Paper Name: Software Engineering

Topic: Software Testing

Paper Code: BCA CC409

Semester IV

By Ms. Amrita Prakash

Assistant Professor,

Department of Computer Science

Patna Women's College

E-mail: amrita.bca@patnawomenscollege.in

Software testing is an essential part of the software development process, which is used to identify the correctness, completeness and quality of the developed software.

Its main objective is to detect errors in the software. Errors occur when any aspect of a software product is incomplete, inconsistent, or incorrect. Errors can broadly be classified into three types, namely,

1. **Requirements errors**
2. **Design errors and**
3. **Programming errors**

Software testing involves activities aimed at evaluating an attribute and capability of a program or a system and ensuring that it meets its required results.

IEEE defines Software testing as "the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results."

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Verification: Are we developing the software right?

Validation: Are we developing the right software?

Software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
- Validation emphasizes on user requirements.

Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
- Verifications concentrates on the design and system specifications.

Target of the test are -

- **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
- **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

Errors can be present in the software due to the following reasons:-

- Programming Errors
- Unclear Requirements
- Software complexity
- Changing Requirements
- Time Pressures
- Poorly documented code

Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

- **Manual** - This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

  Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

- **Automated** This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

Guidelines for Software Testing

- Define the expected output
- Inspect Output of each test completely
- Include test cases for Invalid and Unexpected Conditions
- Test the modified program to check its Expected Performance

Testability

The ease with which a program is tested is known as testability. It can also be defined as the degree to which a program facilitates the establishment of test criteria and execution of tests to determine whether the criteria have been met or not.

Characteristics of Testability

- **Easy to operate :** High quality software can be tested in a better manner.

- **Observability:** Testers can easily identify whether the output generated for certain input is accurate or not simply by observing it.

- **Decomposability:** By breaking software into independent modules problems can be easily isolated and the models can be easily tested.

- **Stability:** Software becomes stable when changes made to the software are controlled and when the existing tests can still be performed.

- **Easy to understand:** Software that is easy to understand can be tested in an efficient manner software can be properly understood by gathering maximum information about it for example to have a proper knowledge of the software its documentation can be used which provides complete information of software food thereby increasing its clarity and making testing easier.

Characteristics of Software Test

- **High probability of detecting errors:** To detect maximum errors, the tester should understand the software thoroughly and try to find the possible ways in which the software can fail.

- **No Redundancy:** Resources and testing time are limited in software development process. Thus, it is not beneficial to develop several tests, which have the same intended purpose. Every test should have a distinct purpose

- **Choose the most appropriate test:** There can be different tests that have the same intent but due to certain limitations, such as time and resource constraint, only few of them are used.

- **Moderate:** Test is considered good if it is neither too simple nor too complex. Many tests can be combined to form one test case.

Test Plan

A test plan describes how testing would be accomplished. A test plan is defined as a document that describes the objectives, scope, method, and purpose of software testing. This plan identifies test items, features to be tested, testing tasks and the persons involved in performing these tasks. It also identifies the test environment and the test design and measurement techniques that are to be used. Note that a properly defined test plan is an agreement between testers and users describing the role of testing in software.

A complete test plan helps people outside the test group to understand the 'why' and 'how' of product validation. Whereas an incomplete test plan can result in a failure to check how the software works on different hardware and operating systems or when software is used with other software. To avoid this problem, IEEE states some components that should be covered in a test plan. These components are listed in Table 5.3.

Table 5.3 Test Plan Components

Component	Purpose
Responsibilities	Assigns responsibilities and keeps people on track and focused.
Assumptions	Avoids misunderstandings about schedules.
Test	Outlines the entire process and maps specific tests. The testing scope, schedule, and duration are also outlined.
Communication	Communication plan (who, what, when, how about the people) is developed.
Risk Analysis	Identifies areas that are critical for success.
Defect Reporting	Specifies how to document a defect so that it can be reproduced, fixed, and retested.
Environment	Specifies the technical environment, data, work area, and interfaces used in testing. This reduces or eliminates misunderstandings and sources of potential delay.

Steps in Development of Test Plan: A carefully developed test plan facilitates effective test execution, proper analysis of errors, and preparation of error report.

To develop a test plan, a number of steps are followed, which are listed below:

1. Set objectives of test plan: Before developing a test plan, it is necessary to understand its purpose. The objectives of a test plan depend on the objectives of software. For example, if the objective of software is to accomplish all user requirements, then a test plan is generated to meet this objective. Thus, it is necessary to determine the objective of software before identifying the objective of test plan.

2. Develop a test matrix: Test matrix indicates the components of software that are to be tested. It also specifies the tests required to test these components. Test matrix is also used as a test proof to show that a test exists for all components of software that require testing. In addition, test matrix is used to indicate the testing method which is used to test the entire software.

3. Develop test administrative component: It is necessary to prepare a test plan within a fixed time so that software testing can begin as soon as possible. The test administrative component of test plan specifies the time schedule and resources (administrative people involved while developing the test plan) required to execute the test plan. However, if implementation plan (a plan that describes how the processes in software are carried out) of software changes, the test plan also changes. In this case, the schedule to execute the test plan also gets affected.

4. Write the test plan: The components of test plan, such as its objectives, test matrix, and administrative component are documented. All these documents are

then collected together to form a complete test plan. These documents are organised either in an informal or formal manner. In informal manner, all the documents are collected and kept together. The testers read all the documents to extract information required for testing software. On the other hand, in formal manner, the important points are extracted from the documents and kept together. This makes it easy for testers to extract important information, which they require during software testing.

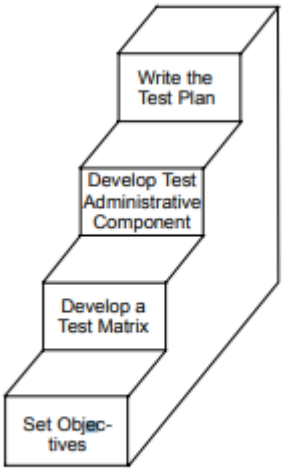

Figure 5.4 Steps in Test Plan

A test plan is shown in Figure 5.5. This plan has many sections, which are listed below:

1.0 Overview
- 1.1 Project Objectives
- 1.2 System Description
- 1.3 Plan Objectives
- 1.4 References
- 1.5 Issues, Assumptions

2.0 Test Scope
- 2.1 Features to be tested
- 2.2 Features not to be tested

3.0 Test Methodologies
- 3.1 Testing Approach
- 3.2 Test Data
- 3.3 Test Documents
- 3.4 Requirements Validation
- 3.5 Control Procedures

4.0 Test Phases
- 4.1 Definition
- 4.2 Participants
- 4.3 Source of Data
- 4.4 Entrance and Exit Criteria
- 4.5 Requirements
- 4.6 Work Products
- 4.7 Test Completion Acceptance

5.0 Test Environment
- 5.1 Hardware
- 5.2 Software
- 5.3 Location
- 5.4 Staffing and Training

6.0 Schedule

7.0 Approvals and Distribution

Figure 5.5 Test Plan

•**Overview**: Describes the objectives and functions of the software to be performed. It also describes the objectives of test plan, such as defining responsibilities, identifying test environment and giving a complete detail of the sources from where the information is gathered to develop the test plan.

• **Test scope:** Specifies features and combination of features, which are to be tested. These features may include user manuals or system documents. It also specifies the features and their combinations that are not to be tested.

• **Test methodologies:** Specifies types of tests required for testing features and combination of these features, such as regression tests and stress tests. It also provides description of sources of test data along with how test data is useful to ensure that testing is adequate, such as selection of boundary or null values. In addition, it describes the procedure for identifying and recording test results.

• **Test phases:** Identifies various kinds of tests, such as unit testing, integration testing and provides a brief description of the process used to perform these tests. Moreover, it identifies the testers that are responsible for performing testing and provides a detailed description of the source and type of data to be used. It also describes the procedure of evaluating test results and describes the work products, which are initiated or completed in this phase.

• **Test environment:** Identifies the hardware, software, automated testing tools, operating system, compliers, and sites required to perform testing. It also identifies the staffing and training needs.

• **Schedule**: Provides detailed schedule of testing activities and defines the responsibilities to respective people. In addition, it indicates dependencies of testing activities and the time frames for them.

Test Case Design

A test case is a document that describes an input, action, or event and its expected result, in order to determine whether the software or a part of the software is working correctly or not. IEEE defines test case as “a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement”.

Incomplete and incorrect test cases lead to incorrect and erroneous test outputs. To avoid this, a test case should be developed in such a manner that it checks software with all possible inputs. This process is known as exhaustive testing

and the test case, which is able to perform exhaustive testing, is known as ideal test case

Generally, a test case is unable to perform exhaustive testing therefore, a test case that gives satisfactory results is selected. In order to select a test case, certain questions should be addressed.

• How to select a test case?

• On what basis certain elements of program are included or excluded from a test case?

For a given program and its specifications, a test selection criterion specifies the conditions that should be satisfied by a set of test cases.

Test Case Generation

The process of generating test cases helps in locating problems in the requirements or design of software. To generate a test case, initially a criterion that evaluates a set of test cases is specified. Then, a set of test cases that satisfy the specified criterion is generated. There are two methods used to generate test cases, which are listed below:

• **Code based test case generation:** This approach, also known as structure based test case generation is used to analyse the entire software code to generate test cases. It considers only the actual software code to generate test cases and is not concerned with the user requirements. Test cases developed using this approach are generally used for unit testing. These test cases can easily test statements, branches, special values, and symbols present in the unit being tested.

• **Specification based test case generation**: This approach uses specifications, which indicate the functions that are produced by software to generate test cases. In other words, it considers only the external view of software to generate test cases. Specification based test case generation is generally used for integration testing and system testing to ensure that software is performing the required task. Since this approach considers only the external view of the software, it does not test the design decisions and may not cover all statements of a program. Moreover, as test cases are derived from specifications, the errors present in these specifications may remain uncovered.

Several tools known as test case generators are used for generating test cases. In addition to test case generation, these tools specify the components of software that are to be tested. An example of test case generator is ‘astra quick test’,

which captures business processes in the visual map and generates data driven tests automatically

Software Testing Strategies

Software testing strategies can be considered as various levels of testing that are performed to test the software. The first level starts with testing of individual units of software. Once the individual units are tested, they are integrated and checked for interfaces established between them. After this, entire software is tested to ensure that the output produced is according to user requirements.

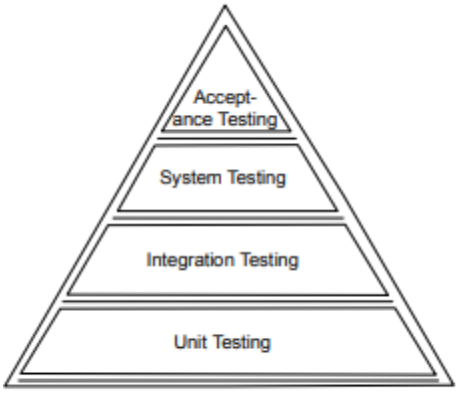

Figure 5.6 Levels of Software Testing

As shown in Figure 5.6, there are four levels of software testing, namely, unit testing, integration testing, system testing, and acceptance testing.

Unit Testing

Unit testing is performed to test the individual units of software. Since software is made of a number of units/modules, detecting errors in these units is simple and consumes less time, as they are small in size. However, it is possible that the outputs produced by one unit become input for another unit. Hence, if incorrect output produced by one unit is provided as input to the second unit, then it also produces wrong output. If this process is not corrected, the entire software may produce unexpected outputs.

To avoid this, all the units in software are tested independently using unit testing. Unit level testing is not just performed once during the software development, rather it is repeated whenever software is modified or used in a new environment.

The other points noted about unit testing are listed below:

• Each unit is tested in isolation from other parts of a program.

• The developers themselves perform unit testing.

• Unit testing makes use of white box testing methods.

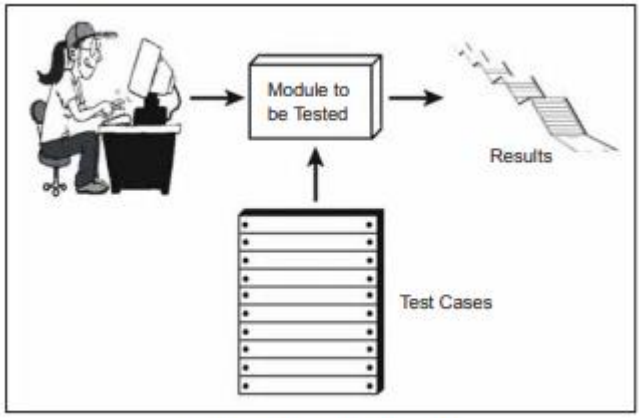

Figure 5.7 Unit Testing

Unit testing is used to verify the code produced during software coding and is responsible for assessing the correctness of a particular unit of source code.

In addition, unit testing performs the functions listed below:

• Tests all control paths to uncover maximum errors that occur during the execution of conditions present in the unit being tested.

• Ensures that all statements in the unit are executed at least once.

• Tests data structures (like stacks, queues) that represent relationships among individual data elements.

• Checks the range of inputs given to units. This is because every input range has a maximum and minimum value and the input given should be within the range of these values.

• Ensures that the data entered in variables is of the same data type as defined in the unit.

• Checks all arithmetic calculations present in the unit with all possible combinations of input values.

Integration testing

Once unit testing is complete, integration testing begins. In integration testing, the units validated during unit testing are combined to form a sub system.

The purpose of integration testing is to ensure that all the modules continue to work in accordance with user/customer requirements even after integration.

The objective of integration testing is to take all the tested individual modules, integrate them, test them again, and develop the software, which is according to design specifications.

The other points that are noted about integration testing are listed below:

• Integration testing ensures that all modules work together properly, are called correctly, and transfer accurate data across their interfaces.

• Testing is performed with an intention to expose defects in the interfaces and in the interactions between integrated components or systems.

• Integration testing examines the components that are new, changed, affected by a change, or needed to form a complete system.

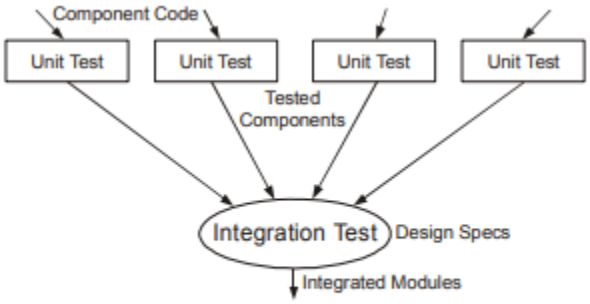

Figure 5.10 Integration of Individual Modules

Regression Testing

Regression testing 're-tests' the software or part of it to ensure that no previously working components, functions, or features fail as a result of the error correction process and integration of modules.

Regression testing is considered an expensive but a necessary activity since it is performed on modified software to provide knowledge that changes do not adversely affect other system components.

Thus, regression testing can be viewed as a quality control tool that ensures that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the change. For instance, suppose a new function is added to the software, or a module is modified to improve its response time.

The changes may introduce errors into the software that was previously correct. For example, suppose part of the code written below works properly.

```
x = b + 1 ;

proc (z) ;

b = x + 2 ;

x = 3;
```

Now suppose in an attempt to optimise the code it is transformed into the following:

```
proc (z) ;

b = b + 3 ;

x = 3 ;
```

This may result in an error if procedure 'proc' accesses variable 'x'.

Thus, testing should be organised with the purpose of verifying possible degradations of correctness or other qualities due to later modifications.

During regression testing, existing test cases are executed on the modified software so that errors can be detected.

Test cases for regression testing consist of three different types of tests, which are listed below:

- Tests that are used to execute software function.
- Tests that check the function, which is likely to be affected by changes.
- Tests that check software modules that have already been modified. The various advantages and disadvantages associated with regression testing are listed in Table 5.6.

Table 5.6 Advantages and Disadvantages of Regression Testing

Advantages	Disadvantages
▪ Ensures that the unchanged parts of software work properly. ▪ Ensures that all errors that have occurred in software due to modifications are corrected and are not affecting the working of software.	▪ Time consuming activity. ▪ Considered to be expensive.

System Testing

Software is integrated with other elements, such as hardware, people, and database to form a computer-based system. This system is then checked for errors using system testing. IEEE defines system testing as "a testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements".

System testing compares the system with the non-functional system requirements, such as security, speed, accuracy, and reliability. The emphasis is on validating and verifying the functional design specifications and examining how modules work together. This testing also evaluates external interfaces to other applications and utilities or the operating environment.

Figure 5.16 Types of System Testing

During system testing, associations between objects (like fields), control and infrastructure (like time management, error handling), feature interactions or problems that occur when multiple features are used simultaneously and compatibility between previously working software releases and new releases are tested. System testing also tests some properties of the developed software, which are essential for users.

These properties are listed below:

• Usable: Verifies that developed software is easy to use and is understandable.

• Secure: Verifies that access to important or sensitive data is restricted even for those individuals who have authority to use software.

• Compatible: Verifies that developed software works correctly in conjunction with existing data, software and procedures.

• Documented: Verifies that manuals that give information about developed software are complete, accurate and understandable.

• Recoverable: Verifies that there are adequate methods for recovery in case of failure

When functionality is being tested without taking the actual implementation in concern it is known as black-box testing. The other side is known as white-box testing where not only functionality is tested but the way it is implemented is also analyzed.

Exhaustive tests are the best-desired method for a perfect testing. Every single possible value in the range of the input and output values is tested. It is not possible to test each and every value in real world scenario if the range of values is large.

Black-box testing

Black box testing, also known as functional testing, checks the functional requirements and examines the input and output data of these requirements. The functionality is determined by observing the outputs to corresponding inputs. For example, when black box testing is used, the tester should only know the 'legal' inputs and what the expected outputs should be, but not how the program actually arrives at those outputs.

It is carried out to test functionality of the program. It is also called 'Behavioral' testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok', and problematic otherwise.

In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

The black box testing is used to find errors listed below:

• Interface errors, such as functions, which are unable to send or receive data to/from other software.

• Incorrect functions that lead to undesired output when executed.

• Missing functions and erroneous data structures.

• Erroneous databases, which lead to incorrect outputs when software uses the data present in these databases for processing.

• Incorrect conditions due to which the functions produce incorrect outputs when they are executed.

• Termination errors, such as certain conditions due to which function enters a loop that forces it to execute indefinitely.

In this testing, various inputs are exercised and the outputs are compared against specification to validate the correctness. Note that test cases are derived from these specifications without considering implementation details of the code. The outputs are compared with user requirements and if they are as specified by the user, then the software is considered to be correct, else the software is tested for the presence of errors in it.

Black-box testing techniques:

- **Equivalence class** - The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.
- **Boundary values** - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
- **Cause-effect graphing** - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.
- **Pair-wise Testing** - The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair-wise for their different values.
- **State-based testing** - The system changes state on provision of input. These systems are tested based on their states and input.

Table 5.9 Advantages and Disadvantages of Black Box Testing.

Advantages	Disadvantages
▪ Tester requires no knowledge of implementation and programming language used.	▪ Only small number of possible inputs can be tested as testing every possible input consumes a lot of time.
▪ Reveals any ambiguities and inconsistencies in the functional specifications.	▪ There can be unnecessary repetition of test inputs if the tester is not informed about the test cases that software developer has already tried.
▪ Efficient when used on larger systems.	▪ Leaves many program paths untested.
▪ Non-technical person can also perform black box testing.	▪ Cannot be directed towards specific segments of code, hence is more error prone.

White-box testing

White box testing, also known as structural testing is performed to check the internal structure of a program.

To perform white box testing, tester should have a thorough knowledge of the program code and the purpose for which it is developed.

The basic strength of this testing is that the entire software implementation is included while testing is performed. This facilitates error detection even when the software specification is vague or incomplete.

The objective of white box testing is to ensure that the test cases (developed by software testers by using white box testing) exercise each path through a program.

That is, test cases ensure that all internal structures in the program are developed according to design specifications.

The test cases also ensure that:

• All independent paths within the program have been executed at least once.

• All internal data structures are exercised to ensure validity.

• All loops (simple loops, concatenated loops, and nested loops) are executed at their boundaries and within operational bounds.

• All the segments present between the control structures (like ‘switch’ statement) are executed at least once.

• Each branch (like ‘case’ statement) is exercised at least once.

• All the branches of the conditions and the combinations of these conditions are executed at least once. Note that for testing all the possible combinations, a ‘truth table’ is used where all logical decisions are exercised for both true and false paths.

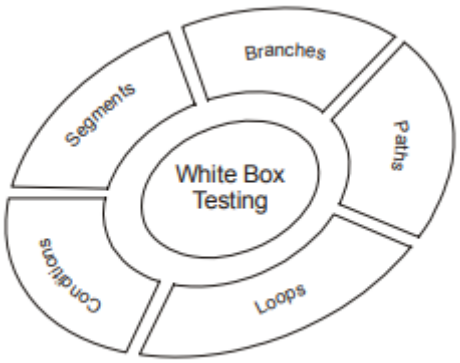

Figure 5.20 White Box Testing

The below are some White-box testing techniques:

- **Control-flow testing** - The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.
- **Data-flow testing** - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Table 5.8 Advantages and Disadvantages of White Box Testing

Advantages	Disadvantages
▪ Covers the larger part of the program code while testing.	▪ Tests that cover most of the program code may not be good for assessing the functionality of surprise (unexpected) behaviours and other testing goals.
▪ Uncovers typographical errors.	▪ Tests based on design may miss other system problems.
▪ Detects design errors that occur when incorrect assumptions are made about execution paths.	▪ Tests cases need to be changed if implementation changes.

The effectiveness of white box testing is commonly expressed in terms of test or code coverage metrics, which measure the fraction of code exercised by test cases. The various types of testing, which occur as part of white box testing are *basis path testing*, *control structure testing*, and *mutation testing*.

Testing vs. Quality Control, Quality Assurance and Audit

We need to understand that software testing is different from software quality assurance, software quality control and software auditing.

- **Software quality assurance** - These are software development process monitoring means, by which it is assured that all the measures are taken as per the standards of organization. This monitoring is done to make sure that proper software development methods were followed.
- **Software quality control** - This is a system to maintain the quality of software product. It may include functional and non-functional aspects of software product, which enhance the goodwill of the organization. This system makes sure that the customer is receiving quality product for their requirement and the product certified as 'fit for use'.
- **Software audit** - This is a review of procedure used by the organization to develop the software. A team of auditors, independent of development team examines the software process, procedure, requirements and other aspects of SDLC. The purpose of software audit is to check that software and its development process, both conform standards, rules and regulations.