Chapter 5

IMPLEMENTATION AND MAINTENANCE

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- System Implementation
- System Maintemtnce

INTRODUCTION

System Implementation is the process of converting the physical system specification into working and reliable software and hardware document the work that has been done and provide help for current an 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.

Although testing is done during implementation, we must begin planning for testing earlier in the project.

Planning involves determining what needs to be tested and collecting test data. This is often done during the analysis phase because testing requirements are related to system requirements.

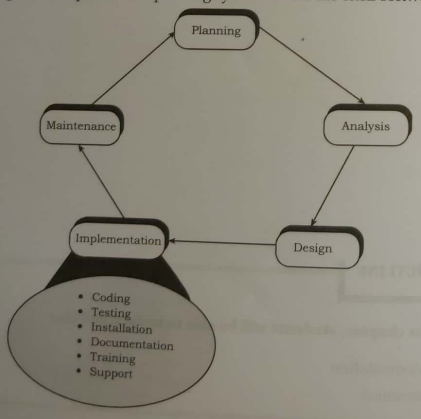
Installation is the process during which the current system is replaced by the new system. This includes conversion of existing data, software, documentation, and work procedures to those consistent with the new system. Users must give up the old way of doing their jobs, whether manual or automated, and adjusted to accomplishing the same task with the new system.

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 tom the new projects, analyst need to prepare documents that will reveal all of the important information that are accumulated about this system during its development and its implementation.

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 it's design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases. Software testing is a 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.

APPLICATIONS OF SOFTWARE TESTING

- Cost Effective Development - Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.
- Product Improvement - During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.
- Test Automation - Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.
- Quality Check - Software testing helps in determining following set of properties of any software such as
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability

DIFFEREST TYPES OF TESTS

1. **Inspections:** A testing technique in which participants examine program code for predictable language specific errors. Syntax, grammar, and some other routine errors can be checked by automated inspection software. 60 to 90 percent of all software effects as well as provide programmers with feedback that enables them to avoid making the same types of errors in future work.
2. **Desk checking:** Desk checking is an informal manual test that programmers can use to verify coding and algorithm logic before a program launch. This enables them to spot errors that might prevent a program from working as it should. Modern debugging tools

make desk checking less essential than it was in the past, but it can still be a useful way of spotting logic errors.

It refers to the manual approach of reviewing source code (sitting a desk), rather than running it through a debugger or another automated process. In some cases, a programmer may even use a pencil and paper to record the process and output of functions within a program. For example, the developer may track the value of one or more variables in a function from beginning to end. Manually going through the code line-by-line may help a programmer catch improper logic or inefficiencies that a software debugger would not.

While desk checking is useful for uncovering logic errors and other issues within a program's source code, it is time-consuming and subject to human error. Therefore, an IDE or debugging tool is better suited for detecting small problems, such as syntax errors. It is also helpful to have more than one developer desk check a program to reduce the likelihood of overlooking errors in the source code.

A testing technique in which the programmer or someone else who understands the logic of the program works through the code with a paper and pencil. The reviewer acts as the computer, mentally checking each step and its results for the entire set of computer instructions.

3. **Unit testing:** It focuses on smallest unit of software design. In this we test an individual unit or group of inter related units. It is often done by programmer by using sample input and observing its corresponding outputs. Automated technique whereby each module is tested alone in an attempt to discover any errors that may exist in the module's code.

4. **Integration testing:** The process of bringing together more than one modules that a program comprises for testing purposes. Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

5. **System testing:** The process of bringing together of all of the programs that a system comprises for testing purposes. System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. Programs are typically integrated in a top-down incremental fashion. The system can be tested in two ways:

   i. **Black box testing**: Black box testing 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.

   ii. **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 called white box testing because the test cases are totally visible to the general users and they can also make test cases.

S.N.	Black Box Testing	White Box Testing
1	Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
3	Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.
4	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
5	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
6	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.
7	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.
8	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.
9	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.
10	The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.

6. **Stub testing:** A technique used in testing modules, especially where modules are written and tested in a top down fashion, where a few lines of codes are used to substitute for subordinate modules. Top-level modules contain many call to subordinate modules, we may wonder how they can be tested if the lower level modules haven't been written yet. This is called stub testing.

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.

   i. **Alpha testing:** User testing of a completed information system using simulated data. The types of tests performed during alpha testing include the following:

      a. **Recovery testing:** Forces the software to fail in order to verify that recovery is properly performed.

      b. **Security testing:** Verifies that protection mechanisms built into the system will protect it from improper penetration.

      c. **Stress testing:** Tries to break the system (eg: what happens when a record is written to the database with incomplete information or what happens under extreme online transaction loads or with a large number of concurrent users).

      d. **Performance testing:** Determines how the system performs in the range of possible environments in which it may be used (eg: different hardware configurations, networks, operating systems).

   iii. **Beta testing:** User testing of a completed information system using real data in the real user environment. The intent of the beta test is to determine whether the software, documentation, technical support and training activities work as intended. Beta testing can be viewed as a rehearsal of the installation phase.

INSTALLATION

The process of moving from the current information system to the new one is called installation. In this activity, all the users must give up their reliance on the current system and begin to rely on new system. There are different ways and an 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 factor. Different ways

of installation are:

1. **Direct installation**: Changing over from the old information system to a new one by turning off the old system when the new one is turned on. Any errors resulting from the new system will have a direct impact on the users. 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. Direct installation can be very risky. 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. It is the least expensive installation method, and it creates considerable interest in making the installation a success.

2. **Parallel installation:** Running the old information system and the new one at the same time until management decides the old system can be turned off. All of the work done by the old system is concurrently performed by the new system. Outputs are compared 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 two staffs to operate and maintain. A parallel approach can also be confusing to users because they must deal with both systems. A parallel approach may not be feasible, especially if the users of the system cannot tolerate redundant effort or if the size of the system is large.

3. **Pilot installation:** It is also known as single-location installation. Rather than converting all of the organization at once, single location installation involves changing form the current to the new system in only one place or in a series of separate sites over time. 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. Problems with the system 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 system staff to support two versions of the system. Problems are isolated at one site at a time. If staff members can devote all of their efforts to success at the pilot site. If different locations require sharing of data, extra programs will need to be written to synchronize the current and new systems.

4. **Phased installation:** It is also called staged installation. Different parts of the old and new systems are used in cooperation until the whole new system is installed. By converting gradually, the organization's risk is spread out over time and place. Also phased installation allows for some benefits from the new system before the whole system is ready. For example, a 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 that pieces of the old system cannot be incrementally replaced so this strategy is not feasible. 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.

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.

   i. **Alpha testing:** User testing of a completed information system using simulated data. The types of tests performed during alpha testing include the following:

      a. **Recovery testing:** Forces the software to fail in order to verify that recovery is properly performed.

      b. **Security testing:** Verifies that protection mechanisms built into the system will protect it from improper penetration.

      c. **Stress testing:** Tries to break the system (eg: what happens when a record is written to the database with incomplete information or what happens under extreme online transaction loads or with a large number of concurrent users).

      d. **Performance testing:** Determines how the system performs in the range of possible environments in which it may be used (eg: different hardware configurations, networks, operating systems).

   iii. **Beta testing:** User testing of a completed information system using real data in the real user environment. The intent of the beta test is to determine whether the software, documentation, technical support and training activities work as intended. Beta testing can be viewed as a rehearsal of the installation phase.

INSTALLATION

The process of moving from the current information system to the new one is called installation. In this activity, all the users must give up their reliance on the current system and begin to rely on new system. There are different ways and an 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 factor. Different ways

of installation are:

1. **Direct installation**: Changing over from the old information system to a new one by turning off the old system when the new one is turned on. Any errors resulting from the new system will have a direct impact on the users. 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. Direct installation can be very risky. Direct installation requires a complete installation of the whole system.

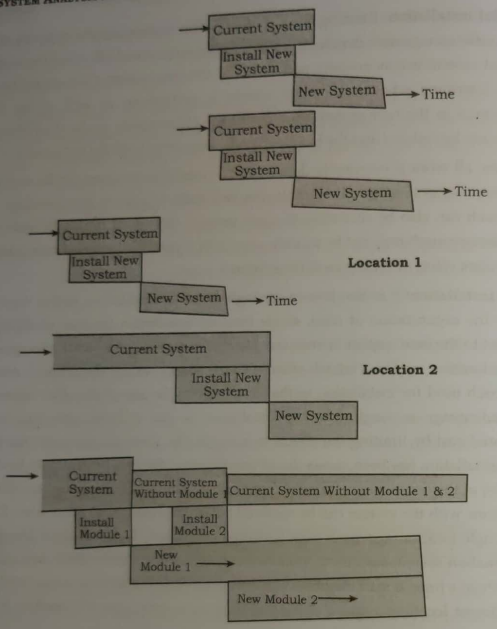
   For a large system, this may mean a long time until the new system can be installed, thus delaying system benefits or even missing the opportunities that motivated the system request. It is the least expensive installation method, and it creates considerable interest in making the installation a success.

2. **Parallel installation:** Running the old information system and the new one at the same time until management decides the old system can be turned off. All of the work done by the old system is concurrently performed by the new system. Outputs are compared 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 two staffs to operate and maintain. A parallel approach can also be confusing to users because they must deal with both systems. A parallel approach may not be feasible, especially if the users of the system cannot tolerate redundant effort or if the size of the system is large.

3. **Pilot installation:** It is also known as single-location installation. Rather than converting all of the organization at once, single location installation involves changing form the current to the new system in only one place or in a series of separate sites over time. 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. Problems with the system 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 system staff to support two versions of the system. Problems are isolated at one site at a time. If staff members can devote all of their efforts to success at the pilot site. If different locations require sharing of data, extra programs will need to be written to synchronize the current and new systems.

4. **Phased installation:** It is also called staged installation. Different parts of the old and new systems are used in cooperation until the whole new system is installed. By converting gradually, the organization's risk is spread out over time and place. Also phased installation allows for some benefits from the new system before the whole system is ready. For example, a 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 that pieces of the old system cannot be incrementally replaced so this strategy is not feasible. 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.



DOCUMENTING THE SYSTEM

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 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 requirement, and the form and type 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 program.
   - 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 techniques such as dataflow and 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
- 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 provide users access to information on new releases, bugs and tips for more effective 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.

SYSTEM MAINTENANCE

Once software system is put into use or installed, new requirements emerges and existing requirements change as the business running that system changes and 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 results obtained from the evaluation process help the organization to determine whether its information systems are effective and efficient or otherwise. The process of monitoring, evaluating, and modifying of existing information systems to make required or desirable improvements may be termed as System Maintenance. System maintenance is an ongoing activity, which covers a wide variety of activities, including removing program and design errors, updating documentation and test data and updating user support.

System maintenance is the general process of changing a system after it has been delivered. The changes may be Simple changes to correct coding errors, more extensive changes to correct design errors or significant enhancement to correct specification errors or accommodate new requirements.

There are four major activities occur within maintenance:

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

1. ***Obtaining Maintenance Requests:*** In this step a formal process be established whereby users can submit system change requests. 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.

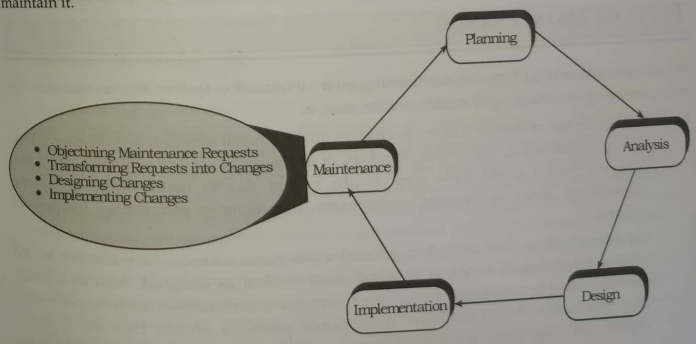
2. **Transforming Request into Changes:** Once a request is received, analysis must be performed to identify the of the scope of the request. It must be determined how the request will affect the current system and how long such a project will take. As with the initial development of a system, the size of a maintenance request can be analyzed for risk and feasibility.

3. **Designing changes:** Next, a change request can be transformed into a formal design change, which can then be fed into the maintenance implementation phase.

4. **Implementing Changes:** In this activity once the change design is approved, proposed changes are implemented in respective components of the system.

ANALOGY BETWEEN SDLC AND SYSTEM MAINTENANCE

Many similarities exist between the SDLC and the activities within the maintenance process. The first phase of the SDLC—planning—is analogous to the maintenance process of obtaining a maintenance request. The SDLC analysis phase is analogous to the maintenance process of transforming requests into a specific system change. 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.

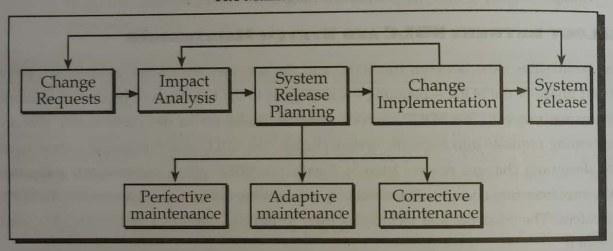


CONDUCTING MAINTENANCE

A significant portion of the expenditures for information systems 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.

Maintenance processes vary considerably depending on the type of system being maintained, the development process used in the organization and the people involved in the process. However, the maintenance process is triggered by a set of change requests from system users, management or customers. The cost and impact of these changes are assed to see how much f the system is affected by the change and how much it might costs to implement the change. If the proposed changes are accepted, new release of the system is planned. During release planning, all the proposed changes are considered. A decision is then made on which changes to implement in the next version of the system. The changes are implemented and validated.

The Maintenance Process



TYPES OF MAINTENANCE

We can perform several types of **maintenance** on an information system. By maintenance, we mean the fixing or enhancing of an information system.

1. **Corrective maintenance:** It refers to changes made to repair defects in the design, coding, or implementation of the system. This type of maintenance implies removing errors in a program, which might have crept in the system due to faulty design or wrong assumptions. Thus, in corrective maintenance, processing or performance failures are repaired.

   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.

2. **Adaptive maintenance:** In adaptive maintenance, program functions are changed to enable the information system to satisfy the information needs of the user. It 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.

   This type of maintenance may become necessary because of organizational changes which may include:

   a. Change in the organizational procedures,
   b. Change in organizational objectives, goals, policies, etc.
   c. Change in forms,
   d. Change in information needs of managers.
   e. Change in system controls and security needs, etc.

3. **Perfective maintenance:** Perfective maintenance means adding new programs or modifying the existing programs to enhance the performance of the information system. This type of maintenance undertaken to respond to user's additional needs which may be due to the changes within or outside of the organization. Outside changes are primarily environmental changes, which may in the absence of system maintenance, render the information system ineffective and inefficient. These environmental changes include:

   a) Changes in governmental policies, laws, etc.,
   b) Economic and competitive conditions, and
   c) New technology.

   It 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.

4. **Preventive maintenance:** Preventive changes refer to changes made to increase the understanding and maintainability of your software in the long run. Preventive changes are focused in decreasing the deterioration of your software in the long run. Restructuring, optimizing code and updating documentation are common preventive changes. Executing preventive changes reduces the amount of unpredictable effects software can have in the long term and helps it become scalable, stable, understandable and maintainable. It **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.

The Cost of Maintenance

The cost of maintenance represent a large proportion of the budget of most organization that use the software system. For some organizations, as much as 60 to 80 percent of their information systems budget is allocated to maintenance activities. These huge maintenance costs are due to the fact that many organizations have accumulated more and more older legacy systems that require more and more maintenance. More maintenance means more maintenance work for programmers. In addition, about one-third of the costs of establishing and keeping a presence on the Web go to programming maintenance.

Maintenance cost as a proportion cost of development costs vary from one application domain to another. For business application system, maintenance cost were broadly comparable with system development cost. For embedded real time systems, maintenance cost may be up to four times higher than development cost.

So while developing system, good software engineering techniques such as precise specification, use of object oriented development and configuration management are used that contribute to maintenance cost reduction.

Factors Influencing Maintenance Cost

Numerous factors influence the **maintainability** of a system. These factors, or cost elements, determine the extent to which a system has high or low maintain- ability. Of these factors, three are most significant: the number of latent defects, the number of customers, and documentation quality. The others—personnel, tools, and software structure—have noticeable, but less, influence.

- **Latent defects:** This is the number of unknown errors existing in the system after it is installed. Because corrective maintenance accounts for most maintenance activity, the number of latent defects in a system influences most of the costs as- sociated with maintaining a system.
- **Number of customers for a given system:** In general, the greater the number of customers, the greater the maintenance costs. For example, if a system has only one customer, problem and change requests will come from only one source. Also, training, error reporting, and support will be simpler. Maintenance requests are less likely to be contradictory or incompatible.
- **Quality of system documentation:** Without quality documentation, maintenance efforts can increase exponentially. High-quality documentation leads reduction in the sys- tem maintenance effort when compared with average-quality documentation. In other words, quality documentation makes it easier to find code that needs to be changed and to understand how the code needs to be changed. Good documentation also explains why a system does what it does and why alternatives were not feasible, which saves wasted maintenance efforts.

- **Maintenance personnel:** In some organizations, the best programmers are assigned to maintenance. Highly skilled programmers are needed because the maintenance programmer is typically not the original programmer and must quickly understand and carefully change the software.
- **Tools:** Tools that can automatically produce system documentation where none exists can also lower maintenance costs. Also, tools that can automatically gen- erate new code based on system specification changes can dramatically reduce maintenance time and costs.
- **Well-structured programs:** Well-designed system are easier to understand and fix.

Managing Maintenance

As maintenance activities consume more and more of the systems development bud- get, 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. We will address this concern by dis- cussing several topics related to the effective management of systems maintenance.

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. In other words, should the maintenance group be separated from the development group? Or should the same people who build the system also maintain it? A third option is to let the primary end users of the system in the functional units of the business have their own maintenance personnel.

Measuring Maintenance Effectiveness

A second management issue is the mea- surement of maintenance effectiveness. As with the effective management of person- nel, 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.

Tracking the types of failures also provides important management information for future projects. For example, if a higher frequency of errors occurs when a partic- ular development environment is used, such information can help guide personnel assignments; training courses; or the avoidance of a particular package, language, or environment during future development. The primary lesson here is that without measuring and tracking maintenance activities, you cannot gain the knowledge to improve or know how well you are doing relative to the past. To effectively manage and to continuously improve, you must measure and assess performance over time.

Controlling Maintenance Requests

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.

Configuration Management An aspect of managing maintenance is **configuration management**, which is the process of ensuring that only authorized changes are made to a system. Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life. A configuration management is used to keep track of an organization's hardware, software and related information. This includes software versions and updates installed on the organization's computer systems. CM also involves logging the network addresses belonging to the hardware devices used. Software is available for all of these tracking tasks. 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 pro- cess can reside in the library. Before any code can be checked back in to the librar- ian, the code must pass the quality control procedures, testing, and documentation standards established by the organization.

When various maintenance personnel working on different maintenance tasks complete each task, the librarian notifies those still working that updates have been made to the baseline modules. This means that all tasks being worked on must now incorporate the latest baseline modules before being approved for check-in. Following a formal process of checking modules out and in, a system librarian helps to ensure that only tested and approved modules become part of the baseline System. It is also the responsibility of the librarian to keep copies of all prior versions of all system modules, including the **build routines** needed to construct *any version* of

the system that has *ever* existed. It may be important to reconstruct old versions of the system if new ones fail or to support users that cannot run newer versions on their computer system.

Special software systems have been created to manage system configuration and version control activities (see the box "Configuration Management Tools"). This software is becoming increasingly necessary as the change control process becomes ever more complicated in organizations deploying several different networks, oper-ating systems, languages, and database management systems in which there may be many concurrent versions of an application, each for a different platform. One func-tion of this software is to control access to modules in the system library. Each time a module is checked out or in, this activity is recorded, after being authorized.

Role of CASE in Maintenance

In traditional systems development, much of the time is spent on coding and testing. When software changes are approved, code is first changed and then tested. Once the functionality of the code is assured, the documentation and specification documents are updated to reflect system changes. Over time, the process of keeping all system documentation "current" can be a very boring and time-consuming activity that is often neglected. This neglect makes future maintenance by the same or *differ- ent* programmers difficult at best.

A primary objective of using automated tools for systems development and maintenance is to radically change the way in which code and documentation are modified and updated. When using an integrated development environment, ana- lysts maintain design documents such as data flow diagrams and screen designs, not source code. In other words, design documents are modified and then code generators automatically create a new version of the system from these updated designs. Also, because the changes are made at the design specification level, most documentation changes, such as an updated data flow diagram, will have already been completed during the maintenance process itself. Thus, one of the biggest advantages to using automated tools, for example, is its usefulness in system maintenance.

In addition to using general automated tools for maintenance, two special purpose tools, reverse engineering and reengineering tools, are used to maintain older systems that have incomplete documentation. These tools are often referred to as *design recovery* tools because their primary benefit is to create high-level design documents of a program by reading and analyzing its source code.

Reverse engineering tools are those that can create a representation of a system or program module at a design level of abstraction. For example, reverse engineering tools read program source code as input; perform an analysis; and extract information such as program control structures, data structures, and data flow. Once a program is represented at a design level using both graphical and textual representations, the design can be more effectively restructured to current business needs or programming practices by an analyst. For example, Microsoft's Visual Studio.NET can be used to reverse engineer applications into UML or other development diagrams.

Similarly, **reengineering** tools extend reverse engineering tools by automatically (
interactively with a systems analyst) altering an existing system in an effort to improve
quality or performance. As reverse and reengineering capabilities are included in more popul
development environments, the ability to extend the life and evolve the capabilities of existi
systems will be enhanced.

MULTIPLE CHOICE QUESTIONS

d. Inclusiveness

EXERCISE

1. What are the deliverables from coding, testing, and installation?
2. Explain the code-testing process.
3. What are structured walk-throughs for code? What is their purpose? How are they conducted? How are they different from code inspections?
4. What are the four approaches to installation? Which is the most expensive? Which is the most risky? How does an organization decide which approach to use?
5. What is the conventional wisdom about implementation success?
6. List and define the factors that are important to successful implementation efforts.
7. Explain Lucas's model of implementation success.
8. What is the difference between system documentation and user documentation?
9. What proof do you have that individual differences matter in computer training?
10. What types of security policies and procedures does your university have in place for campus information systems?
11. List the steps in the maintenance process and contrast them with the phases of the SDLC.
12. What are the different types of maintenance and how do they differ?
13. Describe the factors that influence the cost of maintenance.
14. Are any factors more important than others? Why?

15. Describe three ways for organizing maintenance personnel and contrast the advantages and disadvantages of each approach.
16. What types of measurements must be taken to gain an understanding of the effectiveness of maintenance? Why is tracking mean time between failures an important measurement?
17. What managerial issues can be better understood by measuring maintenance effectiveness?
18. Describe the process for controlling maintenance requests. Should all requests be handled in the same way or are there situations when you should be able to circumvent the process? If so, when and why?
19. What is meant by configuration management? Why do you think organizations have adopted the approach of using a systems librarian?
20. How are automated tools used in the maintenance of information systems? What is the difference between reverse engineering and reengineering tools?

❑❑❑