Unit 5 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:

Implementation is the process of building properly working system, install it in the organization, replacing old system and working methods, and finalize system and user documentation and training end prepared to support the system to assists users as they encounter difficulties. Because implementation is such an important phase of system development, all the activities that are to be done during this phase are to be planned.

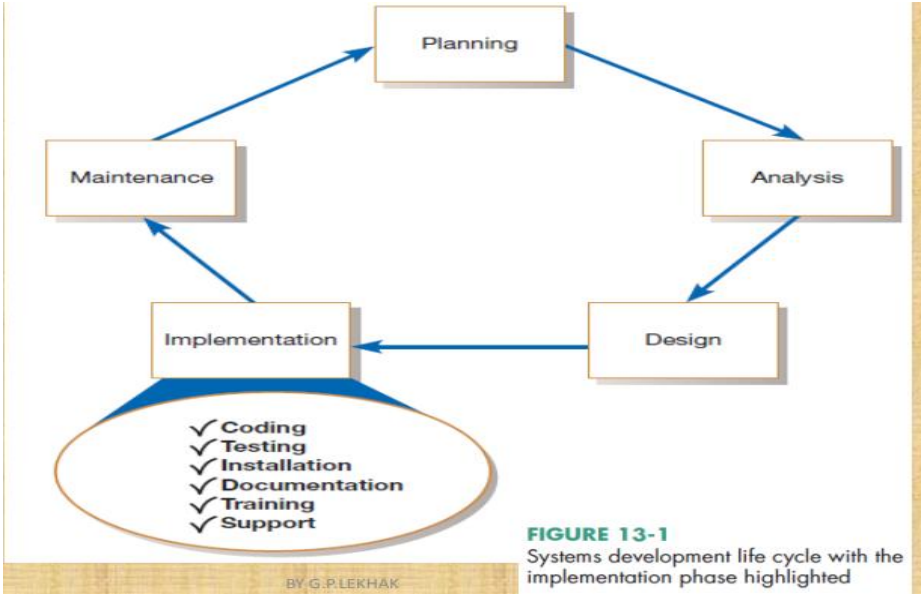


FIGURE 13-1
Systems development life cycle with the implementation phase highlighted

System implementation is made up of many activities. The **six** major activities are:

1. Coding
2. Testing
3. Installation
4. Documentation
5. Training
6. Support

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.

The Process of Coding, Testing, and Installation

Coding:

Coding is the process where, physical design specifications created by the analysis team are turned into working computer code by program team. It means actually writing code or monitoring coding done by programmers to ensure that programs meet design specifications.

Coding Deliverables: Code, Program Documentation

Testing:

Testing is the process of finding and fixing bugs and errors. Tests are performed using various strategies. Testing performed in parallel with coding. It involves using test data and scenarios to verify that each component and the whole system work under normal and abnormal circumstances.

Testing Deliverables: Test Scenarios (test plan) and test data, Results of programs and system testing

Installation:

It is the process during which the current system is replaced by new system. It includes installing the new system in organizational sites as well as dealing with personal and organizational resistance to the change that the new system causes.

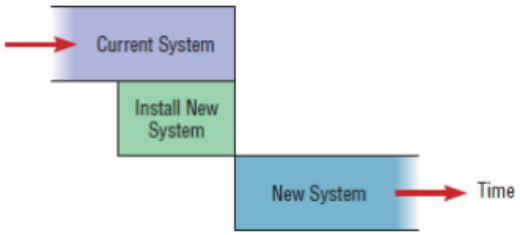
Installation Deliverables: User guides, User training plan, Installation and conversion plan

Types of Installation: Four installation strategies:

- Direct Installation
- Parallel Installation
- Single-location installation
- Phased Installation

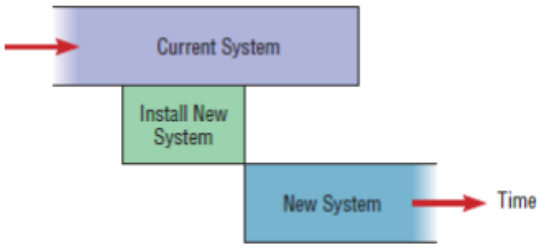
Direct Installation

Changing over from the old information system to a new one by turning off the old system when the new one is turned on



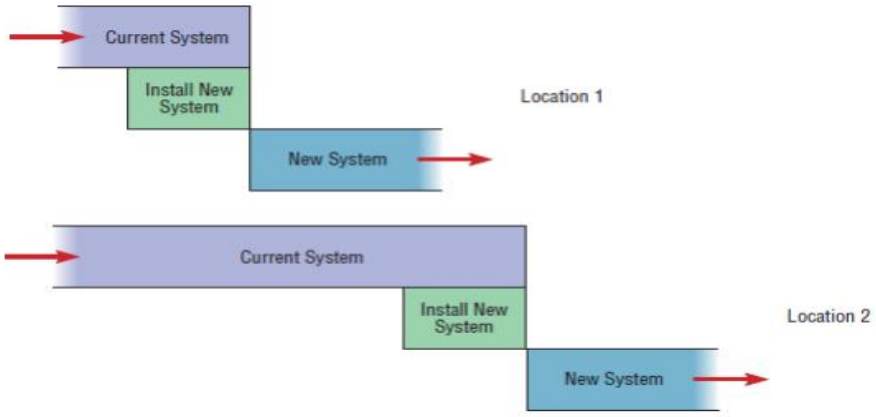
Parallel Installation

Running the old information system and the new one at the same time until management decides the old system can be turned off.



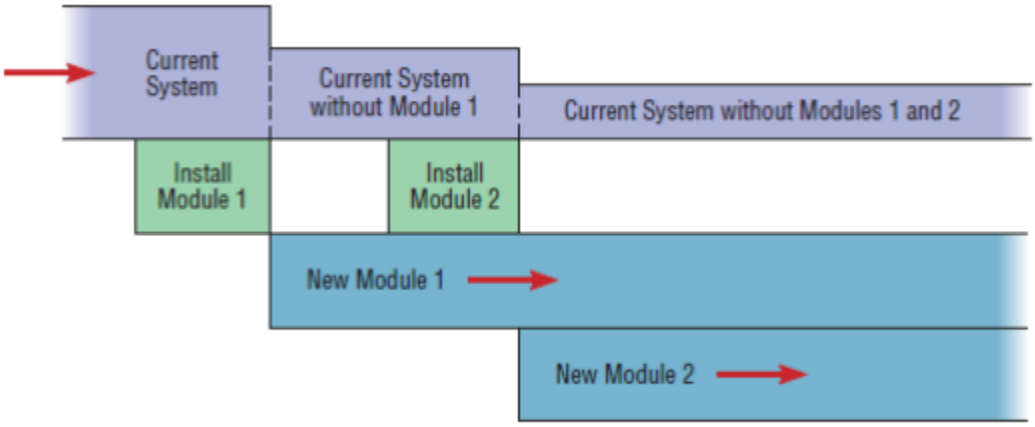
Single location installation

Trying out an information system at one site and using the experience to decide if and how the new system should be deployed throughout the organization



Phased Installation

Changing from the old information system to the new one incrementally, starting with one or a few functional components and then gradually extending the installation to cover the whole new system.



The Process of Documenting the System, Training Users, and Supporting Users

Documentation:

It is the process of converting software specification with the hard copy format it acts as references maintained for user. It includes reviewing all project dictionary or CASE repository entries for completeness as well as finalizing all user documentation, such as user guides, reference cards and tutorials.

Two audiences for final documentation:

• Information systems personnel (System Builders, internal users) who will maintain the system throughout its productive life.

• People who will use the system (System Users) as part of their daily lives.

Documentation Deliverables: System Documentation, User Documentation

A. System documentation

It is detailed information about a system's design specifications, its internal workings and its functionality. The intended audience: maintenance programmers.

System documentation is of **two** types:

- **Internal documentation** System documentation that is part of the program source code or is generated at compile time
- **External documentation** System documentation that includes the outcome of structured diagramming techniques such as data-flow and entity-relationship diagrams

B. User Documentation

Written or other visual information about an application system, how it works, and how to use it

User Training:

It is the process of providing necessary skills and knowledge to user. It may include a variety of human and computer-assisted sessions as well as tools to explain the purpose and use of the system.

The training can be:

- Application specific or
- General for operating system and off the shelf software.

User Training Deliverables: Classes, Tutorials, User training modules, training materials, and computer based training aids

User Support:

It is the process of building of mechanism which acts as guidelines for the user in case of any problem arises. It ensures that users can obtain the assistance they need as questions and problems arise. User Support Deliverables: Help desk, online help, Bulletin boards and other support mechanisms.

Software Application Testing

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. We should emphasize that although testing is done during implementation, you must begin planning for testing earlier in the project. Planning involves determining what needs to be tested and collecting test data. These activities are often done during the analysis phase, because testing requirements are related to system requirements.

Testing software begins earlier in the systems development life cycle, even though many of the actual testing activities are carried out during implementation. During analysis, you develop an overall test plan. During design, you develop a unit test plan, an integration test plan, and a system test plan. During implementation, these various plans are put into effect, and the actual testing is performed.

The purpose of these written test plans is to improve communication among all the people involved in testing the application software. The plan specifies what each person's role will be during testing. The test plans also serve as checklists you can use to determine whether all testing steps have been completed. The overall test plan is not just a single document but is a collection of documents. Each of the component documents represents a complete test plan for one part of the system or for a particular type of test.

Some organizations have specially trained personnel who supervise and support testing. Testing managers are responsible for developing test plans, establishing testing standards, integrating testing and development activities in the life cycle, and ensuring that test plans are completed. Testing specialists help develop test plans, create test cases and scenarios, execute the actual tests, and analyze and report test results.

Different Types of Tests

Inspection:

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 wellknown 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,

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.

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.

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.

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:**

- **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.

- **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.

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.

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.

The most complete acceptance testing will include **alpha testing**, (also called mock client testing), where simulated but typical data are used for system testing; **beta testing**, in which live data are used in the users' real working environment; and a system audit conducted by the organization's internal auditors or by members of the quality assurance group.

Alpha Testing

During alpha testing, the entire system is implemented in a test environment to discover whether the system is overtly destructive to itself or to the rest of the environment.

Beta Testing

In beta testing, a subset of the intended users runs the system in their own environments using their own data. The intent of the beta test is to determine whether the software, documentation, technical support, and training activities work as intended. In essence, beta testing can be viewed as a rehearsal of the installation phase.

System Maintenance

Maintenance is the changes made to a system to fix or enhance its functionality. Software maintenance is a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities, and optimization. The deliverables and outcomes from the process of maintenance are the development of a new version of the software and new versions of all design documents created or modified during the maintenance effort.

Types of Maintenance

There are four types of maintenance:

Corrective maintenance

Changes made to a system to repair flaws in its design, coding, or implementation

Adaptive maintenance

Changes made to a system to evolve its functionality to changing business needs or technologies

Perfective maintenance

Changes made to a system to add new features or to improve performance

Preventive maintenance

Changes made to a system to avoid possible future problems

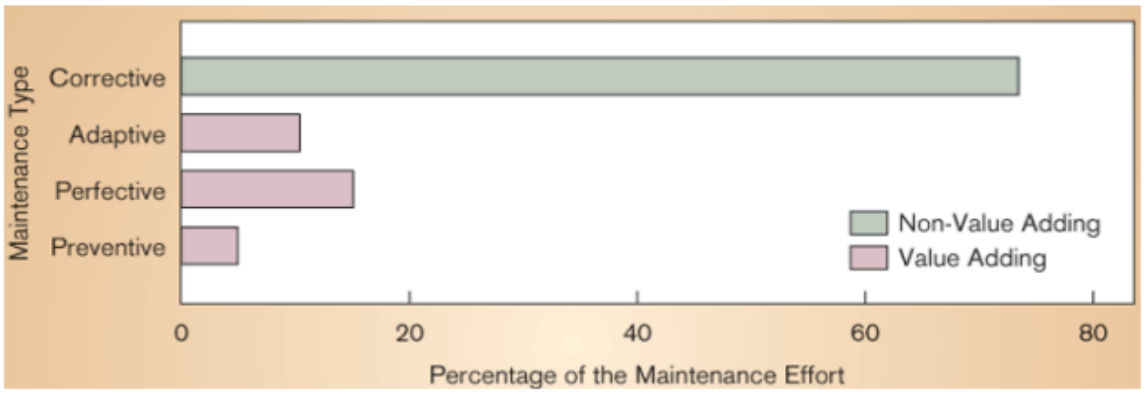


Figure: Maintenance Types and their Corresponding Efforts

By far, most maintenance is corrective, and therefore urgent and non-value adding.

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

It is the process of returning to the beginning of the SDLC and repeating development steps focusing on system change until the change is implemented. Maintenance is the longest phase in the SDLC. Maintenance is like a mini-SDLC.

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 the duration of such a project. 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.

Four major activities are involved in maintenance:

(1) Obtaining maintenance requests

(2) Transforming requests into changes

(3) Designing changes

(4) Implementing changes

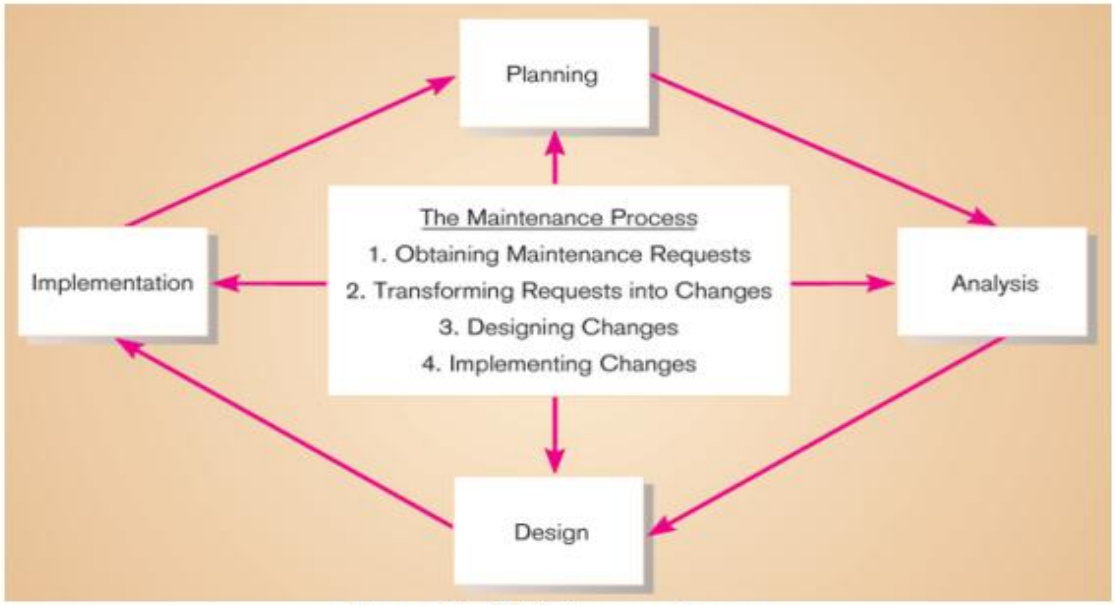


Figure: The Maintenance Process

The figure shows that the first phase of the SDLC - systems planning and selection - is analogous to the maintenance process of obtaining a maintenance request (step 1). The SDLC phase systems analysis is analogous to the maintenance process of transforming requests into a specific system change (step 2). The systems design phase of the SDLC, of course, equates to the designing changes process (step 3). Finally, the SDLC phase implementation and operation equates to implementing changes (step 4). This similarity between the maintenance process and the SDLC is no accident. The concepts and techniques used to develop a system initially are also used to maintain it.

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.

The Cost of Maintenance

Information systems maintenance costs are a significant expenditure. For some organizations, as much 60-80 percent of their information systems budget is allocated to maintenance activities. Maintainability is the ease with which software can be understood, corrected, adapted, and enhanced. Systems with low maintainability result in uncontrollable maintenance expenses.

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

The factors that influence system maintainability are:

- Latent defects
- Number of customers for a given system
- Quality of system documentation
- Maintenance personnel
- Tools
- Well-structured programs

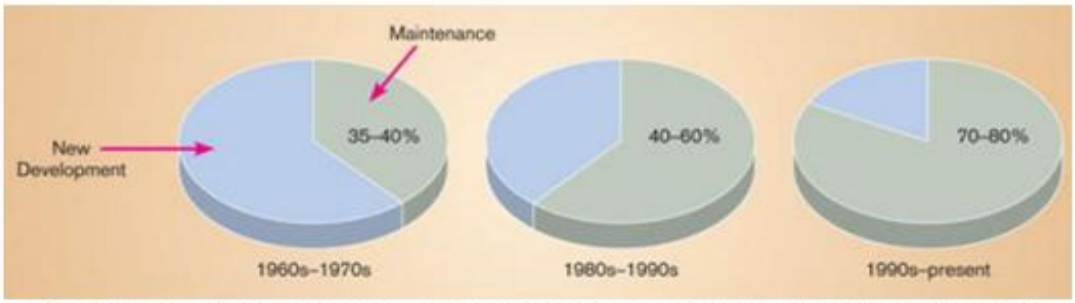
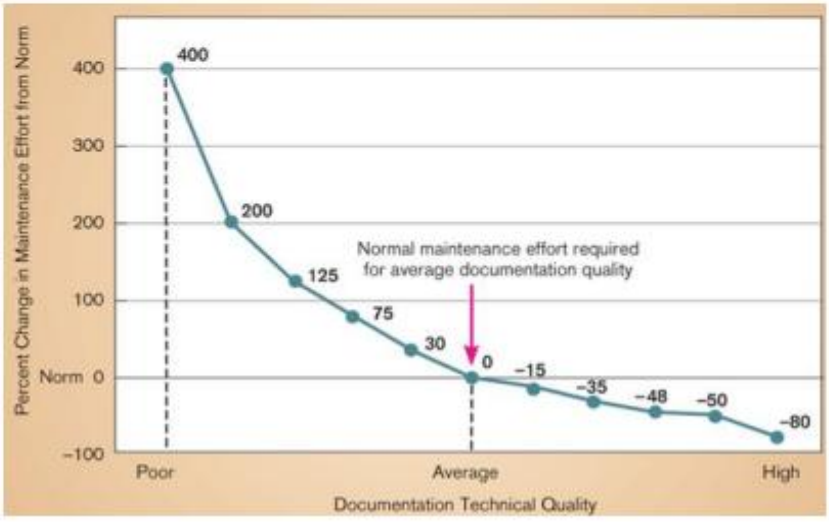


Fig: New development versus maintenance as a percent of software budget



A well-documented system is easier to understand, and therefore easier to maintain.

Managing Maintenance

Maintenance can be managed through four ways:

(a) Managing Maintenance Personnel

(b) Measuring Maintenance Effectiveness

(c) Controlling Maintenance Request

(d) Configuration Management

Managing Maintenance Personnel

Number of people working in maintenance has surpassed number working in development. Maintenance work is often viewed negatively by IS personnel. Organizations have historically have rewarded people involved in new development better than maintenance personnel. Organizations often rotate personnel in and out of maintenance roles in order to lessen negative feelings about maintenance. Three possible organizational structures exist:

- **Separate** Maintenance group consists of different personnel than development group.
- **Combined** Developers also maintain systems.
- **Functional** Maintenance personnel work within the functional business unit.

Measuring Maintenance Effectiveness

Because maintenance can be so costly, it is important to measure its effectiveness. To measure effectiveness, you must measure these 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 measure is referred to as the mean time between failures (MTBF).

As its name implies, the MTBF measure shows the average length of time between the identification of one system failure until the next. Over time, you should expect the MTBF value to increase rapidly after a few months of use (and corrective maintenance) of the 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.

Controlling Maintenance Requests

Another maintenance activity is managing maintenance requests. 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.

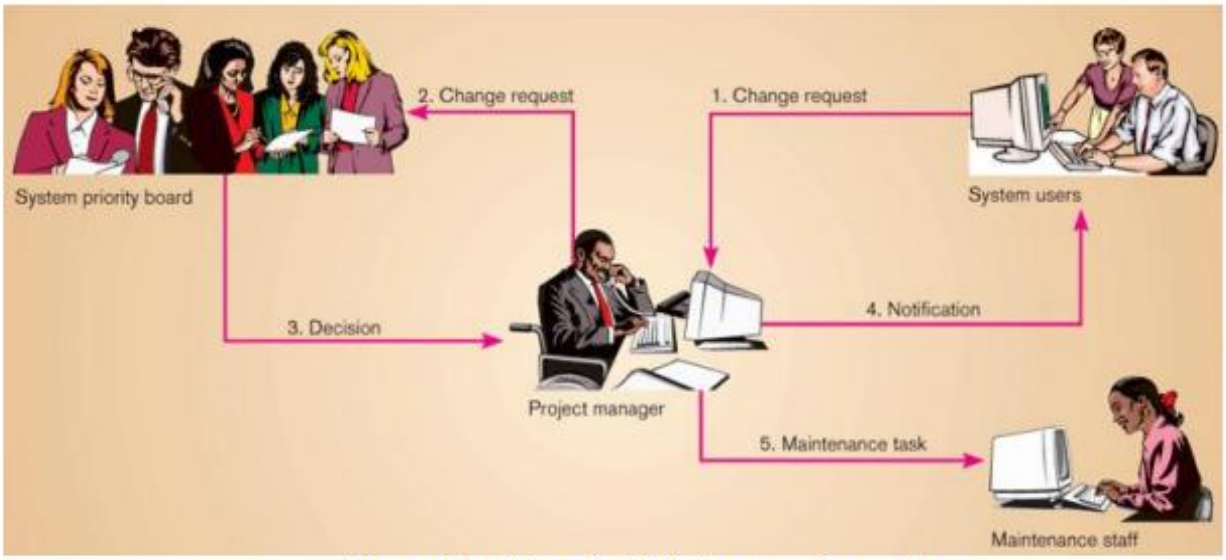


Figure: The Flow of a Maintenance Request

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.

Some of the configuration techniques are:

- **Baseline modules** Software modules that have been tested, documented, and approved to be included in the most recently created version of a system
- **System librarian** A person responsible for controlling the checking out and checking in of baseline modules when a system is being developed or maintained
- **Build routines** Guidelines that list the instructions to construct an executable system from the baseline source code.