UNIT-III

Testing fundamentals: Objectives, principles, testability, Test cases: White box & Black box testing strategies: verification & validation, UNIT test, integration testing, validation, testing, system testing, System Implementation, Maintenance and documentation, Document Configurations Maintaining a Configuration.

Unit - III

9

TESTING AND QUALITY ASSURANCE

Objectives

Software development is not a precise science, and humans being as error prone as they are, software development must be accompanied by quality assurance activities. It is typical for developers to spend around 40% of the total project time on testing. For life critical software (e.g. flight control, reactor monitoring), testing can cost 3 to 5 times as much as all other activities combined. The destructive

nature of testing requires that the developer discard preconceived notions of the correctness of his/her developed software. This means that testing must be done from an entirely different perspective that of a developer.

In software development project, errors can be come in at any stage during development. The main causes of errors are :

1. not obtaining the right requirements,
2. not getting the requirements right, and
3. not translating the requirements In a clear and understandable manner so that programmers implement them properly

There are techniques available for detecting and eliminating errors that originate in various stages. However, no technique is perfect.

1. Fundamentals of Software Testing :

Testing is basically a process to detect errors in the software product. Before going into the details of testing techniques one should know what errors are. In day-to-day life we say whenever something goes wrong there is an error. This definition is quite vast. When we apply this concept to software products then we say whenever there is difference between what is expected out of software and what is being achieved, there is an error. For the output of the system, if it differs from what was required, it is due to an error. This output can be some numeric or alphabetic value, some formatted report, or some specific behavior from the system. In case of an error there may be change in the format of out, some unexpected behavior from system, or some value different from the expected is obtained. These errors can due to wrong analysis, wrong design, or some fault on developer's part.

All these errors need to be discovered before the system is implemented at the customer's site. Because having a system that does not perform as desired be of no use. All the effort put in to build it goes waste. So testing is done. And it is equally important and crucial as any other stage of system development. For different types of errors there are different types of testing techniques. In the section that follows we'll try to understand those techniques.

Objectives of Testing : First of all the objective of the testing should be clear. We can define testing as a process of executing a program with the aim of finding errors. To perform testing, test cases are designed. A test case is a particular made up artificial situation upon which a program is exposed so as to find errors. So a good test case is one that finds undiscovered errors. If testing is done properly, it uncovers errors and after fixing those errors we have software that is being developed according to specifications.

Test Information Flow : Testing is a complete process. For testing we need two types of inputs. First is software configuration. It includes software requirement specification, design specifications and source code of program. Second is test configuration. It is basically test plan and procedure. Software configuration is

required so that the testers know what is to be expected and tested whereas test configuration is testing plan that is, the way how the testing will be conducted on the system. It specifies the test cases and their expected value. It also specifies if any tools for testing are to be used. Test cases are required to know what specific situations need to be tested. When tests are evaluated, test results are compared with actual results and if there is some error, then debugging is done to correct the error. Testing is a way to know about quality and reliability. Error rate that is the occurrence of errors is evaluated. This data can be used to predict the occurrence of errors in future.

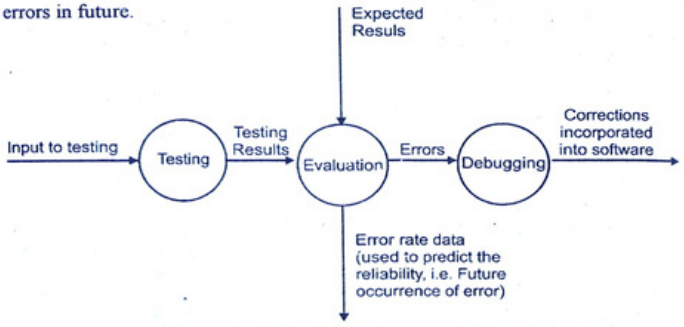

Fig. 1 : Testing process

Test Case Design : We now know, test cases are integral part of testing. So we need to know more about test cases and how these test cases are designed. The most desired or obvious expectation from a test case is that it should be able to find most errors with the least amount of time and effort.

A software product can be tested in two ways. In the first approach only the overall functioning of the product is tested. Inputs are given and outputs are checked. This approach is called black box testing. It does not care about the internal functioning of the product.

The other approach is called white box testing. Here the internal functioning of the product is tested. Each procedure is tested for its accuracy. It is more intensive than black box testing. But for the overall product both these techniques are crucial. There should be sufficient number of tests in both categories to test the overall product.

2. White Box Testing:

White box testing focuses on the internal functioning of the product. For this different procedures are tested. White box testing tests the following

- Loops of the procedure
- Decision points
- Execution paths

For performing white box testing, basic path testing technique is used. We will illustrate how to use this technique, in the following section.

2.1. Basis Path Testing :

Basic path testing a white box testing technique .It was proposed by Tom McCabe. These tests guarantee to execute every statement in the program at least one time during testing. Basic set is the set of all the execution path of a procedure.

Flow graph Notation : Before basic path procedure is discussed, it is important to know the simple notation used for the representation of control flow. This notation is known as flow graph. Flow graph depicts control flow and uses the following constructs.

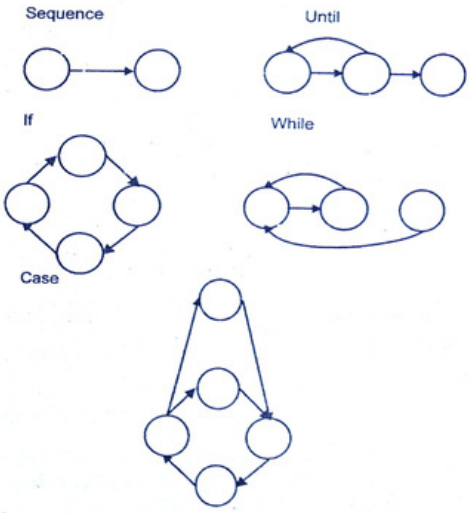

Fig. 2 : The structured constructs in the now graph form

These individual constructs combine together to produce the flow graph for a particular procedure.

Basic terminology associated with the flow graph is

Node: Each flow graph node represents one or more procedural statements. Each node that contains a condition is called a predicate node.

Edge: Edge is the connection between two nodes. The edges between nodes represent flow of control. An edge must terminate at a node, even if the node does not represent any useful procedural statements.

Region: A region in a flow graph is an area bounded by edges and nodes.

Cyclomatic complexity: Independent path is an execution flow from the start point to the end point Since a procedure contains control statements, there are various execution - paths depending upon decision taken on the control statements. So Cyclomatic complexity provides the number of such execution independent paths. Thus it provides a upper bound for number of tests that must produced because for each independent path, a test should be conducted to see if it is actually reaching the end point of the procedure or not.

Cyclomatic Complexity : Cyclomatic Complexity for a flow graph is computed in one of three ways:

1) The numbers of regions of the flow graph correspond to the Cyclomatic complexity.

2) Cyclomatic Complexity, V(G), for a flow graph G is defined as V(G) = E-N + 2) where E is the number of flow graph edges and N is the number of flow graph nodes.

3) Cyclomatic complexity, V(G), for a graph flow G is also defined as V(G) = P +I where P is the number of predicate nodes contained in the flow graph G.

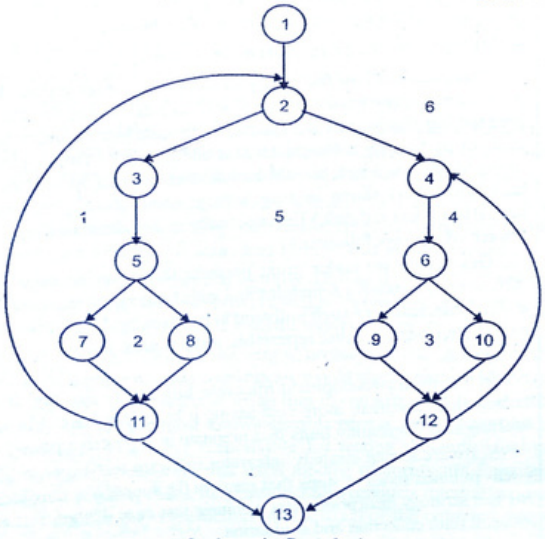

Fig. 3 : Cyclomatic Complexity

Example: Consider the following flow graph

Region, R = 6

Number of Nodes = 13

Number of edges = 17

Number of Predicate Nodes = 5

Cyclomatic Complexity, V(C) :

V(C) = R = 6;

Or

V(C) = Predicate Nodes + I

=5+ I =6

Or

V(C)= E-N+2

= 17-13+2 =6

Deriving Test Cases : The main objective of basic path testing is to derive the test cases for the procedure under test. The process of deriving test cases is following

1. From the design or source code, derive a flow graph.

2. Determine the Cyclomatic complexity, V(G) of this flow graph using any of the formula discussed above.

Even without a flow graph, V(G) can be determined by counting the number of conditional statements in the code and adding one to it.

3. Prepare test cases that will force execution of each path in the basis set.

Each test case is executed and compared to the expected results.

Graph Matrices : Graph matrix is a two dimensional matrix that helps in determining the basic set. It has rows and columns each equal to number of nodes in flow graph. Entry corresponding to each node-node pair represents an edge in flow graph. Each edge is represented by some letter (as given in the flow chart) to distinguish it from other edges. Then each edge is provided with some link weight, 0 if there is no connection and I if there is connection. For providing weights each letter is replaced by I indicating a connection. Now the graph matrix is called connection matrix. Each, row with two entries represents a predicate node. Then for each row sum of the entries is obtained and I is subtracted from it. Now the value so obtained for each row is added and I is again added to get the cyclomatic complexity.

Once the internal working of the different procedure are tested, then the testing for the overall functionality of program structure is tested. For this black box testing techniques are used which are discussed in the next section.

3. Black Box Testing :

Black box testing test the overall functional requirements of product. Input are supplied to product and outputs are verified. If the outputs obtained are same as the expected ones then the product meets the functional requirements. In this approach internal procedures are not considered. It is conducted at later stages of testing. Now we will look at black box testing technique.

Black box testing uncovers following types of errors.

1) incorrect or missing functions
2) interface errors
3) external database access
4) performance errors
5) Initialization and termination errors.

The following techniques are employed during black box testing

3.1. Equivalence Partitioning :

In equivalence partitioning, a test case is designed so as to uncover a group or class of error. This limits the number of test cases that might need to be developed otherwise. Here input domain is divided into classes or group of data. These classes are known as equivalence classes and the process of making equivalence classes is called equivalence partitioning. Equivalence classes represent a set of valid or invalid states for input condition.

An input condition can be a range, a specific value, a set of values, or a boolean value. Then depending upon type of input equivalence classes is defined. For defining equivalence classes the following guidelines should be used.

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.

2. If an input condition requires a specific value, then one valid and two invalid equivalence classes are defined.

3. If an input condition specifies a member of a set, then one valid and one invalid equivalence class are defined.

4. If an input condition is Boolean, then one valid and one invalid equivalence class are defined.

For example, the range is say, 0< count < Max I 000. Then form a valid equivalence class with that range of vaiues and two invalid equivalence classes, one with values less than the lower bound of range (i.e., count < 0) and other with values higher than the higher bound (count> 1000).

3.2. Boundary Value Analysis :

It has been observed that programs that work correctly for a set of values in an equivalence class fail on some special values. These values often lie on the boundary of the equivalence class. Test cases that have values on the boundaries of equivalence classes are therefore likely to be error producing so selecting such test cases for those boundaries is the aim of boundary value analysis.

In boundary value analysis, we choose input for a test case from an equivalence class, such that the input lies at the edge of the equivalence classes. Boundary values for each equivalence class, including the equivalence classes of the output, should be covered. Boundary value test cases are also called "extreme cases".

Hence, a boundary value test case is a set of input data that lies on the edge or boundary of a class of input data or that generates output that lies at the boundary of a class of output data.

In case of ranges, for boundary value analysis it is useful to select boundary elements of the range and an invalid value just beyond the two ends (for the two invalid equivalence classes. For example, if the range is 0.0 <= x <= 1.0, then the test cases are 0.0, 1.0 for valid inputs and -0.1 and 1.1 for invalid inputs.

For boundary value analysis, the following guidelines should be used:

1. For input ranges bounded by a and b, test cases should include values a and b and just above and just below a and b respectively.

2. If an input condition specifies a number of values, test cases should be developed to exercise the minimum and maximum numbers and values just above and below these limits.

3. If internal data structures have prescribed boundaries, a test case should be designed to exercise the data structure at its boundary.

4. Strategies towards Software testing

Now we know how the testing for software product is done. But testing software is not an easy task since the size of software developed for the various systems is often too big. Testing needs a specific systematic procedure, which should guide the tester in performing different tests at correct time. This systematic procedure is testing strategies, which should be followed in order to test the system developed thoroughly. Performing testing without some testing strategy would be very cumbersome and difficult. Testing strategies are discussed in the part two of this chapter which now follows.

Developers are under great pressure to deliver more complex software on increasingly aggressive schedules and with limited resources. Testers are expected to verify the quality of such software in less time and with even fewer resources. In such an environment, solid, repeatable, and practical testing methods and automation are a must.

In a software development life cycle, bug can be injected at any stage. Earlier the bugs are identified, more cost saving it has. There are different techniques for detecting and eliminating bugs that originate in respective phase.

Software testing strategy integrates software test case design techniques into a well- planned series of steps that result in the successful construction of software. Any test strategy incorporate test planning, test case design, test execution, and the resultant data collection and evaluation.

Testing is a set of activities. These activities so planned and conducted systematically that it leaves no scope for rework or bugs.

Various software-testing strategies have been proposed so far. All provide a template for testing. Things that are common and important in these strategies are

Testing begins at the module level and works "outward" : tests which are carried out, are done at the module level where major functionality is tested and then it works toward the integration of the entire system.

Different testing techniques are appropriate at different points in time: Under different circumstances, different testing methodologies are to be used which will be the decisive factor for software robustness and scalability. Circumstance essentially means the level at which the testing is being done (Unit testing, system testing, Integration testing etc.) and the purpose of testing.

The developer of the software conducts testing and if the project is big then there is a testing team: All programmers should test and verify that their results are according to the specification given to them while coding. In cases where programs are big enough or collective effort is involved for coding, responsibilities for testing lies with the team as a whole.

Debugging and testing are altogether different processes. Testing aims to finds the errors whereas debugging is the process of fixing those errors. But debugging should be incorporated in testing strategies

A software strategy must have low-level tests to test the source code and high-level tests that validate system functions against customer requirements.

Verification and Validation : Verification is a process to check the deviation of actual results from the required ones. This activity is carried out in a simulated environment so that actual results could be obtained without taking any risks.

Validation refers by the process of using software in a live environment in order to find errors. The feedback from the validation phase generally produces changes in the software to deal with bugs and failures that are uncovered.

Validation may continue for several months. During the course of validating the system, failure may occur and the software will be changed. Continued use may produce additional failures and the need for still more changes.

Planning for Testing : One of the major problem before testing is while planning. Because of natural reasons a developer would like to declare his program as bug free. But this does not essentially mean that the programmer himself should not test his program. He is the most knowledgeable person with context his own program. Therefore, he is always responsible for testing the individual units (modules) of the program, ensuring that each module performs the function for which it was designed. In many cases, the developer also conducts integration testing- a testing step that leads to the construction (and testing) of the complete program structure.

Only after the software architecture is complete does an Independent Test Group (ITG) become involved. ITG test the product very thoroughly. Both the developer and ITGs should be made responsible for testing. First developer should test the product after that ITG can do it. In this case since the developer knew that there are other people who will again test their product they'll conduct tests thoroughly. When developer and ITG work together, the product is tested thoroughly and unbiased.

Testing Strategies : Once it is decided who'll do testing then the main issue is how to go about testing. That is in which manner testing should be performed. As shown in fig. 4 first unit testing is performed. Unit testing focuses on the individual modules of the product. After that integration testing is performed. When modules are integrated into bigger program structure then new errors arise often. Integration testing uncovers those errors. After integration testing, other high order tests like system tests are performed. These tests focus on the overall system. Here system is treated as one entity and tested as a whole. Now we'll take up these different types of tests and try to understand their basic concepts.

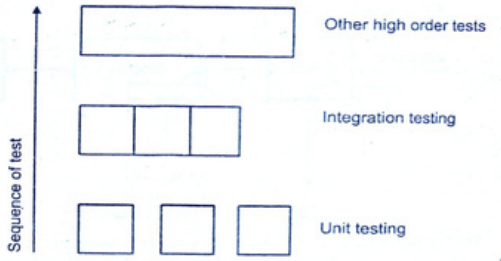

Fig. 4 : Sequence of tests

5. Unit Testing:

We know that smallest unit of software design is a module, Unit testing is performed to check the functionality of these units it is done before these modules are integrated together to build the overall system. Since the modules are small in size, individual programmers can do unit testing on their respective modules. So unit testing is basically white box oriented. Procedural design descriptions are used and control paths are tested to uncover errors within individual modules. Unit testing can be done for more than one module at a time.

The following are the tests that are performed during the unit testing:

Module interface test: here it is checked if the information is properly /lowing into the program unit and properly coming out of it.

Local data Structures: these are tested to see if the local data within unit (module) is Stored properly-by them.

Boundary Conditions: It is observed that much software often fails at boundary conditions. That's why boundary conditions are tested to ensure that the program is properly working at its boundary conditions.

Independent Paths : All independent paths are tested to see that they are properly executing their task and terminating at the end of the program.

Error Handling Paths: These are tested to check if errors are handled properly by them. See fig. 5 for overview of unit testing.

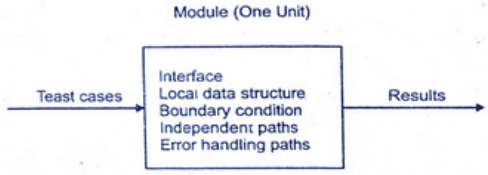

Fig. 5 : Unit test

Unit test procedure

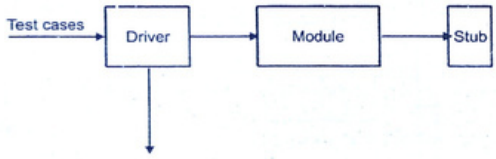

Fig. 6 : Unit test Procedure

Unit testing begins after the source code is developed, reviewed and verified for the correct syntax. Here design documents help in making test cases. Though each module performs a specific task yet it is not a stand-alone program. It may need data from some other module or it may need to send some data or control information to some other module. Since in unit testing each module is tested individually, so the need to obtain data from other module or passing other module is achieved by the use of stubs and drivers. Stubs and drivers are used to simulate those modules A driver is basically a program that accept test case data 'and passes that data to the module that is being tested. It also prints the relevant results. Similarly stubs are also programs that are used replace modules that are subordinate to the module to be tested. It does minimal data manipulation, prints verification of entry, and returns. Fig. 6 illustrates this unit test procedure.

Drivers and stubs are overhead because they are developed but are not a part of the product. This overhead can be reduced if these are kept very simple.

Once the individual modules are tested then these modules are integrated to form the bigger program structures. So next stage of testing deals with the errors

that occur while integrating modules. That's why next testing done is called integration testing, which is discussed next.

6. Integration Testing :

Unit testing ensures that all modules have been tested and each of them works properly individually. Unit testing does not guarantee if these modules will work fine if these are integrated together as a whole system. It is observed that many errors crop up when the modules are joined together. Integration testing uncovers errors that arises when modules are integrated to build the overall system.

Following types of errors may arise:

Data can be lost across an interface. That is data coming out of a module is not going / into the desired module.

Sub-functions, when combined, may not produce the desired major function.

Individually acceptable imprecision may be magnified to unacceptable levels. For example, in a module there is error-precision taken as +- 10 units. In other module same error-precision is used. Now these modules are combined. Suppose the error precision from both modules needs to be multiplied then the error precision would be +-100 which would not be acceptable to the system.

Global data structures can present problems: For example, in a system there is a global memory. Now these modules are combined. All are accessing the same global memory. Because so many functions are accessing that memory, low memory problem can arise.

Integration testing is a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing. The objective is to take unit tested modules, integrate them, find errors, remove them and build the overall program structure as specified by design.

There are two approaches in integration testing. One is top down integration and the other is bottom up integration. Now we'll discuss these approaches.

6.1. Top-Down Integration :

Top-down integration is an incremental approach to construction of program structure. In top down integration, first control hierarchy is identified. That is which module is driving or controlling which module. Main control module, modules subordinate to and ultimately subordinated to the main control block are integrated to some bigger structure. For integrating depth-first or breadth-first approach is used.

In depth first approach all modules on a control path are integrated first. See fig. 7. Here sequence of integration would be (MI, M2, M3), M4, MS, M6, M7, and M8.

In breadth first all modules directly subordinate at each level are integrated together. Using breadth first for fig. 6 the sequence of integration would be (MI, M2, M8), (M3, M6), M4, M7, and MS.

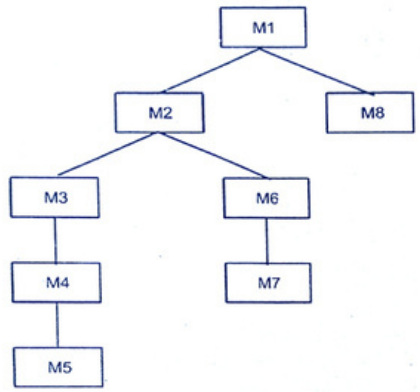

Fig. 7 : Top down integration

Another approach for integration is bottom up integration, which we are going to discuss now.

6.2. Bottom-Up Integration :

Bottom-up integration testing starts at the atomic modules level. Atomic modules are the lowest levels in the program structure. Since modules are integrated from the bottom up, processing required for modules that are subordinate to a given level is always available, so stubs are not required in this approach.

A bottom-up integration implemented with the following steps:

1. Low-level modules are combined into clusters that perform a specific software sub function. These clusters are sometimes called builds.
2. A driver (a control program for testing) is written to coordinate test case input and output.
3. The build is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

Fig. 8 shows the how the bottom up integration is done. Whenever a new module is added to as a part of integration testing, the program structure changes. There may be new data flow paths, some new I/O or some new control logic. These changes may cause problems with functions in the tested modules, which were working fine previously.

To detect these errors regression testing is done. Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that

changes have not propagated unintended side effects in the programs. Regression testing is the activity that helps to ensure that changes (due to testing or for other reason) do not introduce undesirable behavior or additional errors.

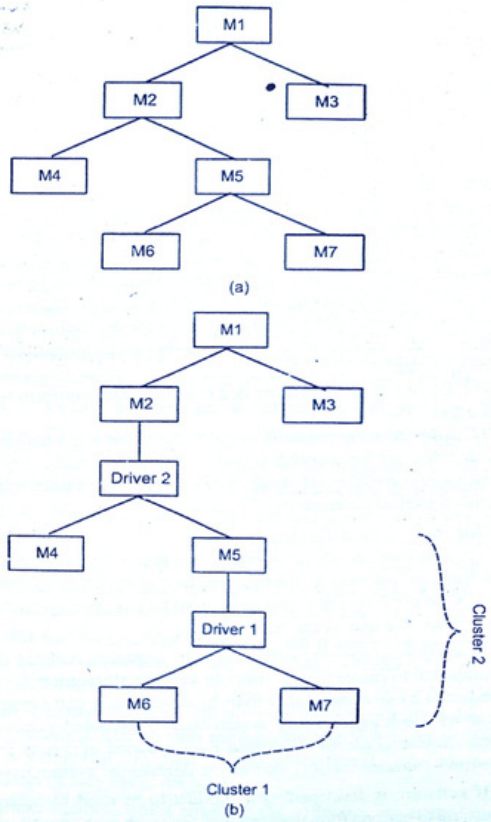

Fig. 8 : (a) Program Modules (b) Bottom-up integration applied to program modules in **(a)**

As integration testing proceeds, the number of regression tests can grow quite large. Therefore, regression test suite should be designed to include only those tests that address one or more classes of errors in each of the major program functions. It is impractical and inefficient to re-execute every test for every program functions once a change has occurred.

7. Validation Testing :

After the integration testing we have an assembled package that is free from modules and interfacing errors. At this stage a final series of software tests, validation testing begin. Validation succeeds when software functions in a manner that can be expected by the customer.

Major question here is what are expectations of customers. Expectations are defined in the software requirement specification identified during the analysis of the system. The specification contains a section titled "Validation Criteria" Information contained in that section forms the basis for a validation testing.

Software validation is achieved through a series of black-box tests that demonstrate conformity with requirements. There is a test plan that describes the classes of tests to be conducted, and a test procedure defines specific test cases that will be used in an attempt to uncover errors in the conformity with requirements.

After each validation test case has been conducted, one of two possible conditions exists:

(1) The function or performance characteristics conform to specification and are accepted, or

(2) A deviation from specification is uncovered and a deficiency list is created. Deviation or error discovered at this stage in a project can rarely be corrected prior to scheduled completion. It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

7.1. Alpha and Beta Testing :

For a software developer, it is difficult to foresee how the customer will really use a program. Instructions for use may be misinterpreted; strange combination of data may be regularly used; and the output that seemed clear to the tester may be unintelligible to a user in the field.

When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements. Acceptance test is conducted by customer rather than by developer. It can range from an informal "test drive" to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.

If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one. Most software product builders use a process called alpha and beta testing to uncover errors that only the end user seems able to find.

Customer conducts the alpha testing at the developer's site. The software is used in a natural setting with the developer. The developer records errors and usage problem. Alpha tests are conducted in a controlled environment.

The beta test is conducted at one or more customer sites by the end user(s) of the software. Here, developer is not present. Therefore, the beta test is a live application of the software in an environment that cannot be controlled by the developer. The customer records all problems that are encountered during beta testing and reports these to the developer at regular intervals. Because of problems reported during beta test, the software developer makes modifications and then prepares for release of the software product to the entire customer base.

8. System Testing :

Software is only one element of a larger computer-based system. Ultimately, software is incorporated with other system elements and a series of system integration and validation tests are conducted. These tests fall outside the scope of software engineering process and are not conducted solely by the software developer.

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all system elements have been properly integrated and perform allocated functions. In the following section, different system tests are discussed.

8.1. Recovery Testing :

Many computer-based systems must recover from faults and resume operation within a pre-specified time. In some cases, a system may be fault tolerant; that is, processing faults must not cause overall system function to cease. In other cases, a system failure must be corrected within a specified period or severe economic damage will occur.

Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. It the recovery is automated (performed by system itself), re-initialization mechanisms, data recovery, and restart are each evaluated for correctness. If the recovery requires human intervention, the mean time to repair is evaluated to determine whether it is within acceptable limits.

8.2. Stress Testing :

Stress tests are designed to confront program functions with abnormal situations. Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example, **(1)** special tests may be designed that generate 10 interrupts are seconds, when one or two is the average rate; (2) input data rates may be increased by an order of magnitude to determine how input functions will respond; (3) test cases that require maximum memory or other resources may be executed; (4) test cases that may cause excessive hunting for disk resident data may be created; or (5) test cases that may cause thrashing in a virtual operating system may be designed. The testers attempt to break the program.

8.3. Security Testing :

Any computer-based system that manages sensitive information or causes actions that can harm or benefit individuals is a target for improper or illegal penetration.

Security testing attempts to verify that protection mechanism built into a system will protect it from unauthorized penetration. During security testing, the tester plays the role of the individual who desires to penetrate the system. The tester may attack the system with custom software designed to break down any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to find the key to system entry; and so on.

Given enough time and resources, good security testing will ultimately penetrate a system. The role of the system designer is to make penetration cost greater than the value of the information that will be obtained in order to deter potential threats.

9. Role of Quality in Software Development :

Till now in this chapter we have talked about the testing of the system. Testing is one process to have a error free system but it is only a small part of the quality assurance activities, which are employed for achieving the good quality and error free system. We have not yet talked about quality of system produced. Quality enforcement is very important in system development. In this part we'll study how we can apply quality factors in order to produce a quality system. Here mainly software system and quality is considered.

We employ software to solve and simplify our real-life problems and it goes without saying that it should be of high class i.e. quality software, thus here the software quality and it's assurance come into the picture. Software quality assurance is an umbrella activity, which must be applied throughout the software engineering process.

But this leads us to few questions such as how to define quality and what exactly it is with respect to software. Hence it becomes a subjective issue but then there are some basic do's and don'ts to be taken care of during the course of software cycle which if applied thoroughly will lead to a reasonable quality software and that is what the following definition of software quality states- Conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally developed software. Though this definitions speak a volume about the matter, to further make it more simple for our understanding we can focus on the points given below.

1. Software requirements have to be met by the newly developed system and thus any lack of conformance in them will be serious quality lag.

2. Specified standards define a development criteria which displays the manner in which the software is engineered. Any deviation from the given criteria will result into lack of the quality.

3. No matter how much vocal, straight, honest and frank the client may be but then there are always a set of implicit requirements which are not mentioned but they should be met.

The dominant nature of the software market is its dynamic growth. New application areas are continually emerging along with birth of new technologies. New methods of information processing and new programming environments have created a dilemma in management of software development. For software developers whatever they produce, it must be sold in order to survive in this dynamic software industry. Also these companies should always try for continuous improvement in quality.

For quality, Software Engineering Institute Carnegie Mellon University, Pennsylvania has devised a quality model known as Capability Maturity Model CMM. This model is based on the principles of Total Quality Management (TQM) for improving the entire software development process. The goal of CMM is that a firm's process capability is reflected to the extent to which it plans for and monitors software quality and customer satisfaction. As organizations grow, software project management practices are adequately utilized and software development undergoes a more refilled process. In CMM , the process capability of a company is understood in five advancing maturity levels, a brief overview of them is given below:

LEVEL 1 : Initial - processes are not adequately defined and documented.

LEVEL 2 : Repeatable - implementation of project management tools.

LEVEL 3 : Defined - all projects use an organized, documented and standardized set of activities that are implemented throughout the project life cycle.

LEVEL 4 : Managed - Specific process and quality outcome measures are implemented.

LEVEL 5 : Optimized - Continuous process improvement is adopted in the entire organization. Sophisticated methods of defect prevention, technology changes management and process change management is implemented.

The CMM has become tool to evaluate the potential of an organization to achieve high quality in software development. Each s/w company should aim to achieve high CMM level.

Software Quality Factors : Till now we have been talking s/w quality in general. What it means to be a quality product. We also looked at CMM in brief We need to know various quality factors upon which quality of a s/w produced is evaluated. These factors are given below.

The various factors, which influence the software, are termed as software factors. They can be broadly divided into two categories. The classification is done on the basis of measurability. The first category of the factors is of those that can be measured directly such as number of logical errors and the second category clubs those factors which can be measured only indirectly for example maintainability but

the each of the factors are to be measured to check for the content and the quality control. Few factors of quality are available and they are mentioned below;

Correctness - extent to which a program satisfies its specification and fulfills the client's objective.

Reliability - extent to which a program is supposed to perform its function with the required precision.

Efficiency - amount of computing and code required by a program to perform its function.

Integrity - extent to which access to software and data is denied to unauthorized users.

Usability- labor required to understand, operate, prepare input and interpret output of a program.

Maintainability- effort required to locate and fix an error in a program.

Flexibility- effort needed to modify an operational program.

Testability- effort required to test the programs for their functionality.

Portability- effort required to run the program from one platform to other or to different hardware.

Reusability- extent to which the program or it's parts can be used as building blocks or as prototypes for other programs.

Interoperability- effort required to couple one system to another.

Now as you consider the above-mentioned factors it becomes very obvious that the measurements of all of them to some discrete value are quite an impossible task. Therefore, another method was evolved to measure out the quality. A set of matrices is defined and is used to develop expressions for each of the factors as per the following expression

$$Fq = C1 * M1 + C2 * M2 + \ldots\ldots\ldots Cn * Mn$$

where Fq is the software quality factor, Cn are regression coefficients and Mn is metrics that influences the quality factor. Metrics used in this arrangement is mentioned below.

Auditability- ease with which the conformance to standards can be verified,

Accuracy - precision of computations and control

Communication commonality- degree to which standard interfaces, protocols and bandwidth are used.

Completeness- degree to which full implementation of functionality required has been achieved,

Conciseness- program's compactness in terms of lines of code,

Data commonality - use of uniform design and documentation techniques throughout the software-development.

Data commonality- use of standard data structures and types throughout the program.

Error tolerance - damage done when program encounters an error.

Execution efficiency- run-time performance of a program.

Expandability - degree to which one can extend architectural, data and procedural design.

	Correctness	Reliability	Efficiency	Integrity	Maintainability	Flexible	Testability	Portability	Reusability	Interoperability	Usability	
Audit ability	X	X			X							
Accuracy			X									
Communication Commonality	X	X			X							
Completeness		X	X									
Complexity	X					X						
Concision	X	X						X		X		
Data commonality		X	X								X	
Error tolerance	X					X						
Execution efficiency		X						X			X	
Expand ability												
Generality			X									
H/w independence					X		X					X
Instrumentation												
Modularity									X			
Operability							X					
Security					X							
Self documentation Simplicity												

Fig. 9 : Quality factors and metrics

Hardware independence- degree to which the software is de-coupled from its operating hardware.

Instrumentation- degree to which the program monitors its own operation and identifies errors that do occur.

Modularity- functional independence of program components.

Operability- ease of programs operation.

Security- control and protection of programs and database from the unauthorized users.

Self-documentation- degree to which the source code provides meaningful documentation. -

Simplicity- degree to which a program is understandable without much difficulty. Software system independence degree to which program is independent of nonstandard programming language features, operating system characteristics and other environment constraints.

Traceability- ability to trace a design representation or actual program component back to initial objectives.

Training-degree to which the software is user-friendly to new users.

There are various 'checklists' for software quality. One of them was given by Hewlett Packard that has been given the acronym FURPS - for Functionality, Usability, Reliability Performance and Supportability.

Functionality is measured via the evaluation of the feature set and the program capabilities, the generality of the functions that are derived and the overall security of the system.

Considering human factors, overall aesthetics, consistency and documentation assesses usability.

Reliability is figured out by evaluating the frequency and severity of failure, the accuracy of output results, the mean time between failure (MTBF), the ability to recover from failure and the predictability of the program.

Performance is measured by measuring processing speed, response time, resource consumption, throughput and efficiency.

Supportability combines the ability to extend the program, adaptability, serviceability or in other terms maintainability and also testability, compatibility, configurability and the ease with which a system can be installed.

Software Quality assurance : Software quality assurance is a series of planned, systematic sequential actions that enable the quality of the software produced. Usually all software development units have their own SQA team.

This SQA team takes care of the quality at the high end that is the overall product and at the lower order the quality is the sole responsibility of the individual who may engineer, review and test at any level.

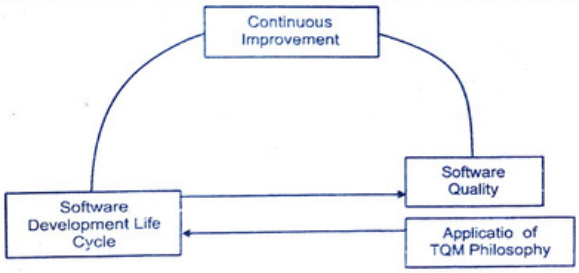

Fig. 10 : The adoption of the philosophy of Continuous Improvement in the software development life cycle.

Software quality has to be a characteristic of the software produced and thus it is designed into it rather than to be imposed later. It is the duty of every individual in the software development team that quality is maintained even before some formal quality assurance procedures are applied. This practice can improve the quality of all the software produced by the organization.

10. Activities Involved :

Software quality assurance clubs together various tasks and activities. There are seven major activities namely application of technical methods, conducting of Formal Technical Reviews (FTR), software- testing, control of change, measurement and record keeping and reporting.

SQA starts early along with the process of software development. It is initiated with the help of technical methods and tools that enable the analyst to achieve a high quality specification and help the designer to develop a high-quality design. We can always point out these measures of specification and design quality and once this is done then both the specifications (or the prototype) and the design are individually checked for quality.

FTR is employed for this purpose and by its means the members of the technical staff undergo a meeting to identify the quality problems. Many-a-times reviews are found to be equally effective just as the testing of uncovering defects in software. The next step in the software testing which involves a series of test case design methods to enable effective error detection. It is believed that the software testing can dig out most of the errors but practically as the fact goes no matter how rigorous the testing may be it is unable to uncover all the errors. Some errors remain uncovered. Hence we have to look up to other measures also but this should not diminish the importance of software testing.

Application of formal and systematic standards and procedures vary from project to project and company to company. Either the standards are self-imposed in the whole of the software engineering process by the software development team

or they are carried out as per the client's dictation or as a regulatory mandate. If formal i.e. written, well documented standards are not available then it becomes imperative to initiate an SQA activity so that the standards are complied with. Also an assessment of this compliance must be regularly undertaken either by the means of FIR or as an SQA group audits which the team may do on its own.

Nothing bothers the whole of the software engineering process more than the changes. Any and every change to software will almost incorporate an error or trigger side effects that will propagate errors. The ill influence of changes can vary with nature of the change and it also depends at which stage they are to be incorporated. Early changes are much less harmful than the ones, which are taken into design at a later stage. This is so because any change in the specification or the design will demand a re-lay out of all the work done so far and in the subsequent plans. Hence the importance of a well-defined specification and detail design are again highlighted. Before the coding starts these designs are to be freezed and all the software development process after the detail design takes the design skeleton as the base on which the software is built. Therefore both the client and the software development team should be sure of the design developed before they freeze it and the process goes to the next step. Any doubts or requests should be taken into account then and there only. Any further request to add new features from the client will lead to changes in the design and it will result in more effort and time requirements on the software development team's behalf. Needless to add, this will invariably result in the increase in the software cost. Hence to minimize these effects, change control process activity is used. It contributes to the software quality by formalizing the requests for change by evaluating its nature and also controlling its impact. Change control is applied during software development and also later during the maintenance phase.

Measurement is such an activity that it cannot be separated from any engineering discipline as it is an integral part of it. So it comes into action here also. Software metrics is used to track software quality and to evaluate the impact of methodological and procedural changes. These metrics encompass a broad array of technical and management oriented measures.

Records are usually developed, documented and maintained for reference purposes. Thus record keeping and recording enable the collection and distribution of SQA information. The documents of reviews, results of FTR, audits, change control, testing and other SQA activities become a part of the project and can be referred by the development staff on a need to know basis.

We now know various activities involved in quality assurance activity. Now we'll take one activity at a time and look into it in detail.

10.1. Application of Technical methods :

There are two major objectives to be achieved by employing technical methods, firstly the analyst should achieve a high quality specification and the designer should develop a high quality design.

Firstly we will take up the specification part. There is no doubt on the fact that a complete understanding and clarity of software requirements is essential for the appropriate and suitable software development. No matter how well designed, well coded but a poorly analyzed and specified program will always be a problem for the end-user and bring disappointment to the developer.

The analyst plays the major role in the requirements analysis phase and he must exhibit the following traits...

The ability to grasp abstract concepts, reorganize into logical divisions and synthesize "solutions" based on each division.

The ability to absorb pertinent facts from conflicting or confused sources.

The ability to understand the user/client environment.

The ability to apply the hardware and/or software system elements to the user/client environments.

Software requirements must be uncovered in a "top-down" manner, major functions, interfaces and information must be fully understood before successive layers of detail are specified.

Modeling : During software requirements analysis, models of the system to be built are created and these models focus on what the system must do and not on how it has to do. These models are a great help in the following manner....

The model aids the analyst in understanding about the system's function, behavior and it's other information thus making the analysis task easier and systematic.

The model is the standard entity for review and it is the key to determine completeness, consistency and accuracy of the specification.

The model is the base/foundation for the design and thus it serves as the representation of the software that can be "mapped" into an implementation context.

Partitioning : Quite often problems are too large and complex to be grasped as a whole and then it is very reasonable to partition/divide them into smaller parts so that they individually become much easier to understand and in the end one can establish interfaces between the parts so that the overall function can be accomplished. During the requirement analysis the information, functional and behavioral domains of software can be partitioned.

Specification Principles : Specification methods irrespective of the modes via which they are carried out are basically a representation process. Requirements are represented in such a manner that leads to successful software implementation. Blazer and Goldman proposed eight principles of good specification and they are given as follows :

1. Separate functionality from implementation: As per the definition, the specification is a description of what is desired rather than how it is to be realized/

implemented. Specification can have two forms. The first form is that of mathematical functions in which as per the given set of inputs a particular set of outputs is produced. In such specifications, the result to be obtained is entirely expressed in a what rather than how form. Thus the result is the mathematical function of the input.

2. A process-oriented systems specification language is required: Here we will take up the second form of the specification. In this situation the environment is dynamic and its changes affect the behavior of some entity interacting with the environment. This is similar to the case or embedded computer system. Here no mathematical function can express the behavior. To express the same we have to use process-oriented description in which the what specification is expressed by specifying a model of the required behavior in the terms of functional responses to various inputs from the environment. Now such kind of process-oriented specifications, which represent the model of the system behavior, have been usually excluded from the formal specification languages but one can not do without them if more complex dynamic situations are to be expressed.

3. A specification must encompass the system of which the software is a component. Interacting components make up a system. The description of each component is only possible in the complete context of the entire system therefore usually a system can be modeled as collection of passive and active components. These objects are interrelated and their relationship to each other is time-variant. Stimulus to active objects or agents as they are called are given by the dynamic relationships and to these the agents respond which may further cause changes to which again the agents respond.

4. A specification must encompass the environment in which the system operates. Similarly, the environment in which the system operates and with which it interacts must be specified.

10.2. FTR (Formal Technical Review) :

The FTR is a software quality assurance activity with the objectives to uncover errors in function, logic or implementation for any representation of the software; to verify that the software under review meets its requirements; to ensure that the software has been represented according to predefined standards; to achieve software that is developed in a uniform manner and to make projects more manageable.

FTR is also a learning ground for junior developers to know more about different approaches to software analysis, design and implementation. It also serves as a backup and continuity for the people who are not exposed to the software development so far. FTR activities include walkthroughs, inspection and round robin reviews and other technical assessments. The above-mentioned methods are different FTR formats.

Review Meetings : Review meeting is important form of FTR and there are some essential parameters for the meeting such as there should be reasonable number of persons conducting the meeting and that too after each one of them has done his/her homework i.e. some preparation and the meeting should not be carried out very long which may lead to wastage of time but rather for duration just enough to churn out some constructive results. FTR is effective when a small and specific part of the overall software is under scrutiny. It is easier and more productive to review in small parts like each module one by one rather than to review the whole thing in one go. The target of the FTR is on a component of the project, a single module. The individual or the team that has developed that specific module or product intimates the product is complete and a review may take place. Then the project leader forwards the request to the review leader who further informs the reviewers who undertake the task. The members of the review meeting are reviewers who undertake the task. The members of the review meeting are reviewers, review-leader, product developers (or the module leader alone) and there one of the reviewers takes up the job of the recorder and notes down all the important issues raised in the meeting.

At the end of each review meeting the decision has to be taken by the attendees of the FTR on whether to accept the product without further modification or to reject the product due to severe errors or to accept the product provisionally. All FIR attendees sign off whatever decision taken. At the end of the review a review issues list and a review summary is report is generated.

10.3. Software Testing

SQA is incomplete without testing and testing is carried out to verify and confirm that the software developed is capable enough to implement, carry out the tasks for which it has been developed. Tests carried should be able to determine any previous undetected errors that might hamper the smooth functioning of the system. There are two different approaches to it. First is the classroom testing where we test and retest the program until they work but in our business environment a more professional approach is taken where we actually try to make the programs fail and then continue testing until we cannot deliberately make them fail any more. Hence the objective is to take care of all possible flaws so that no null, void, vague case or doubt remains before it is installed at the client's site. The data used for testing can either be an artificial or mock data when the system is new or it can be taken from the old system when a new system has replaced it. The volume of data required is very high for adequate testing. This is because all the data used may not be able to mirror all the real situations that the system will encounter later. Testing is such an important stage in the software development that usually 30% of the software development cycle time is dedicated to it.

10.4. Control of Change

During the development of software product, changes are difficult to avoid. There is always a need for some change in the software product. Though these changes must be incorporated into software product, but there should be a specific procedure that must be followed for putting the changes in the product. Before a change is implemented, the required change should be thoroughly analyzed, recorded

and reported to people who are responsible for controlling them for quality and errors. For all these activities there is software configuration management (SCM). It is applied throughout the software engineering process as changes can occur at any stage of software development. SCM activities identifies and control changes and ensures that these changes are properly implemented.

A typical change control process that is followed in many software development process is illustrated in fig 11.

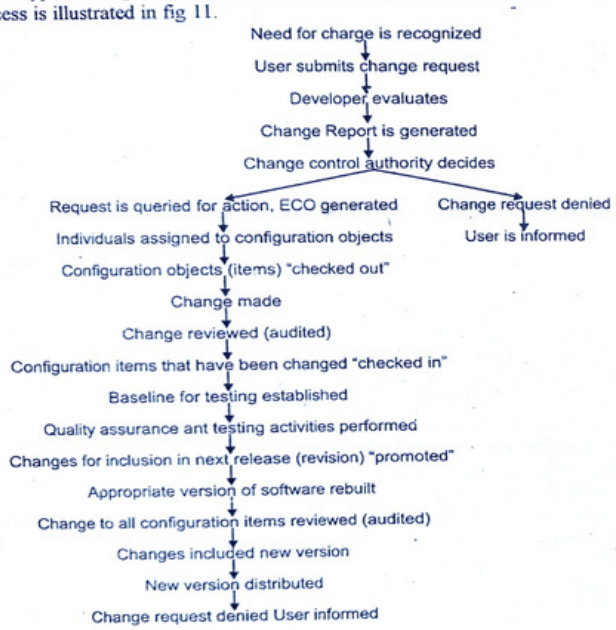

Fig. 11 : Change Control Process

First thing is need for change is recognized and request of change is submitted. Developers analyze the request and makes a change report and it is submitted to change control authority. It is up to the authority to grant permission or deny request. In case change request is denied, the user is informed about it. If permission is granted then an Engineer Change Order (ECO) is generated which contains details of the changes to be made. The individual software engineer is assigned the configuration object that requires change. These objects are taken out of system, changes are made upon them and finally changes are reviewed and again they are

put into system.

Testing strategy is finalized. Quality assurance and testing activities are performed. Changes done to the system are promoted. New version of software is built. Changes made to the software product are again reviewed and finally the new version with the required changes is distributed.

There are two more activities in SCM. These are 'check in' and 'check out' processes for access control and synchronization control.

Access control determines which software engineer has the authority to change and modify a configuration object. More than one software engineer call have the authority over a particular object. Access control is very important it is not advisable to grant access to objects to everyone. Then everybody can make changes to the object

Synchronization controls the parallel changes, done by two or more developers. It ensures one developer is not writing over the work of the other developer. Fig. 12 shows access and synchronization control. It shows whenever there is a need to change some object, it is given to software engineer who have access to that object.

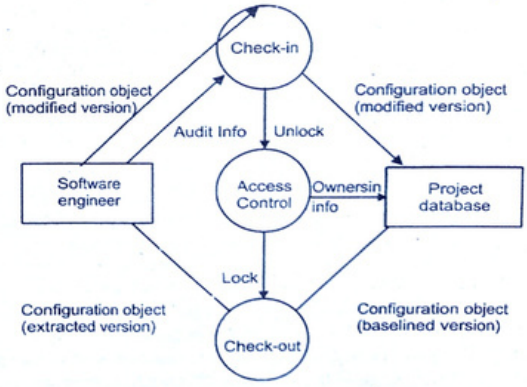

Fig. 12 : Access and synchronization control

Once it is assigned to a software engineer for change then it is locked for other software engineers who might also have access for it. This process is checkout for that object. The software engineer modifies the object and it is reviewed. It is then check in. That lock that has been put on the object is unlocked and can be given to

other software engineers if required. In this way there is no chance of one developer writing over the work of other developer.

10.5. Measurement :

Earlier in our course we discussed what is the objective and meaning of the quality with respect to software development. We defined a set of qualitative factors of the software quality measurement. Since there is no such thing as absolute knowledge we can't expect to measure software quality exactly. Therefore SQ metrics is developed and employed to figure out the measure, content of the quality. A set of software metrics is applied to the quantitative assessment of software quality. In all cases these metrics use indirect measures therefore quality as such in itself is never measured but rather some manifestation of it.

There are a number of software quality indicators that are based on the measurable design characteristics of a computer program.

Design structural quality index (DSQI) is one such measure. The following values must be ascertained to compute the DSQI

S^1 = the total number of modules defined in the program architecture

S_2 = the number of modules whose correct function depends on the source of data input or that produces data to be used elsewhere {in general control modules (among others) would not be counted as part of S_2}

S_3 = the number of modules whose correct function depends on prior processing

S_4 = the number of database items (includes data objects and all attributes that define objects)

S_5 = the total number of unique database items

S_6 = the number of database segments (different records or individual objects)

S_7 = the number of modules with a single entry and exit (exception processing is not considered to be a multiple exit)

When all these values are determined for a computer program, the following intermediate values can be computed:

Program structure: D_1, where D_1 is defined as follows:

If the architectural design was developed using a distinct method (e.g., data flow-oriented design or object oriented design), then $D_1 = 1$; otherwise $D_1 = 0$.

Module independence: $D_2 = 1 - (S_2/S_1)$

Module not dependent on prior processing: $D_3 = 1 - (S_3/S_1)$

Database size: $D_4 = 1 - (S_5/S_4)$

Database compartmentalization: $D_5 = 1 - (S_6/S_4)$

Module entrance/exit characteristic: $D_6 = 1 - (S_7/S_1)$

With the intermediate values determined, the DSQI is computed in the following manner:

$DSQI = \sum W_i D_i$

Where $i = 1$ to 6, W_i is the relative weighting of the importance of each of the intermediate values, and $\sum W_i = 1$ (if all D_i are weighted equally, then $W_i = 0.167$).

The value of DSQI for past designs can be determined and compared to a design that is currently under development. If the DSQI is significantly lower than average, further design work and review is indicated. Similarly, if major changes are to be made to an existing design, the effect of those changes on DSQI can be calculated.

IEEE Standard 982.1-1988 suggests a software maturity index (SMI) that provides an indication of the stability of a software product (based on changes that occur for each release of the product). The following information is determined:

MT = the number of modules in the current release

F_c = the number of modules in the current release that have been changed

F_a = the number of modules in the current release that have been added

F_d = the number of modules from the preceding release that were deleted in the current release

The software maturity index is computed as:

$$SMI = \frac{[MT - (F_a + F_c + F_d)]}{MT}$$

As SMI approaches 1.0, the product begins to stabilize.

10.6. Record Keeping and Reporting :

Record keeping and reporting are another quality assurance activities. It is applied to different stages of software development.

During the FTR, it is the responsibility of reviewer (or recorder) to record all issues that have been raised.

At the end of review meeting, a review issue list that summarizes all issues, is produced. A simple review summary report is also compiled.

A review summary report answers the following questions.

1. What was reviewed?
2. Who reviewed it ?
3. What were the findings and conclusions?

Fig. 13 shows a sample summary report. This becomes an important document and may be distributed to project leader and other interested parties.

Technical Review Summary Report

Review identification:
Project: XXX00 Review Number : X-00012
Date: 11th July' 1996 Location: Old Bldg, Room# 4
Time: 10:00 AM

Product identification
Material Reviewed : Detailed Design module for penalty collection
Producer: Gargi
Brief Description : Penalty collection module for books returned after due date.
Material Reviewed: (note each item separately)
1. Penalty rules
Review Team: Signature
1. Ashish (Leader)
2. Vikram (Recorder)
3. Nagender

Product Appraisal:
Accepted: as is ( ) with minor modifications ( )
Not Accepted: major revision ( ) minor revision ( )
Review Not completed: (Explanation follows)
Supplementary material attached:
Issue list ( ) Annotated Materials ( )
Other (describe)

Fig. 13 : Technical review summary report.

The review issue list serves the following purpose.

1. To identify problem areas within the product.
2. To serve as an action item checklist that guides the producer as corrections are made.

Fig. 14 shows a issue list.

Review Number	:	XX10		
Date of Review	:	07-07-98		
Review leader	:	Ashish	Recorder	: Vikram

ISSUE LIST

1. Penalty rules ambiguous. The penalty rules are not clear. Needs more clarity.
Review team recommends a modification in the penalty rules module.

Fig. 14 : Review issues list

It is important to establish a follow-up procedure to ensure that items on the issue list have been properly corrected.

Review issue list provides the basis for the follow-up procedure to be carried out. After the follow up procedure, review issue list can be used to cross-check the corrections made. Once all this is done and review findings are incorporated, the quality of the system is ensured. This is how various techniques and procedures may be used for delivering a quality system to the vendors.

Questions

Very Short Questions:

1. What is testing ?
2. What is test case ?
3. Name different types of testing ?
4. What is FTR ?
5. What is bug ?

Short Questions:

1. What are fundamentals of software testing ?
2. Define Endurance Testing ?
3. What do you mean by validation and Verification ?
4. What do you mean by Integration testin ?
5. What is test driver and test stub ?

Long Question :

1. What do you mean by Quality Assurance ? Discuss it's role in software evelopment.
2. What is System testing ? Discuss all type of System Testing.
3. What is the difference between white box & black box testing ?
4. What do you mean by integration testing ?
5. Write Short Note on -
   (a) Verification and Validation
   (b) White Box and Black Box Testing
   (c) Top down Integration and Bottom-up integration.
   (d) Unit Testing
   (e) System Testing
   (f) Record Keeping System
   (g) FTR (Formal Technical Review)

❑❑❑

10

DOCUMENTATION IMPLEMENTATION AND MAINTENANCE

Objectives

1. Implementation Phase :

In this phase, the system or system modifications are installed and made operational in a production environment. The phase is initiated after the system has been tested and accepted by the user. Activities in this phase include notification of implementation to end users, execution of the previously defined training plan, data entry or conversion, completion of security certification and accreditation and post

implementation evaluation. This phase continues until the system is operating in production in accordance with the defined user requirements.

The new system can fall into three categories, replacement of a manual process, replacement of a legacy system, or upgrade to an existing system. Regardless of the type of system, all aspects of the implementation phase should be followed. This will ensure the smoothest possible transition to the organization's desired goal.

1.1. Tasks and Activities :

The following activities are performed as part of the implementation phase. A description of these tasks and activities is provided below.

Notification of Implementation : The implementation notice should be sent to all users and organizations affected by the implementation. Additionally, it is good policy to make internal organizations not directly affected by the implementation aware of the schedule so that allowances can be made for a disruption in the normal activities of that section. The notice should include:

- The schedule of the implementation;
- A brief synopsis of the benefits of the new system;
- The difference between the old and new system;
- Responsibilities of end user affected by the implementation during this phase; and
- The process to obtain system support, including contact names and phone numbers.

Execution of Training Plan : It is always a good business practice to provide training before the end user uses the new system. Because there has been a previously designed training plan established, complete with the system user manual, the execution of the plan should be relatively simple. Typically what prevents a plan from being implemented is lack of funding. Good budgeting should prevent this from happening.

Data Entry or Conversion : With the implementation of any system, typically there is old data which is to be included in the new system. This data can be in a manual or an automated form. Regardless of the format of the data, the tasks in this section are two fold, data input and data verification. When replacing a manual system, hard copy data will need to be entered into the automated system. Some sort of verification that the data is being entered correctly should be conducted throughout this process. This is also the case in data transfer, where data fields in the old system may have been entered inconsistently and therefore affect the integrity of the new database. Verification of the old data becomes imperative to a useful computer system.

One of the ways verification of both system operation and data integrity can be accomplished is through parallel operations. Parallel operations consists of running the old process or system and the new system simultaneously until the new system is certified. In this way if the new system fails in any way, the operation can proceed on the old system while the bugs are worked out.

Install System : To ensure that the system is fully operational, install the system in a production environment.

Post-Implementation Evaluation : After the system has been fielded, a post-implementation evaluation is conducted to determine the success of the project through its implementation phase. The purpose of this evaluation is to document implementation experiences to recommend system enhancements and provide guidance for future projects. In addition, change implementation notices will be utilized to document user requests for fixes to problems that may have been recognized during this phase. It is important to document any user request for a change to a system to limit misunderstandings between the end user and the system programmers.

Review Previous Documentation : During this phase, the documentation from all previous phases will be finalized to align it with the delivered system. The Project Manager coordinates these update activities.

1.2. Roles and Responsibilities :

Project Manager: The project leader is responsible and accountable for the successful execution of the Implementation Phase. The project leader is responsible for leading the team that accomplishes the tasks shown above.

Project Team: The project team members (regardless of the organization of permanent assignment) are responsible for accomplishing assigned tasks as directed by the Project Manager.

Procurement Officer: The Procurement Officer is responsible and accountable for preparing solicitation documents under the guidance of the project manager.

Oversight Activities: Agency oversight activities, including the IT office, provide advice and counsel for the Project Manager on the conduct and requirements of the Implementation Phase. Additionally, oversight activities provide information, judgments, and recommendations to the Agency decision makers during project reviews and in support of project decision milestones.

1.3. Deliverables, Responsibilities and Action :

The following deliverables are completed during the Implementation Phase:

Delivered System : After the Integration and Test Phase is completed and all documents are approved, the system - including the production version of the data repository - is delivered to the customer for the Operations and Maintenance Phase.

Change Implementation Notice : A formal request and approval document for changes made during the Implementation Phase.

Version Description Document : The primary configuration control document used to track and control versions of software released to the operational environment. It is a summary of the features and contents for the software build and identifies and describes the version of the software being delivered.

Post-Implementation Review Report : This report is created at the end of the Implementation Phase. It is conducted to ensure that the system functions as planned and expected; to verify that the system cost is within the estimated amount; and to verify that the intended benefits are derived as projected.

1.4. Issues for Consideration :

Implementation represents the culmination of many threads of activity within the project. As the Project Manager pulls them all together, it's important not to overlook critic activities that are not directly associated with the technology implementation, such as:

- Execution of the communications strategy and plan to ensure all participants understand their roles and the objectives of each implementation activity.
- Effective delivery of user training.
- Close management of the data conversion process and products.
- Change management and data verification to ensure that users develop trust in the system's products as early in the process as possible.
- Rigorous documentation of all activities.

1.5. Review Activity :

A post-implementation review shall be conducted to ensure that the system functions as planned and expected; to verify that the system cost is within the estimated amount; and to verify that the intended benefits are derived as projected. Normally, this shall be a one-time review, and it occurs after a major implementation; it may also occur after a major enhancement to the system. The results of an unacceptable review are submitted to the Agency for review and follow-up actions.

During the Implementation Phase Review, recommendations may be made to correct errors, improve user satisfaction or improve system performance. For contractor development, analysis shall be performed to determine if additional activity is within the scope of the statement of work or within the original contract.

2. Documentation :

It is a truth universally acknowledged, that software documentation is a Good Thing and described what they had seen in the way of good practice for financial modelling systems: "Acceptable standards of documentation were established, agreed by the firm, and themselves documented." They went on to say "The standards of control and documentation applicable to systems developments,

But what is documentation for? It seems to be more or less assumed that all documentation is worth while, and the more documentation the better. However, given that most financial models are developed with limited resources, and writing documentation takes time, it's important to consider what forms of documentation are most useful and productive. In order to do this we must think about what we are trying to achieve through the production of documentation.

2.1. What Documentation Does :

Documentation may do any of the following:

Specify the intended working of the document

Record what was done

Explain how the document works

Instruct the user how to use or update the document

Ideally, a documentation serving all four purposes. A specification may be anything from a single sentence to a separate, long, document. Documentation is usually easier to understand if it is written for a single purpose. It really helps a user if instructions are written (and labelled) as instructions, rather than buried deep in the record of a change.

2.2. What Documentation Helps who? :

There is no type of documentation that is always useless to any type of user, but in general some types are more useful than others. The following table summarizes the general utility of the different types of documentation to different users:

	Specify	Record	Explain	Instruct
Viewer	X	X	X	X
Player	X	X	X	X
Tester	X	X	X	X
Changer	X	X	X	X
Developer	X	X	X	X
Auditor	X	X	X	X

Table 1 : The table can form the basis of a useful checklist when reviewing the presence and adequacy of documentation.

Specify : A fairly abstract level of specification is useful to everybody: "This document models the effect of inflation on sales" for example. Some users need rather more detail than that: a Changer or Developer needs full details of the theoretical model that should be used, so that they can implement it, and a Tester needs those details so that they can see whether it has been implemented. You can't

tell whether a document is doing the right thing unless you know what the right thing is. Even a Viewer or Player may need to know the theory behind the document so that they can interpret the results.

Record : A record of what was done is especially useful to an Auditor, but is also helpful to Changers and Developers, who may need to work out why things are going wrong. However, a record is rarely a good substitute for either explanation or instruction. Recording documentation may be a simple narrative of steps taken, or a more formal record of versions, changes made, reviewers, tests performed, and so on. Information about the sources of data or parameters may count as either recording or explaining documentation.

Explain : Explanations of how the document is put together, or why specific design decisions were made are usually primarily intended for Changers and Developers, but are also useful to Auditors. Explanations of the sources of data or parameters may be useful for all users. A simple narrative of "what I did when building this document" is rarely useful as an explanation, especially as any information it gives may be superseded later on in the narrative. Explanations of the significance of outputs are useful to all users, especially Viewers or Changers who may not have the skills necessary to infer what is going on from the formulae or code.

Instruct : Instructions may be directed at any type of user. They are especially important for Viewers and Players, who may not always be able to infer what should be done from the structure or code in the document itself.

Forms of Documentation : Documentation may take many different forms:

- **A separate document** is often used for long, formal specifications that are themselves subject to review and sign-off. Records of changes and versions may also be kept in a separate document or database. The potential disadvantage of a separate document is that it may be difficult to maintain consistency between the documentation.
- **Implicit** documentation is widely used. The names of worksheets, ranges and cells, and modules and variables in VBA code come into this category. Formatting may also provide documentation, for example if colours are used consistently to indicate which cells are inputs, or to indicate potential errors.
- Documentation **within the code** is perhaps the most common form. It is particularly easy to use in document (compared to other types of software) as it often simply takes the form of text in cells. It is well suited to instructions and some types of explanation, as it can be placed close to the cells to which it refers.
- Documentation as a **separate block** in code, for example as a separate worksheet, can be very useful, especially for keeping records.
- Documentation in the **user interface** overlaps with documentation within the code for document, but is clearly distinct in more conventional software. It includes text in user forms, text boxes and other images.

The most appropriate documentation method depends on the type of documentation and the user for which it is intended, as well as the culture in which it will be used. If your team commonly uses separate documents or centralized systems for specific types of documentation, then you should conform to the common practice.

2.3. Who Benefits from Documentation? :

Just about everybody benefits from clear, accurate documentation. The benefits of out of date documentation written without a specific purpose in mind are less obvious, and indeed are often nonexistent. To get the most out of documentation, the following should be true:

- It should be written specifically as specification, record, explanation or instruction. That way it will be easy for the reader to understand.
- It should be easily available to anybody who wants it; this often (but by no means always) means that it should be part of the document that it applies to.
- It should be kept up to date, otherwise it might mislead the reader.
- When writing documentation, it is often helpful to bear in mind the specific type of user for whom you are writing.

It is not only the reader who benefits from documentation. The writer often gains a lot, too; if the writer is a developer, writing some of the documentation in advance can help to focus the mind and prevent false starts. Articulating your ideas can save you from many dead ends. The documentation process can often throw up bugs in the document or ambiguities in the specification, especially if the writer is a user other than a developer of the document.

Appropriate documentation will help people other than the developer have confidence in the results of a document. They will be to able to tell what the document is intended to do, how it does it, what data it uses, how to use it and interpret the results, and what tests and reviews have been performed.

3. Documentation and Training :

Documentation and training should be undertaken through all phases of the systems development life cycle. The phases of the cycle are systems analysis, systems design, implementation, and operation. The benefits of effective documentation and training throughout the life cycle include obtaining the support of top management, gaining the acceptance of middle-management users, and creating a more durable product.

The systems development life cycle is generally described as four phases of activity: **analysis, design, implementation and operation**. The tasks require the collection, organization, and analysis of information about the old system and the proposed new system. They involve developing new ideas and procedures. Most difficult of all, the design of a new system requires change (which often generates resistance and fear) and an urgent need to communicate the concepts and details of the new system to those who must use it and who are most affected by it.

Documentation and training are an effective means of communicating information. Communication is needed among members of the development team and also between the development team and users who will ultimately accept or reject the new system. Yet, unfortunately the functions of documentation and training, useful when communicating information, are often not considered part of the development cycle and thus are neglected or postponed until after the systems development process has been essentially completed.

Documentation and training are also necessary for the ongoing education of systems support staff and users once the system has entered production. Without this knowledge, maintenance and modification is inefficient and sometimes inadequate, and users are unable to use the production system to its maximum potential As the system ages, important details are forgotten and the full power of the system is lost.

To correct these conditions, training and documentation should not be discrete activities undertaken only at the conclusion of the systems development life cycle. Instead, they must begin at project inception and continue into system operation. Good training and documentation help ensure the support of the system by senior management, its acceptance by user middle management, its use by staff, and ultimately the development and maintenance of a more enduring product

Effective execution of the documentation and training functions requires that these tasks be addressed in all phases of the systems development cycle, not just at the end. Further, it requires assignment of responsibility for these functions to a professional for whom this is a primary responsibility. This documentation coordinator/systems trainer must be a person with strong communications and interpersonal skills who has an understanding of the business problems being addressed, as well as the technology employed in their resolution.

3.1. Documentation and Training During Systems Analysis :

During systems analysis, the coordinator/trainer can increase user awareness and help document business activities, needs, and objectives. Early project success can be ensured by establishing channels of communication with the user community. Communications can increase the understanding of the business issues and problems that will be addressed by the future system.

An effective vehicle for improving communications and increasing project awareness in the user community is the project newsletter. Throughout the project life cycle, the newsletter can keep the broad user community informed of project orientation, decisions, and status. Serving as an "early warning system," the newsletter encourages user input during planning and analysis and provides the following information.

(1) The identity of the project and management

(2) Definition of the problems that will be addressed by the project team

(3) Definition of the systems, files and databases that may be affected

(4)The identity of members of the user community assigned to work on the project.

One of the major tasks in systems analysis is the documentation of existing business and systems activities and problems. Documenting the old system presents the opportunity for the coordinator/trainer to become acquainted with the processes and problems that will be addressed during development.

Experience in the user area also makes the coordinator/trainer a valuable information resource. During systems analysis the coordinator/trainer can develop an understanding of system and business functions which will help define later training needs and audiences.

At the conclusion of systems analysis, the documentation coordinator/trainer can assist in preparing the report to senior management. The report should identify business and systems problems, performance requirements, scope and costs/benefits of development. By including the report, or an abstract of it, in a project newsletter, a larger audience can be introduced to the new system, and the process of "selling" the system can begin.

3.2. Documentation and Training During Systems Design :

Systems design requires decisions effecting the choice of hardware, software and manual resources to be used in developing the new system. Usually at this stage, a choice must be made between buying or building the system. The buy decision involves the coordinator/trainer in the evaluation of vendor documentation and training; the build decision involves the coordinator/trainer in the design of user interfaces to support business functions and the design of the help function and online documentation.

The Buy Decision. The buy decision begins with the evaluation of vendor software. All requests for proposals should include questions related to the availability and quality of documentation and training. Development of these questions is the responsibility of the coordinator/trainer. Questions on training and documentation addressed during the initial selection process can be general and designed to eliminate candidates who are definitely unsuitable. Some questions related to training and documentation follow.

Training: Does vendor provide training? Types of training (classroom, self-study, computer assisted)? At vendor location? Onsite? Is training included in licensing agreement? Number of hours? Number of courses? Number of self-study guides? Interactive training programs? Cost of additional training not included in licensing agreement?

Documentation: Is documentation available? System documentation? Program documentation? User documentation? Cost of additional copies of documentation?

After narrowing the field of candidates, detailed evaluation of training and documentation begins. During detailed evaluation the coordinator/trainer attends

vendor training sessions; performs a review of all training materials and documentation; and participates in software testing.

Attendance at vendor training sessions provides the coordinator/trainer with the opportunity to evaluate the quality of classroom instruction and instructional materials. The systems trainer can determine whether:

(1) Classes are scheduled at intervals and locations that will meet the needs of the user. (2) Instructional materials are interesting, logically organized and address the needs of specific user groups. (3) Qualified instructors are used in all classes. (4) Sufficient time is provided to address problems that may be unique to individual users. (5) Instructors or other qualified individuals are available to answer questions that arise after the conclusion of the class.

The coordinator/trainer and other members of the project team also review supplementary instructional materials, self-study guides and computer assisted instruction. When evaluating the contents of self- study guides, the systems trainer considers whether:

(1) Use of self-study materials is appropriate. (2) Organization of the guides is clear, logical and oriented toward the solution of business problems. (3) Learning objectives are clearly stated and users are frequently tested on their understanding of the materials.

If computer assisted instruction is available, the systems trainer exercises these programs to determine whether:

(1) Programs are fully documented and designed for users with no prior experience with software. (2) The organization and presentation of instructional materials is clear and logical. (3) Users have the ability to terminate, without completing the session, and to return later, picking up where they have left off. (4) Learning is regularly assessed with the system providing additional review and problems, if remedial work is required.

The coordinator/trainer also assesses the usability of documentation. Members of the project team who have used the documentation are an excellent source of information for the coordinator/trainer about the accuracy of technical documentation and its ease of understanding. On the basis of these opinions, the coordinator/trainer provides an overall assessment of its organization, its understandability and ease of use.

On-line documentation and help facilities are also useful. As with any other documentation, the coordinator/trainer should evaluate it considering whether:

(1) Online help and documentation are available from all locations in the system. (2) The system supports more than one level of user help, accommodating both general and specific function documentation. (3) Entry into and exit from the help and documentation functions is quick and easy. (4) Instructions used to access the help function are easy to understand and remember. (5) Information provided in

vendor developed on- line documentation can be understood by the business user. (6) On-line documentation and help are easily modified and maintained.

The Build Decision. A build decision engages the coordinator/trainer more directly in product design. Definition of business functions and supporting business activities drive the design of the system, as well as the design of the training. In this process business activities are associated with the system activities that will be used in their support. Subsequently related system activities can be grouped in logical units and user access menus defined. The coordinator/trainer, working with system designers, can design these user access interfaces. Other responsibilities of the coordinator/ trainer in this phase include the development of user instructions for the new system, and design of on-line help and documentation. A logical, well-organized system that is also user friendly will enhance system use and understanding as well as serve as the basis for user-oriented training.

A common problem associated with the systems analysis phase of project development is the lack of communication with user areas throughout the organization. Many designers tend to work with a limited group of key users, and focus their attention upon the system. This can leave many users without adequate knowledge of the new system or the opportunity to provide potentially valuable input. Again, the importance of the project newsletter cannot be overemphasized. While all users may not be actively engaged in the evaluation or design process, they should be kept abreast of decisions and project status. The newsletter can communicate results and direction. In addition, management information sessions to inform the user community of basic design decisions and address questions and concerns about the project can enhance the exchange of information between user management and systems developers. Information sessions are an effective precursor to formal training, familiarizing the audience with basic design concepts and approaches.

3.3. Documentation and Training During Systems Implementation :

During systems implementation, the coordinator/trainer provides training to audiences in the user community develops or tailors on-line help and documentation functions, and compiles systems, program and user documentation. How each of these tasks is accomplished is largely determined by three factors: the buy or build decision; audience location and size; and type of software. When software is purchased, the vendor frequently provides training classes and materials. If these classes fulfill user requirements, the coordinator/trainer functions primarily as an administrator, scheduling classes and enrolling users in them. On the other hand, if training materials or instruction provided by the vendor proves to be inadequate, the trainer tailors or develops educational materials and conducts in-house training sessions.

Audience size and location and the types of software can drive the decision to supplement vendor training materials and conduct in-house sessions. If the audience is large and training has to be repeated on multiple occasions at multiple sites, using

an in-house trainer to conduct classes results in substantial cost savings. The type of software dictates the degree to which training materials may have to be tailored. Users of the utility software, including Lotus 1-2-3, can be trained primarily in the software functions (often using vendor material), because it is expected that users will recognize the application of these utilities to their specific business problems. On the other hand, users of applications-oriented software, including general ledger, accounts payable, accounts receivable and payroll systems, often require training that is tailored to the support of their unique business activities.

The development of tailored, in-house training for use with a purchased package is a time consuming and complex task. In-house development of training starts with the definition of user groups, a review of their functions and related activities. Working together, designer, trainer and user can map business activities to the purchased system activities that will be used to support them. Definition of training modules can be the result of this mapping process. System activities that support related business activities should be addressed in the same training modules.

The decision to build a system brings with it the commitment to build the training. The building of a system presents the opportunity to structure software and training so that it reflects the system activities.

During design, system activities that support related user business activities should be identified. Subsequently related system activities can be grouped so that they can be accessed through common user menus. The systems trainer, as part of the project team, should be actively engaged in the definition of these menus and their contents. In a well-designed system, user menus can serve as the logical basis for the development of user-oriented training modules.

Training Programs : Typically, three levels of audiences require instruction in the system and its capabilities. Overview sessions are developed for managers in order to acquaint them with the functionality available in the system and how it can be used in their specific areas. Classes for first-line supervisors, which incorporate hands-on instruction, are designed to acquaint supervisors with menus, screens, help functions, user documentation and user controls. The third level of instruction focuses upon the needs of the direct user audience. These sessions are hands-on and employ "real life examples" throughout the class. This level should provide the participants with the opportunity to exercise the system and understand how it will be used in solving their business problems. A hands-on approach to instruction encourages system use in the controlled environment of the classroom. To enable all users to attend training, multiple small sections of the same class should be scheduled. This flexible scheduling enables all system users to attend class without disrupting their personal schedules or work activities. Training should occur close to the time when the system becomes available in the user area, because practice on the system helps reinforce procedures learned in class.

The project newsletter plays an important role in the training program. The newsletter can publish descriptions and schedules for user classes, and detailed

enrollment procedures. The newsletter can also serve as a training vehicle. By reporting on project progress and major project-related decisions, it describes system functionality to a broad audience. Although it may be difficult for user management to attend traditional classes, they can become acquainted with system functions through well-chosen "articles" presented in the project newsletter.

Whether the system is built or bought, the coordinator/trainer may be called upon to develop user-oriented on-line help and documentation functions. The on-line help function supplements training information and is a substitute for additional documentation.

Information, supplied in the on-line help function, should be expressed in terms that the user can understand, and should support information requirements at different levels of detail.

During systems implementation, the coordinator/trainer also compiles documentation. With purchased software, this can entail the establishment of a documentation library to house master copies of systems, program and operations documentation. If the software is modified in any way, fully documented changes should be added to the library. Software written in-house imposes additional demands for the development of documentation. In most large companies, documentation standards have been defined. It is the responsibility of the coordinator/trainer to see that these standards are met and that documentation for these systems is complete properly organized and maintained.

3.4. Documentation & Training Considerations During Sys. Operations:

Adaptation of the new systems environment often brings the reassignment of the coordinator/trainer. All too often, this transfer means that formalized training in the system has ended and that documentation, although now complete, will not be maintained. The department that is designated as the "owner" of the system should assume these responsibilities. To ensure a smooth transition to the new "owner" area, the coordinator/trainer must provide for a continuation of this function. This means identifying individuals in the owner area who will be available to maintain documentation and to continue classes, as necessary, over the life of the system. The new coordinators/trainers must become familiar with all instructional materials and system operations. If there is constant turnover in the user departments, classes in system use will be required at regular intervals. The "owner" department coordinator/trainer must provide class instruction and maintain instructional materials and user documentation. Maintenance of other documentation, such as systems, program and operations manuals, is done by the systems maintenance area. Regular audits of the user area and system maintenance function help ensure the integrity of production system instruction, documentation and maintenance procedures.

4. Operations and Maintenance Phase :

More than half of the life cycle costs are attributed to the operations and maintenance of systems. In this phase, it is essential that all facets of operations and

maintenance are performed. The system is being used and scrutinized to ensure that it meets the needs initially stated in the planning phase. Problems are detected and new needs arise. This may require modification to existing code, new code to be developed, and/or hardware configuration changes. Providing user support is an ongoing activity. New users will require training and others will require training as well. The emphasis of this phase will be to ensure that the users needs are met and the system continues to perform as specified in the operational environment. Additionally, as operations and maintenance personnel monitor the current system they may become aware of better ways to improve the system and therefore make recommendations. Changes will be required to fix problems, possibly add features, and make improvements to the system. This phase will continue as long as the system is in use.

4.1. Tasks and Activities :

Systems Operations : Operations support is an integral part of the day-to-day operations of a system. In small systems, all or part of each task may be done by the same person. But in large systems, each function may be done by separate individuals or even separate areas. The Operations Manual was developed in previous SDLC phases. This document defines tasks, activities, and responsible parties and will need to be updated as changes occur.

Systems operations activities and tasks need to be scheduled, on a recurring basis, to ensure that the production environment is fully functional and is performing as specified. The following is a checklist of systems operations key tasks and activities:

- Ensure that systems and networks are running and available during the defined hours of Operations.
- Implement non-emergency requests during scheduled Outages, as prescribed in the Operations Manual.
- Ensure all processes, manual and automated, are documented in the operating procedures. These processes should comply with the system documentation.
- Acquisition and storage of supplies, e.g., paper, toner, tapes, removable disk.
- Perform backups (day-to-day protection, contingency).
- Perform the physical security functions including ensuring adequate UPS, Personnel have proper clearances and proper access privileges etc.
- Ensure contingency planning for disaster recovery is current and tested.
- Ensure users are trained on current processes and new processes.
- Ensure that service level objectives are kept accurate and are monitored.
- Maintain performance measurements, statistics, and system logs. Examples of performance measures include volume and frequency of data to be processed in each mode, order and type of operations.

• Monitor the performance statistics, report the results, and escalate problems when they occur.

Data / Software Administration : Data / Software Administration is needed to ensure that input data and output data and databases are correct and continually checked for accuracy and completeness. This includes insuring that any regularly scheduled jobs are submitted and completed correctly. Software and databases should be maintained at (or near) the current maintenance level. The backup and recovery processes for databases are normally different than the day-to-day DASD volume backups. The backup and recovery process of the databases should be done as a Data / Software Administration task. A checklist of Data / Software Administration tasks and activities are:

• Performing production control and quality control functions (Job submission, checking and corrections).

• Interfacing with other functional areas for day-to-day checking / corrections. Installing, configuring, upgrading and maintaining database(s). This includes updating processes, data flows, and objects (usually shown in diagrams).

• Developing and performing data / database backup and recovery routines for data integrity and recoverability. Ensure documented properly in the Operations Manual.

• Developing and maintaining a performance and tuning plan for online process and databases.

• Performing configuration and design audits to ensure software, system, parameter, and configuration are correct.

Problem and Modification Process : One fact of life with any system is that change is inevitable. Users need an avenue to suggest change and identified problems. A User Satisfaction Review which can include a Customer Satisfaction Survey can be designed and distributed to obtain feedback on operational systems to help determine if the systems are accurate and reliable. Systems administrators and operators need to be able to make recommendations for upgrade of hardware, architecture and streamlining processes. For small in-house systems, modification requests can be handled by an in-house process. For large integrated systems, modification requests may be addressed in the Requirements document and may take the form of a change package or a formal Change Implementation Notice and may require justification and cost benefits analysis for approval by a review board. The Requirements document for the project may call for a modification cut-off and rollout of the system as a first version and all subsequent changes addressed as a new or enhanced version of the system. A request for modifications to a system may also generate a new project and require a new project initiation plan.

System / Software Maintenance : Daily operations of the system /software may necessitate that maintenance personnel identify potential modifications needed

to ensure that the system continues to operate as intended and produces quality data. Daily maintenance activities for the system, takes place to ensure that any previously undetected errors are fixed. Maintenance personnel may determine that modifications to the system and databases are needed to resolve errors or performance problems. Also modifications may be needed to provide new capabilities or to take advantage of hardware upgrades or new releases of system software and application software used to operate the system. New capabilities may take the form of routine maintenance or may constitute enhancements to the system or database as a response to user requests for new/improved capabilities. New capabilities needs may begin a new problem modification process described above. At this phase of the SDLC all security activities have been at least initiated or completed. An update must be made to the System Security plan; an update and test of the contingency plan should be completed. Continuous vigilance should be given to virus and intruder detection. The Project Manager must be sure that security operating procedures are kept updated accordingly.

Review Previous Documentation : Review and update documentation from the previous phases. In particular, the Operations Manual, System Boundary Document, and Contingency Plan need to be updated and finalized during the Operations and Maintenance Phase as required.

4.2. Roles and Responsibilities :

This list briefly outlines some of the roles and responsibilities for key maintenance personnel. Some roles may be combined or eliminated depending upon the size of the system to be maintained. Each system will dictate the necessity for the roles listed below.

System Manager: The System Manager develops, documents and execute plans and procedures for conducting activities and tasks of the Maintenance Process. To provide for an avenue of problem reporting and customer satisfaction, the Systems Manager should create and discuss communications instructions with the systems customers. Systems Managers should keep the Help Desk Personnel informed of all changes to the system especially those requiring new instructions to users.

Technical Support: Personnel which provide technical support to the program. This support may involve granting access rights to the program. Setup of workstations or terminals to access the system. Maintaining the operating system for both server and workstation. Technical support personnel may be involved with issuing user IDs or login names and passwords. In a Client server environment technical support may perform systems scheduled backups and operating system maintenance during downtime.

Vendor Support: The technical support and maintenance on some programs are provided through vendor support. A contract is established outlining the contracted systems administration, operators, and maintenance personnel duties and responsibilities.

One responsibility which should be included in the contract is that all changes to the system will be thoroughly documented.

Help Desk: Help Desk personnel provide the day-to-day users help for the system. Help desk personnel should be kept informed of all changes or modifications to the system. Help Desk Personnel are contacted by the user when questions or problems occur with the daily operations of the system. Help Desk Personnel need to maintain a level of proficiency with the system.

Operations or Operators (turn on/off systems, start tasks, backup etc): For many mainframe systems, an operator provides technical support for a program. The operator performs scheduled backup, performs maintenance during downtime and is responsible to ensure the system is online and available for users. Operators may be involved with issuing user IDs or login names and passwords for the system.

Customers: The customer needs to be able to share with the systems manager the need for improvements or the existence of problems. Some users live with a situation or problem because they feel they must. Customers may feel that change will be slow or disruptive. Some feel the need to create work-arounds. A customer has the responsibility to report problems or make recommendations for changes to a system.

Program Analysts or Programmer: Interprets user requirements, designs and writes the code for specialized programs. User changes, improvements, enhancements may be discussed in Joint Application Design sessions. Analysts programs for errors, debugs the program and tests program design.

Process Improvement Review Board: A board of individuals may be convened to approve recommendations for changes and improvements to the system. This group may be chartered. The charter should outline what should be brought before the group for consideration and approval. The board may issue a Change Directive.

Users Group or Team: A group of computer users who share knowledge they have gained concerning a program or system. They usually meet to exchange information, share programs and can provide expert knowledge for a system under consideration for change.

Contract Manager: The Contract Manager has many responsibilities when a contract has been awarded for maintenance of a program. The Contract Manager should have a certificate of training for completion of a Contracting Officer's Technical Representative (COTR) course. The Contract Manager's main role is to make sure that the interests of the Procurement Office are protected and that no modifications are made to the contract without permission from the Procurement Office.

Data Administrator: Performs tasks which ensure that accurate and valid data are entered into the system. Sometimes this person creates the information systems database, maintains the databases security and develops plans for disaster recovery. The data administrator may be called upon to create queries and reports for a variety of user requests. The data administrator responsibilities include maintaining the databases data dictionary. The data dictionary provides a description of each field in the database, the field characteristics and what data is maintained with the field.

Telecommunications Analyst and Network System Analyst: Plans, installs, configures, upgrades and maintains networks as needed. If the system requires it, they ensure that external communications and connectivity are available.

Computer Systems Security Officer (CSSO): The CSSO has a requirement to review system change requests, review and in some cases coordinate the Change Impact Assessments, participate in the Configuration Control Board process, and conduct and report changes that may be made that effect the security posture of the system.

4.3. Deliverables, Responsibilities and Action

In-Process Review : The In-Process Review occurs at predetermined milestones usually quarterly, but at least once a year. The performance measure should be reviewed along with the health of the system. Performance measures should be measured against the baseline measures. Ad- hoc reviews should be called when deemed necessary by either party.

User Satisfaction Review : User Satisfaction Reviews can be used as a tool to determine the need to proceed with a Process Improvement Review Board meeting or initiate a proposal for a new system. This review can be used as input to the In-Process Review.

4.4. Issues for Consideration :

Documentation : It cannot be stressed enough, that proper documentation for the duties performed by each individual responsible for system maintenance and operation should be up-to-date. For smooth day-to-day operations of any system, as well as disaster recovery, each individual's role, duties and responsibilities should be outlined in detail. A systems administrator's journal or log of changes performed to the system software or hardware is invaluable in times of emergencies. Operations manuals, journals or logs should be readily accessible by maintenance personnel.

Guidelines in Determining New Development from Maintenance : Changes to the system should meet the following criteria in order for the change or modification request to be categorized as Maintenance; otherwise it should be considered as New Development:

- Estimated cost of modification are below maintenance costs
- Proposed changes can be implemented within 1 system year
- Impact to system is minimal or necessary for accuracy of system output

4.5. Review Activity

Review activities occur several times throughout this phase. Each time the system is reviewed, one of three of the following decisions will be made:

- The system is operating as intended and meeting performance expectations.
- The system is not operating as intended and needs corrections or modifications.
- The users are/are not satisfied with the operation and performance of the system.

The In-Process Review (conducted at least annually) shall be conducted in this phase. The In-Process Review shall be performed to evaluate system performance, user satisfaction with the system, adaptability to changing business needs, and new technologies that might improve the system. This review is diagnostic in nature and can lead to development or maintenance activities. Any major system modifications needed after the system has been implemented will follow the life cycle process from planning through implementation. A project management plan, including a feasibility study, will identify modifications to existing system documentation (change pages) rather than new system documentation (for example, a functional requirements document, a system design document, etc.). The appropriate reviews and testing will be conducted, based on the scope of the modification.

Questions

Very short Questions:

1. What is documentation ?
2. What is system implementation ?
3. What do you mean by review activity ?
4. What is the use of Training ?
5. Who will get benifit from documentation ?

Short Questions:

1. What are the roles and responsibilities during Implementation phase ?
2. Describe the use of documentation ?
3. What are the activities performed during operations phase ?
4. Describe various types of training ?
5. What are the various task performed in Maintanence phase ?

Long Question :

1. What do you mean by implementation ? Discuss it's all task and activities ?
2. What do you mean by Documentation ? Discuss it is all forms a benefits ?
3. How you relate documentation a Training during Systems Design ?
4. What do you Mean by maintenance ? Discuss it's all task and activities.
5. Documentation, what role play in System Implementation ?

❑❑❑

CbH