understand the concept of a system from the documented records.
- 3. It helps in restarting a system development, which was postponed due to some reason. The job need not be started from scratch, and old ideas may still be easily recapitulated from the available documents, which avoids duplication of work, and saves lot of times and effort.

- **Types of Documentation:**
- **1. System Documentation**: System documentation records detailed information about a system's design specification, its internal workings and its functionality. System documentation is intended primarily for maintenance programmers. It contains the following information:
- A description of the system specifying the scope of the problem, the environment in which it functions, its limitation, its input requirements, and form and types of output required.
- Detailed diagram of system flowchart and program flowchart.
- A source listing of all the full details of any modifications made since its development.
- Specification of all input and output media required for the operation of the system.
- Problem definition and the objective of developing the programs.
- Output and test report of the program.
- Upgrade or maintenance history, if modification of the program is made.

- There are two types of system documentation. They are:
- i) **Internal documentation**: Internal documentation is part of the program source code or is generated at compile time.
- ii) **External documentation:** External documentation includes the outcome of structured diagramming technique such as dataflow an entity-relationship diagrams.
- **2. User documentation:** User documentation consists of written or other visual information about an application system, how it works and how to use it. User documentation is intended primarily for users. It contains the following information:
- Set up and operational details of each system.
- Loading and unloading procedures.
- Problems which could arise, their meaning reply and operation action.
- Special checks and security measures.
- Quick reference guides about operating a system in a short, concise format.

- **Training And Supporting Users**

  The type of training needed will vary by system type and user expertise. Types of training methods are:
- Resident expert (to fellow users for training).
- Traditional instructor-led classroom training.
- E-learning/ distance learning.
- Blended learning (combination of instructor-led and e-learning).
- Software help components.
- Electronic performance support system: component of a software package or an application in which training and educational information is embedded.
- External sources, such as vendors.
- Computing supports for users has been provided in one of a few forums:
- i) **Automating support**: Online support forums provides users access to information on new releases, bugs and tips form more effective usage. Forums are offered over the internet or over company intranets.

- **ii) Providing support through a help desk:** A help desk is an information systems department function and is staffed by IS personnel. The help desk is the first place users should call when they need assistance with an information system. The help desk staff members either deal with the users questions or refer the users to the most appropriate person.

- **Organizational Issues In Systems Implementation (474)**
- The best efforts of the systems development team is to design and build a quality system and to manage the change process in the organization, the implementation effort sometimes fails. Sometimes employees will not use the new system that has been developed for them or, if they do use it, their level of satisfaction with it is very low. **Why do systems implementation efforts fail?** This question has been the subject of information systems research for over 60 years.
- **Why Implementation Sometimes fails?**
- The conventional wisdom that has emerged over the years is that there are at **least two conditions necessary for a successful implementation effort**: management support of the system under development and the involvement of users in the development process (Ginzberg, 1981b). Conventional wisdom holds that if both of these conditions are met, you should have a successful implementation.

- **Management support and user involvement are important** to implementation success, but they may be overrated compared to other factors that are also important. Research has shown that **the link between user involvement and implementation** success is sometimes weak (Ives and Olson, 1984). User involvement can help reduce the risk of failure when the system is complex, but user participation in the development process only makes failure more likely when there are financial and time constraints in the development process (Tait and Vessey, 1988). Information systems implementation failures are too common, and the implementation process is too complicated, for the conventional wisdom to be completely correct.
- Over the years, other studies have found evidence of **additional factors** that are important to a successful implementation process. **Three such factors are**: **commitment to the project**, **commitment to change**, and the **extent of project definition and planning** (Ginzberg, 1981b).

- **Commitment to the project involves** managing the systems development project so that the **problem being solved** is well understood and the system being developed to deal with the problem actually solves it.
- **Commitment to change** involves being willing to change behaviors, procedures, and other aspects of the organization.
- **The extent of project definition and planning** is a measure of how well the project was planned. The more extensive the planning effort is, the less likely implementation failure is. Still another important factor related to implementation success is user expectations (Ginzberg, 1981a). The more realistic a user's early expectations about a new system and its capabilities are, the more likely it is that the user will be satisfied with the new system and actually use it.

- **System Maintenance (486)**
- **Introduction**
- In this chapter, we discuss systems maintenance, the **largest systems development expenditure for many organizations**. In fact, more programmers today work on maintenance activities than work on new development. Your first job after graduation may very well be as a maintenance programmer/analyst. This disproportionate (too large or too small in comparison to something else) distribution of maintenance programmers is interesting because software does not wear out in a physical manner as do buildings and machines. **There is no single reason** why software is maintained; however, most reasons relate to **a desire to evolve system functionality in order to overcome internal processing errors** or to better support changing business needs. Thus, maintenance is a fact of life for most systems. This means that maintenance can begin soon after the system is installed.

- **Maintaining Information Systems:** Once an information system is installed, the system is essentially in the **maintenance phase** of the systems development life cycle (SDLC). When a system is in the maintenance phase, some person within the systems development group is responsible for collecting maintenance requests from system users and other interested parties, such as system auditors, data center and network management staff, and data analysts. Once collected, each request is analyzed to better understand how it will alter the system and what business benefits and necessities will result from such a change. If the change request is approved, a system change is designed and then implemented. As with the initial development of the system, implemented changes are formally reviewed and tested before installation into operational systems.
- **The Process of Maintaining information Systems**
- As we can see in Figure 14-1, the maintenance phase is the last phase of the SDLC. It is here that the SDLC becomes a cycle, with the last activity leading back to the first. This means that the process of maintaining an information system is the process of returning to the beginning of the SDLC and repeating development steps until the change is implemented.

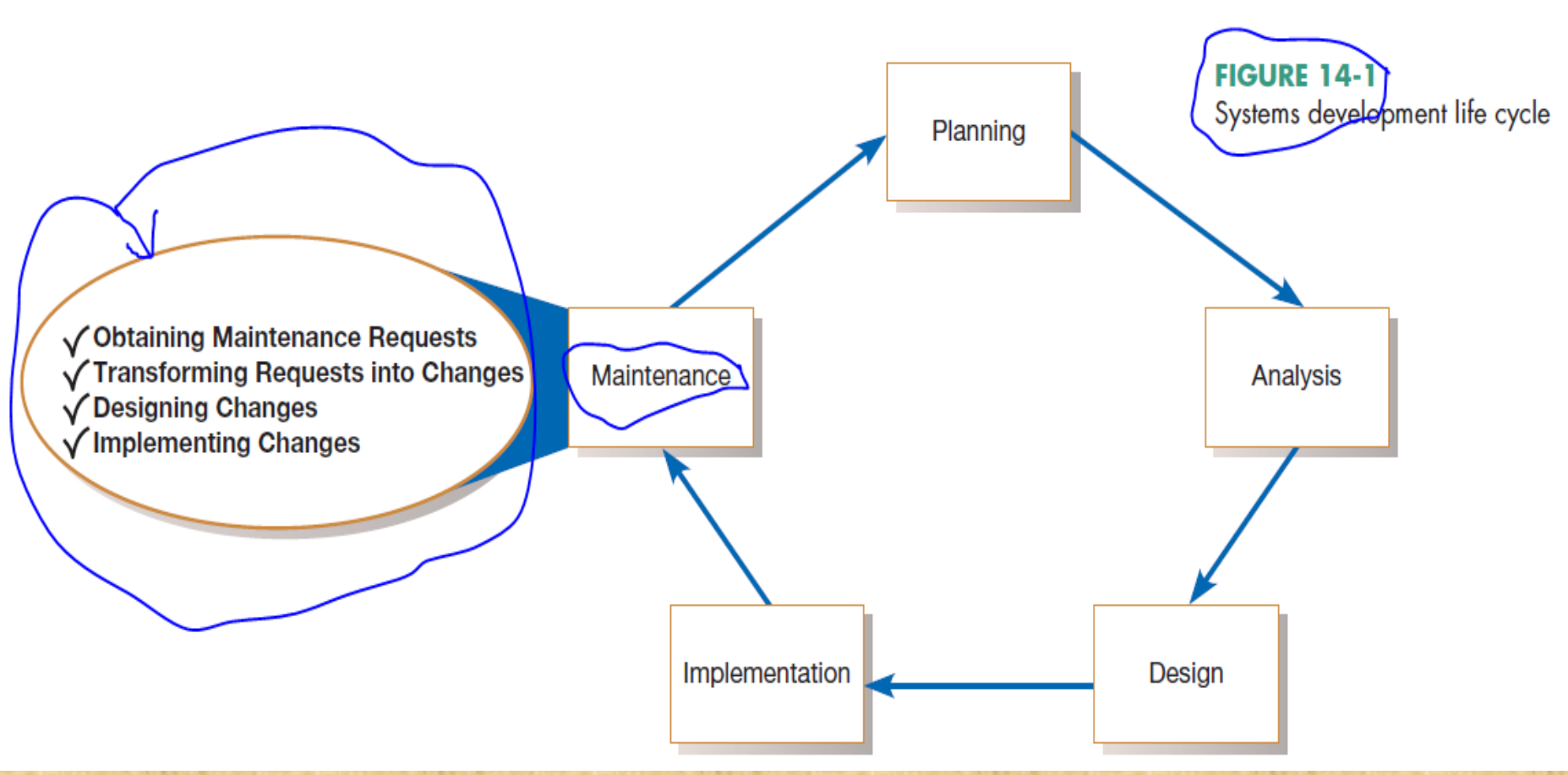
FIGURE 14-1
Systems development life cycle
Planning
Analysis
Design
Implementation
Maintenance
✓Obtaining Maintenance Requests
✓Transforming Requests into Changes
✓Designing Changes
✓Implementing Changes

- Also shown in Figure 14-1, four major activities occur within maintenance:

  1. **Obtaining maintenance requests**
  2. **Transforming requests into changes**
  3. **Designing changes**
  4. **Implementing changes**

- Obtaining maintenance requests requires that a formal process be established whereby users can submit system change requests. Earlier in this book, we presented a user request document called a **System Service Request (SSR**), which is shown in Figure 14-2 (in next slide). Most companies have some sort of document like an SSR to request new development, to report problems, or to request new features within an existing system. When developing the procedures for obtaining maintenance requests, organizations must also specify an individual within the organization to collect these requests and manage their dispersal to maintenance personnel.

Pine Valley Furniture
System Service Request

REQUESTED BY Juanita Lopez DATE November 5, 2017

DEPARTMENT Purchasing, Manufacturing Support

LOCATION Headquarters, 1-322

CONTACT Tel: 4-3267 FAX: 4-3270 e-mail: jlopez

TYPE OF REQUEST

[X] New System
[ ] System Enhancement
[ ] System Error Correction

URGENCY

[ ] Immediate—Operations are impaired or opportunity lost
[ ] Problems exist, but can be worked around
[X] Business losses can be tolerated until new system is installed

PROBLEM STATEMENT

Sales growth at PVF has caused greater volume of work for the manufacturing support unit within Purchasing. Further, more concentration on customer service has reduced manufacturing lead times, which puts more pressure on purchasing activities. In addition, cost-cutting measures force Purchasing to be more aggressive in negotiating terms with vendors, improving delivery times, and lowering our investments in inventory. The current modest systems support for manufacturing purchasing is not responsive to these new business conditions. Data are not available, information cannot be summarized, supplier orders cannot be adequately tracked, and commodity buying is not well supported. PVF is spending too much on raw materials and not being responsive to manufacturing needs.

SERVICE REQUEST

I request a thorough analysis of our current operations with the intent to design and build a completely new information system. This system should handle all purchasing transactions, support display and reporting of critical purchasing data, and assist purchasing agents in commodity buying.

IS LIAISON Chris Martin (Tel: 4-6204 FAX: 4-6200 e-mail: cmartin)

SPONSOR Sal Divario, Director, Purchasing

---------- TO BE COMPLETED BY SYSTEMS PRIORITY BOARD ----------

[ ] Request approved Assigned to
Start date
[ ] Recommend revision
[ ] Suggest user development
[ ] Reject for reason

- Once a request is received, analysis must be conducted to gain an understanding of the scope of the request. It must be determined how the request will affect the current system and how long such a project will take. As with the initial development of a system, the size of a maintenance request can be analyzed for risk and feasibility. Next, a change request can be transformed into a formal design change, which can then be fed into the maintenance implementation phase. Thus, many similarities exist between the SDLC and the activities within the maintenance process. Figure 14-3 equates SDLC phases to the maintenance activities described previously. The first phase of the SDLC—planning—is analogous to the maintenance process of obtaining a maintenance request (step 1). The SDLC analysis phase is analogous to the maintenance process of transforming requests into a specific system change (step 2). The SDLC design phase, of course, equates to the designing changes process (step 3). Finally, the SDLC phase implementation equates to step 4, implementing changes. This similarity between the maintenance process and the SDLC is no accident. The concepts and techniques used to initially develop a system are also used to maintain it.

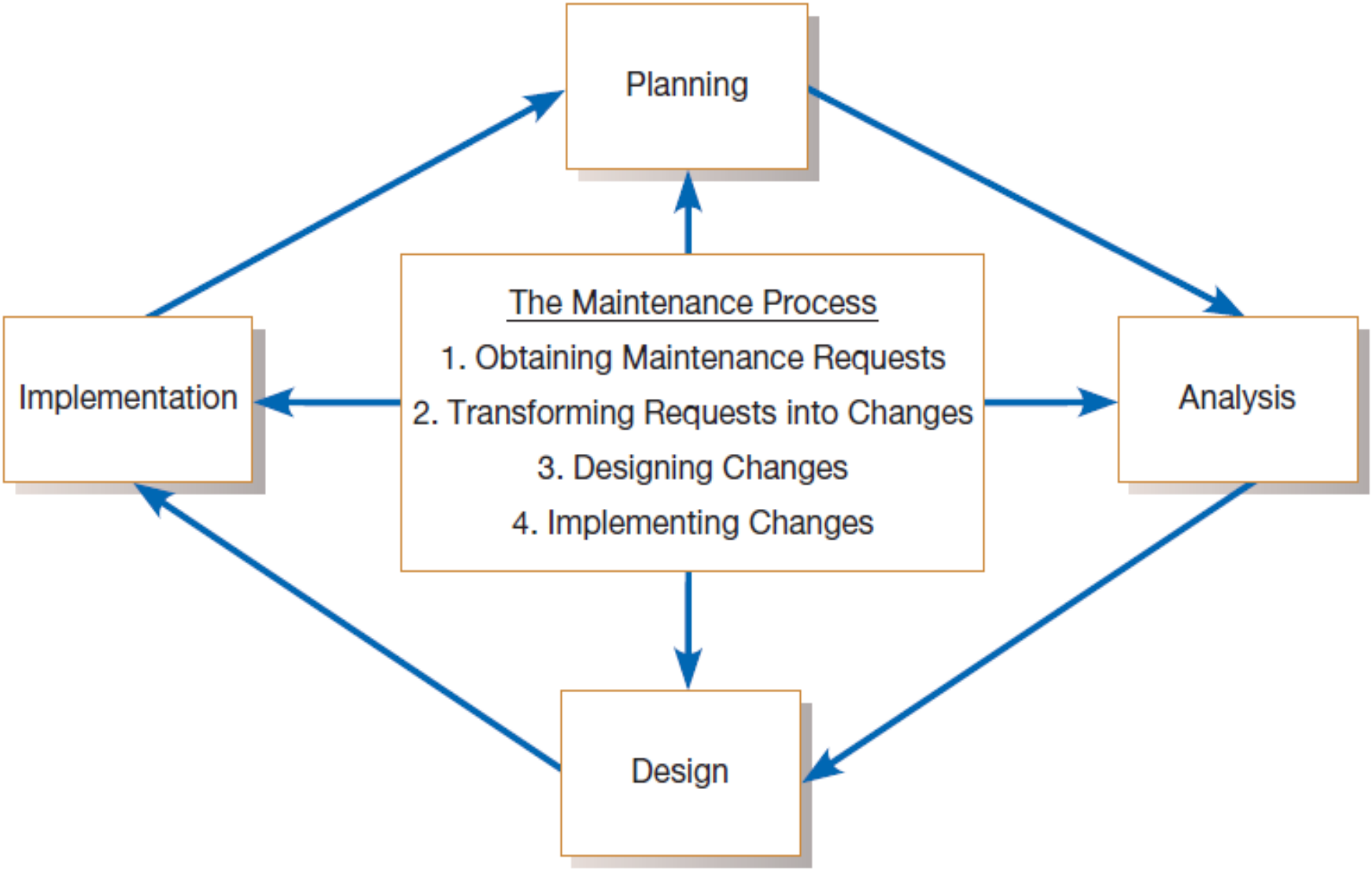

FIGURE 14-3
Maintenance activities parallel those of the SDLC

- **Conducting Systems Maintenance**
- A significant within organizations does not go to the development of new systems but to the **maintenance of existing systems**. We will describe various types of maintenance, factors influencing the complexity and cost of maintenance, and alternatives for managing maintenance.
- **Types of Maintenance**

  By maintenance, we mean the fixing or enhancing of an information system. **Corrective maintenance** refers to changes made to repair defects in the design, coding, or implementation of the system. For example, if you had recently purchased a new home, corrective maintenance would involve repairs made to things that had never worked as designed, such as a faulty electrical outlet or a misaligned door. Most corrective maintenance problems surface soon after installation. When corrective maintenance problems surface, they are typically urgent and need to be resolved to curtail possible interruptions in normal business activities. Of all types of maintenance, corrective accounts for as much as 75 percent of all maintenance activity (Andrews and Leventhal, 1993; Pressman, 2005).

- This is unfortunate because corrective maintenance adds little or no value to the organization; it simply focuses **on removing defects from an existing system without adding new functionality** (see Figure 14-4).
- **Adaptive maintenance** involves making changes to an information system to evolve its functionality to changing business needs or to migrate it to a different operating environment. Within a home, adaptive maintenance might be adding storm windows to improve the cooling performance of an air conditioner. Adaptive maintenance is usually less urgent than corrective maintenance because business and technical changes typically occur over some period of time. **Contrary to corrective maintenance**, adaptive maintenance is generally a small part of an organization's maintenance effort, but it adds value to the organization.

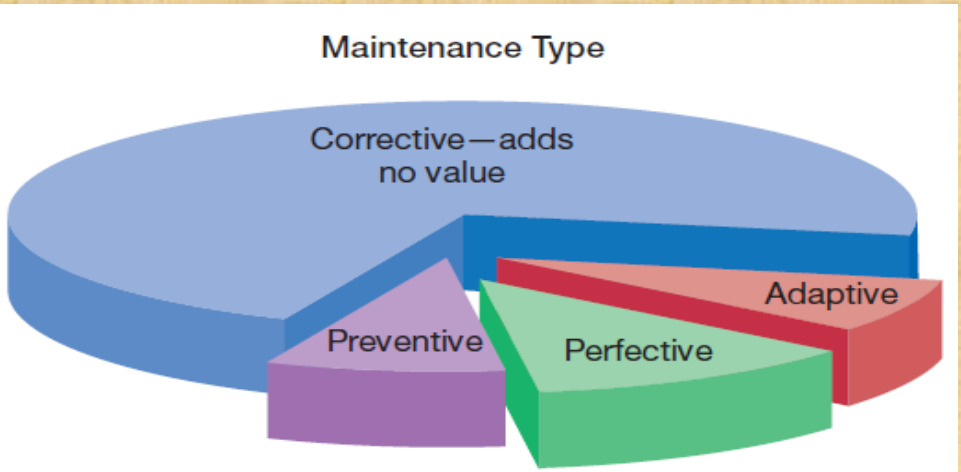

FIGURE 14-4
Value and non-value adding of different types of maintenance
(*Sources:* Based on Andrews and Leven-thal, 1993; Pressman, 2005.)

- **Perfective maintenance** involves making enhancements to improve processing performance or interface usability or to add desired, but not necessarily required, system features (bells and whistles). In our home example, perfective maintenance **would be adding a new room**. Many systems professionals feel that perfective maintenance is not really maintenance but rather new development.
- **Preventive maintenance** involves **changes made to a system to reduce the chance of future system failure**. An example of preventive maintenance might be to increase the number of records that a system can process far beyond what is currently needed or to generalize how a system sends report information to a printer so that the system can easily adapt to changes in printer technology. In our home example, **preventive** maintenance could be **painting the exterior to better protect the home from severe weather conditions**. As with adaptive maintenance, both perfective and preventive maintenance are typically a much lower priority than corrective maintenance. Over the life of a system, corrective maintenance is most likely to occur after initial system installation or after major system changes. This means that adaptive, perfective, and preventive maintenance activities can lead to corrective maintenance activities if not carefully designed and implemented.

- **The Cost of Maintenance**
- Information systems maintenance costs are a significant expenditure. For some organizations, as much as **60 to 80** percent of their information systems budget is allocated to maintenance activities (Kaplan, 2002). These huge maintenance costs are due to the fact that many organizations have accumulated more and more older so-called legacy systems that require more and more maintenance (see Figure 14-5). More maintenance means more maintenance work for programmers.

  For systems developed in-house, on average, 52 percent of a company's programmers are assigned to maintain **existing software** (Lytton, 2001). In situations where a company has not developed its systems in-house but instead has licensed software, as in the case of ERP systems, maintenance costs remain high. The standard cost of maintenance for most ERP vendors is 22 percent annually (Nash, 2010).

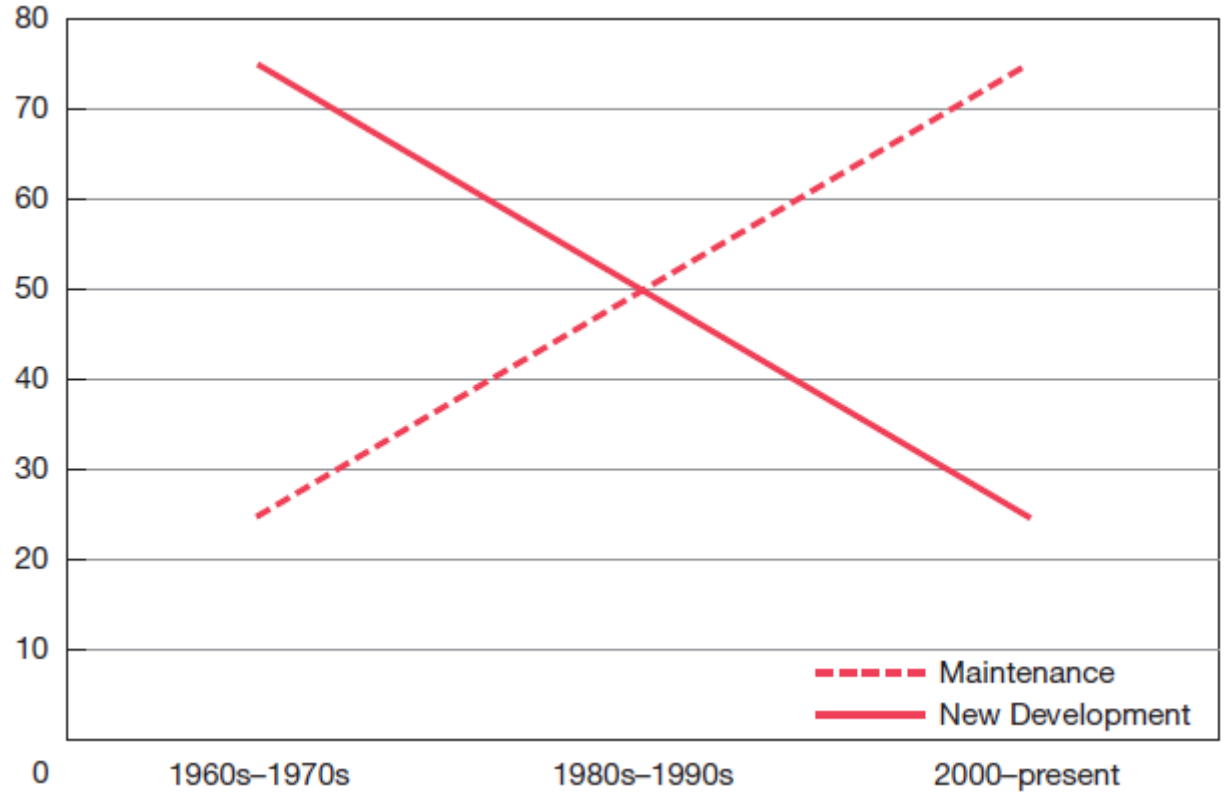

FIGURE 14-5

New development versus maintenance as a percentage of the software budget over the years

(*Source:* Based on Pressman, 2005.)

- **Managing Maintenance**

  As maintenance activities consume more and more of the systems development budget, maintenance management has become increasingly important. Today, far more programmers worldwide are working on maintenance than on new development. In other words, maintenance is the largest segment of programming personnel, and this implies the need for careful management.

- **Managing Maintenance Personnel:** One concern with managing maintenance relates to personnel management. Historically, many organizations had a "maintenance group" that was separate from the "development group." With the increased number of maintenance personnel, the development of formal methodologies and tools, changing organizational forms, end-user computing, and the widespread use of very high-level languages for the development of some systems, organizations have rethought the organization of maintenance and development personnel.

TABLE 14-2 Advantages and Disadvantages of Different Maintenance Organizational Structures

Type	Advantages	Disadvantages
Separate	Formal transfer of systems between groups improves the system and documentation quality	All things cannot be documented, so the maintenance group may not know critical information about the system
Combined	Maintenance group knows or has access to all assumptions and decisions behind the system's original design	Documentation and testing thoroughness may suffer due to a lack of a formal transfer of responsibility
Functional	Personnel have a vested interest in effectively maintaining the system and have a better understanding of functional requirements	Personnel may have limited job mobility and lack access to adequate human and technical resources

- **Measuring Maintenance Effectiveness :** A **second management issue** is the measurement of maintenance effectiveness. As with the effective management of personnel, the measurement of maintenance activities is fundamental to understanding the quality of the development and maintenance efforts. To measure effectiveness, you must measure the following factors:

- Number of failures
- Time between each failure
- Type of failure
- Measuring the number of and time between failures will provide you with the basis to calculate a widely used measure of system quality. This metric is referred to as the **mean time between failures (MTBF).** As its name implies, the MTBF metric shows the average length of time between the identification of one system failure and the next. Over time, you should expect the MTBF value to rapidly increase after a few months of use (and corrective maintenance) of the system (see Figure 14-7 for an example of the relationship between MTBF and age of a system). If the MTBF does not rapidly increase over time, it will be a signal to management that major problems exist within the system that are not being adequately resolved through the maintenance process.

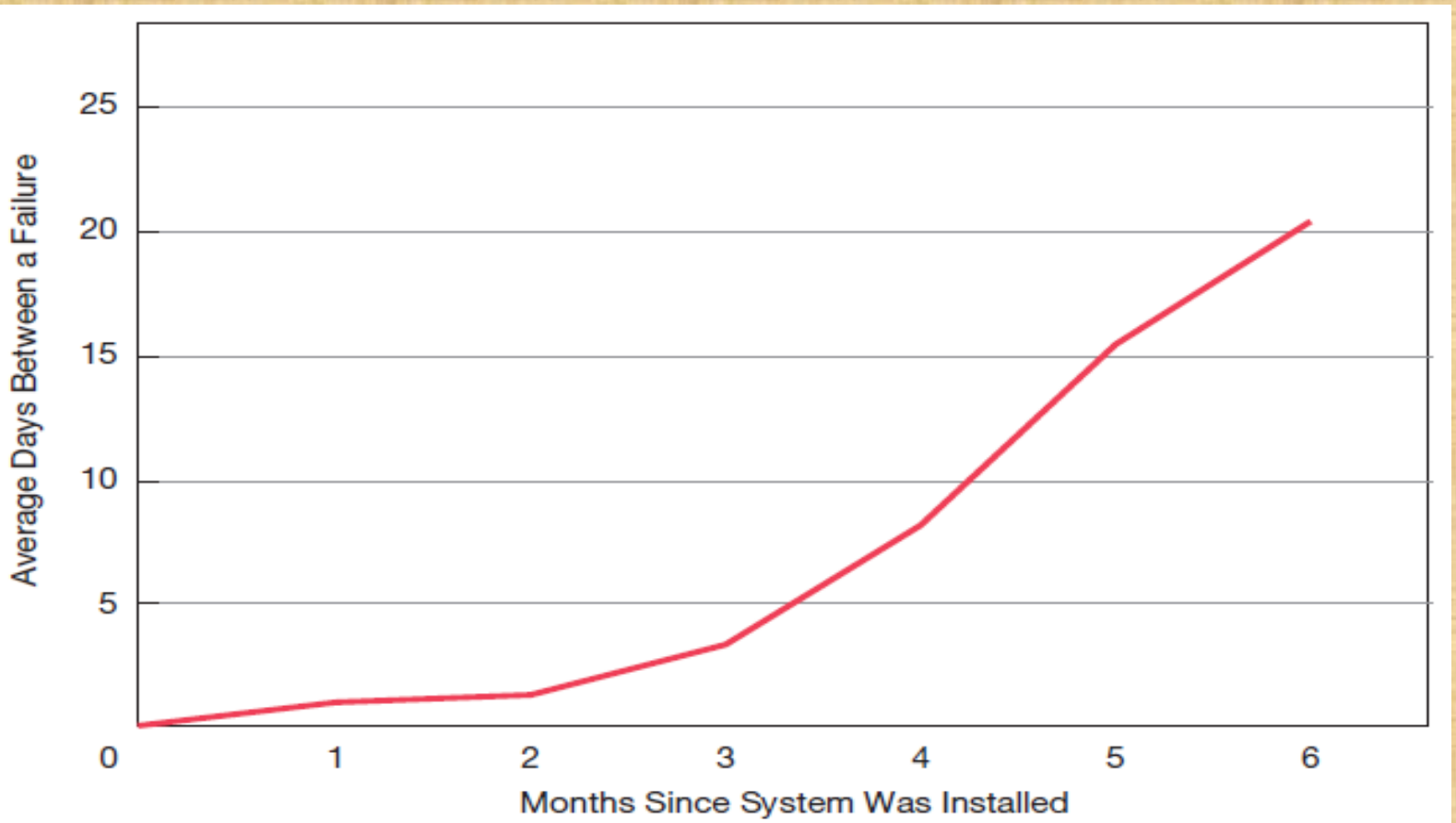

FIGURE 14-7
How the mean time between failures should change over time

- **Controlling Maintenance Requests (494)**
- Another maintenance activity is managing maintenance requests. There are various types of maintenance requests—some correct minor or severe defects in the systems, whereas others improve or extend system functionality. From a management perspective, a key issue is deciding which requests to perform and which to ignore. Because some requests will be more critical than others, some method of prioritizing requests must be determined. **Figure 14-8 shows a flowchart** that suggests one possible method you could apply for dealing with maintenance change requests. First, you must determine the type of request. If, for example, the request is an error—that is, a corrective maintenance request—then the flowchart shows that the request is placed in the queue of tasks waiting to be performed on the system. For an error of high severity, repairs to remove it must be made as soon as possible. If, however, the error is considered "**nonsevere**," then the change request can be categorized and prioritized based upon its type and relative importance.

- If the change request is not an error, then you must determine whether the request is to adapt the system to technology changes and/or business requirements, perfect its operation in some way, or enhance the system so that it will provide new business functionality. Enhancement-type requests must first be evaluated to see whether they are aligned with future business and information systems' plans. If not, the request will be rejected and the requester will be informed. If the enhancement appears to be aligned with business and information systems plans, it can then be prioritized and placed into the queue of future tasks. Part of the prioritization process includes estimating the scope and feasibility of the change.

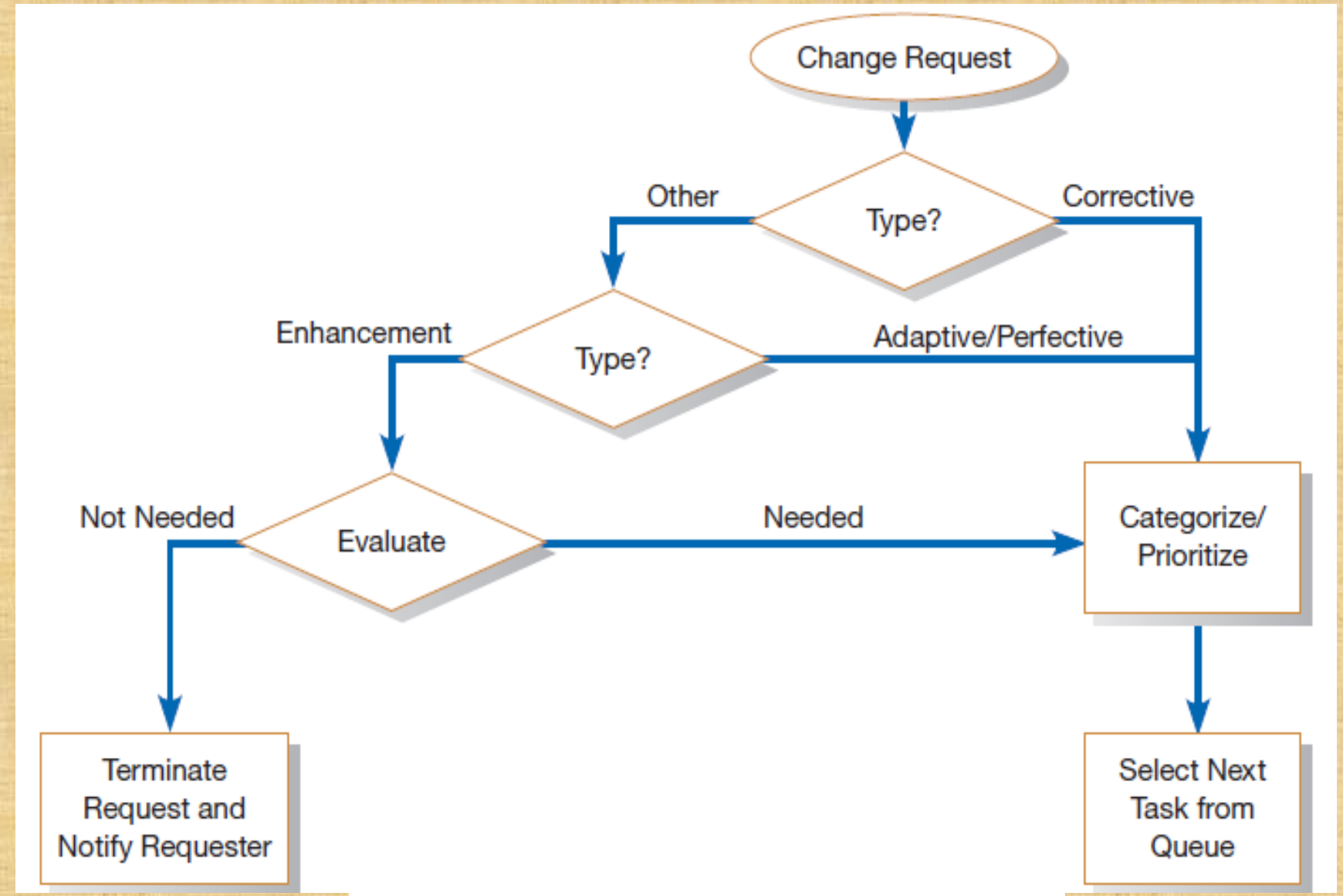

FIGURE 14-8
How to prioritize maintenance requests

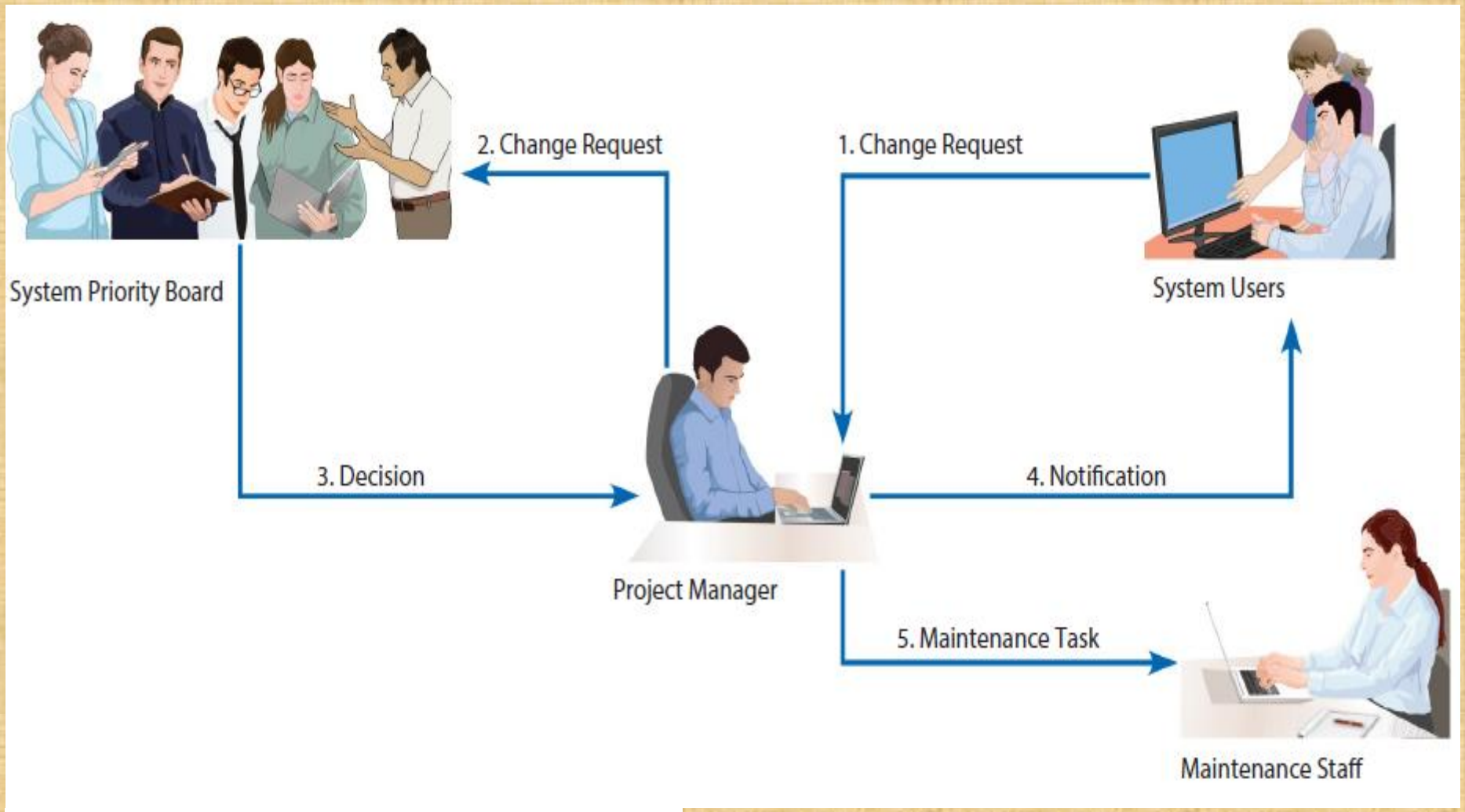

FIGURE 14-9
How a maintenance request moves through an organization

- **Configuration Management**
- A final aspect of managing maintenance is configuration management**,** which is the process of ensuring that **only authorized changes are made to a system**. Once a system has been implemented and installed, the programming code used to construct the system represents the **baseline modules of the** system. The baseline modules are the software modules for the most recent version of a system whereby each module has passed the organization's quality assurance process and documentation standards. A **system librarian controls the checking out** and checking in of the baseline source code modules. If maintenance personnel are assigned to make changes to a system, they must first check out a copy of the baseline system **modules—no one is allowed to directly modify the baseline modules**. Only those modules that have been tested and have gone through a formal check-in process can reside in the library. Before any code can be checked back in to the librarian, the code must pass the quality control procedures, testing, and documentation standards established by the organization.