UNIT 6: SOFTWARE TESTING

6.1 **SOFTWARE TESTING PROCESS:**

- Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and code generation.
- Testing involves exercising the program using data like the real data processed by unexpected system outputs.
- Testing may be carried out during the implementation phase to verify that the software behaves as intended by its designer and after the implementation is complete. This later phase checks conformance with requirements is complete.
- Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not. This activity results in the actual, expected and difference between their results. In simple words testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements.
- According to ANSI/IEEE 1059 standard, Testing can be defined as *A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item*.

Who does testing?

- It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in the context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, following professionals are involved in testing of a system within their respective capacities:
  - Software Tester
  - Software Developer

    - Project Lead/Manager
    - End User

- Different companies have difference designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, and QA Analyst etc.

- It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

When to Start Testing?

- An early start to testing reduces the cost, time to rework and error free software that is delivered to the client. However in Software Development Life Cycle (SDLC) testing can be started from the Requirements Gathering phase and lasts till the deployment of the software. However it also depends on the development model that is being used. For example in Water fall model formal testing is conducted in the Testing phase, but in incremental model, testing is performed at the end of every increment/iteration and at the end the whole application is tested.

- Testing is done in different forms at every phase of SDLC like during Requirement gathering phase, the analysis and verifications of requirements are also considered testing. Reviewing the design in the design phase with intent to improve the design is also considered as testing. Testing performed by a developer on completion of the code is also categorized as Unit type of testing.

When to Stop Testing?

- Unlike when to start testing it is difficult to determine when to stop testing, as testing is a never ending process and no one can say that any software is 100% tested. Following are the aspects which should be considered to stop the testing:
    - Testing Deadlines.

    - Completion of test case execution.
    - Completion of Functional and code coverage to a certain point.
    - Bug rate falls below a certain level and no high priority bugs are identified.

- Different kinds of testing use different types of data:
  - Statistical testing may be used to test the programs performance and reliability
  - Defect testing is intended to find the areas where the program does not conform to its specification

Testing Vs Debugging:

- Defect testing a debugging are sometimes considered to be part of the same process. In fact they are quite different. Testing establishes the existence of defects. Debugging usually follows testing but they differs as to goals, methods and psychology

  1. Testing starts with unknown conditions, uses predefined procedures, and has predictable outcomes only whether or not the program passes the test id unpredictable.

  2. Testing can and should be planned designed and scheduled the procedures for and duration of debugging cannot be so constrained.

  3. Testing is a demonstration of error or apparent correctness

  4. Testing proves a programmers failure. Debugging is the programmer's vindication (justification for some act or belief)

  5. Testing as executed hold strives to predictable, dull, constrained, rigid and inhuman.

  6. Much of the testing can be done without design knowledge.

  7. Testing can often be done by an outsider. Debugging must be done by an insider.

  8. Much of test execution and design can be automated. Automated debugging is still a dream.

Testing Objectives:

1. Testing is a process of executing a program with the intend of finding on error.

2. A good test case is one that high probability of finding an as yet undiscovered error.

3. A successful test is one that uncovers an as yet undiscovered error.

6.2 Testing Principles:

The various testing principle a listed below:

1. All tests should be traceable to customer requirements. The most serve defects are those that cause the program fail to meet its requirements.

2. Test should be planned long before testing begins. All tests can be planned and designed before any code has been generated.

3. Testing should begin “in the small” and progress towards testing “in the large”. The first tests planned and executed generally focusing on the individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

4. Exhaustive testing is not possible.

5. To be more effective testing should be conducted by a third party.

Attributes of a good testing:

1. A good testing has a high probability of finding an error.

2. A good test is not redundant.

3. In a group of tests that have a similar intent, time and resource, the test that has the highest likelihood of uncovering a whole class of errors should be used.

4. A good test should be neither too simple nor too complex. Each test should be executed separately.

The Testing Process

Systems should be tested as a single, monolithic unit only for small programs. Large systems are built out of sub-systems which are building out of modules which are composing of producers and functions. The testing process should therefore proceed in stages where testing is carried out incrementally in connection with system implementation.

The most widely used testing process consists of five stages

1. Unit testing
2. Module testing
3. Sub-system testing
4. Component testing
5. Integration testing

TYPES OF SOFTWARE TESTING

- There are different levels during the process of Testing. Levels of testing include the different methodologies that can be used while conducting Software Testing. Following are the main levels of Software Testing:

    1. Functional Testing.

    2. Non-Functional Testing.

1. Functional Testing

        - The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for.

        - Functional Testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

- There are five steps that are involved when testing an application for functionality.

Steps	Description
I	The determination of the functionality that the intended application is meant to perform.
II	The creation of test data based on the specifications of the application.
III	The output based on the test data and the specifications of the application.
IV	The writing of Test Scenarios and the execution of test cases.
V	The comparison of actual and expected results based on the executed test cases.

- An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

1. Unit Testing

- Here individual components are tested to ensure that they operate correctly. Each component is tested separately.
- This type of testing is performed by the developers before the setup is handed over to the testing team to formally execute the test cases.
- Unit testing is performed by the respective developers on the individual units of source code assigned areas.
- The developers use test data that is separate from the test data of the quality assurance team.

- The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing

- Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.
- There is a limit to the number of scenarios and test data that the developer can use to verify the source code. So after he has exhausted all options there is no choice but to stop unit testing and merge the code segment with other units.
- The testing of combined parts of an application to determine if they function correctly together is Integration testing.

2. Integration testing

- There are two methods of doing Integration Testing Bottom-up Integration testing and Top down Integration testing.

S.N.	Integration Testing Method
1	**Bottom-up integration** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	**Top-Down integration** This testing, the highest-level modules are tested first and progressively lower-level modules are tested after that.

- In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic those it will encounter in customers' computers, systems and network.

3. Module Testing:

- A module is collection of dependent components such as an object class, an abstract data type or some looser collection of procedures and functions. A module encapsulates related components so can be tested without other system modules.

4. Sub-System Testing:

Here his phase involves testing collection of modules which have been integrated into sub-systems. Sub-systems be independently designed and implemented. The most common problems which arise in large s/w systems are sub -systems interface mismatches.

5. System Testing:

- The sub-systems are integrated to the entire system. The testing process is concerned with finding errors which results from anticipated interactions between sub-systems and system components.
- It is also concerned with validating that the system meets its functional and non-functional requirements.
- This is the next level in the testing and tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards. This type of testing is performed by a specialized testing team.
- System testing is so important because of the following reasons:
  - System Testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.

   - The application is tested thoroughly to verify that it meets the functional and technical specifications.
   - The application is tested in an environment which is very close to the production environment where the application will be deployed.
   - System Testing enables us to test, verify and validate both the business requirements as well as the Applications Architecture.

6. **Regression Testing**

- Whenever a change in a software application is made it is quite possible that other areas within the application have been affected by this change.
- To verify that a fixed bug hasn't resulted in another functionality or business rule violation is Regression testing. The intent of Regression testing is to ensure that a change, such as a bug fix did not result in another fault being uncovered in the application.
- Regression testing is so important because of the following reasons:
   - Minimize the gaps in testing when an application with changes made has to be tested.
   - Testing the new changes to verify that the change made did not affect any other area of the application.
   - Mitigates Risks when regression testing is performed on the application.
   - Test coverage is increased without compromising timelines.
   - Increase speed to market the product.

7. **Acceptance Testing**

- This is arguably the most importance type of testing as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirements.
- The QA team will have a set of pre written scenarios and Test Cases that will be used to test the application.
- More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated.
- Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors or Interface gaps, but also to point out any bugs in the application that will result in system crashers or major errors in the application.
- By performing acceptance tests on an application the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.
- This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system producer rather than stimulated test data.
- Acceptance testing may reveal errors and omissions in the system requirements definition because the real data exercise the system in different ways from the test data. Acceptance testing may also reveal requirements problems where the systems facilities do not really meet the user's needs or the system performance is unacceptable.

(i) Alpha testing:

- Acceptance testing is sometimes called alpha testing. The alpha testing process continues until the system developer and the client agree with the delivery system is an acceptable implementation of the system requirements.

- This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined are known as alpha testing. During this phase, the following will be tested in the application:
    - Spelling Mistakes
    - Broken Links
    - Cloudy Directions
    - The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

(ii) Beta testing:

- When a system is to be marketed as a software product, a testing process called beta testing is often used.
- Beta testing involves delivering a system to a number of potential customers to agree to use that system.
- They report problems to the system developers. This exposes the product to real use and detects errors which may not have been anticipated by the system builders. After this feedback, the system is modified and either released or further beta testing or for general sale.
- This test is performed after Alpha testing has been successfully performed. In beta testing a sample of the intended audience tests the application. Beta testing is also known as pre-release testing. Beta test versions of software are ideally distributed to a wide

audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase the audience will be testing the following:

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release to the general public will increase customer satisfaction.

2. Non-Functional Testing

- ➢ Non-functional testing of Software involves testing the Software from the requirements which are non functional in nature related but important a well such as performance, security, user interface etc.
- ➢ Some of the important and commonly used non-functional testing types are mentioned as follows:

1. Performance Testing

- ➢ It is mostly used to identify any bottlenecks or performance issues rather than finding the bugs in software. There are different causes which contribute in lowering the performance of software:
  - o Network delay.

- Client side processing.
- Database transaction processing.
- Load balancing between servers.
- Data rendering.

- Performance testing is considered as one of the important and mandatory testing type in terms of following aspects:
  - Speed (i.e. Response Time, data rendering and accessing)
  - Capacity
  - Stability
  - Scalability
- It can be either qualitative or quantitative testing activity and can be divided into different sub types such as **Load testing** and **Stress testing**.

  Load Testing

  - A process of testing the behavior of the Software by applying maximum load in terms of Software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of Software and its behavior at peak time.
  - Most of the time, Load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test etc.
  - Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the Load testing for the Software. The quantity of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

- This testing type includes the testing of Software behavior under abnormal conditions. Taking away the resources, applying load beyond the actual load limit is Stress testing.

- The main intent is to test the Software by applying the load to the system and taking over the resources used by the Software to identify the breaking point. This testing can be performed by testing different scenarios such as:

    - Shutdown or restart of Network ports randomly.
    - Turning the database on or off.
    - Running different processes that consume resources such as CPU, Memory, server etc.

2. **Usability Testing**

- It is a black box technique and is used to identify any error(s) and improvements in the Software by observing the users through their usage and operation.

- According to Nielsen, Usability can be defined in terms of five factors i.e. Efficiency of use, Learn-ability, Memor-ability, Errors/safety, satisfaction. According to him the usability of the product will be good and the system is usable if it possesses the above factors.

- Nigel Bevan and Macleod considered that Usability is the quality requirement which can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

- Molich in 2000 stated that user friendly system should fulfill the following five goals i.e. Easy to Learn, Easy to Remember, Efficient to Use, Satisfactory to Use and Easy to Understand.

- In addition to different definitions of usability, there are some standards and quality models and methods which define the usability in the form of attributes and sub attributes such as ISO-9126, ISO-9241-11, ISO-13407 and IEEE std.610.12 etc.

3. UI Vs Usability Testing

- UI testing involves the testing of Graphical User Interface of the Software. This testing ensures that the GUI should be according to requirements in terms of color, alignment, size and other properties.

- On the other hand Usability testing ensures that a good and user friendly GUI is designed and is easy to use for the end user. UI testing can be considered as a sub part of Usability testing.

4. Security Testing

- Security testing involves the testing of Software in order to identify any flaws ad gaps from security and vulnerability point of view. Following are the main aspects which Security testing should ensure:
    - Confidentiality
    - Integrity
    - Authentication.
    - Availability.
    - Authorization.
    - Non-repudiation.
  - Software is secure against known and unknown vulnerabilities.
  - Software data is secure.
  - Software is according to all security regulations.

- Input checking and validation.
- SQL insertion attacks
- Injection flaws
- Session management issues.
- Cross-site scripting attacks.
- Buffer overflows vulnerabilities
- Directory traversal attacks.

5. **Portability Testing**

- Portability testing includes the testing of Software with intend that it should be re-useable and can be moved from another Software as well. Following are the strategies that can be used for Portability testing.
  - Transferred installed Software from one computer to another.
  - Building executable (.exe) to run the Software on different platforms.
- Portability testing can be considered as one of the sub parts of System testing, as this testing type includes the overall testing of Software with respect to its usage over different environments. Computer Hardware, Operating Systems and Browsers are the major focus of Portability testing. Following are some pre-conditions for Portability testing:
  - Software should be designed and coded, keeping in mind Portability Requirements.
  - Unit testing has been performed on the associated components.
  - Integration testing has been performed.
  - Test environment has been established.

6.4 Test case design

- IEEE Standard 610 (1990) defines test case as follows:

  “(1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

  “(2) (IEEE Std 829-1983) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.”

- “A test idea is a brief statement of something that should be tested. For example, if you're testing a square root function, one idea for a test would be ‘test a number less than zero’. The idea is to check if the code handles an error case.”

- Basically test design is the act of creating and writing test suites for testing software.
- Test analysis and identifying test conditions gives us a generic idea for testing which covers quite a large range of possibilities. But when we come to make a test case we need to be very specific. In fact now we need the exact and detailed specific input. But just having some values to input to the system is not a test, if you don’t know what the system is supposed to do with the inputs, you will not be able to tell that whether your test has passed or failed.
- Once a given input value has been chosen, the tester needs to determine what the expected result of entering that input would be and document it as part of the test case. Expected results include information displayed on a screen in response to an input. If we don’t decide on the expected results before we run a test then there might be a chance that we will notice that there is something wildly wrong. However, we would probably not notice small differences in calculations, or results that seemed to look OK. So we would conclude that the test had passed, when in fact the software has not given the correct result. Small differences in one calculation can add up to something very major later on, for example if results are multiplied by a large factor. Hence, ideally expected results should be predicted before the test is run.
- Incomplete and incorrect test case lead to incorrect and erroneous test output. To avoid this, the test case must be prepared in such a way that they check the software with all the

possible inputs. This process is known as exhaustive testing and the test case which is able to perform exhausting testing is ideal test case.

- In order to select a test case, certain question should be addressed:
  - How to select a test case?
  - On what basis are certain elements of program included or excluded from test case?
- There are two method used to generate test case. They are

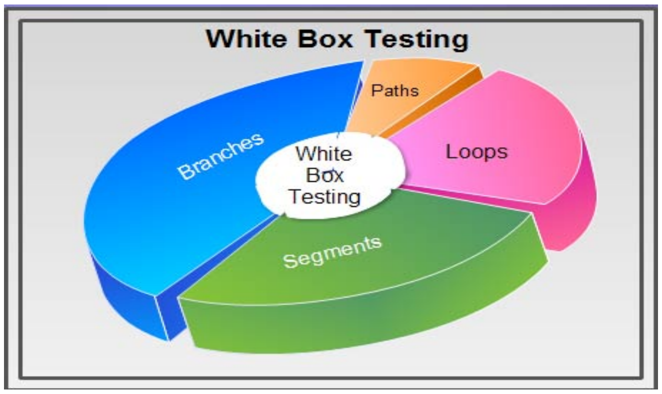
1. Code-based test case generation
2. Specification- based test case generation

1. Code-based test case generation
   - Also known as structure based test case generation
   - Is used to assess the entire software code to generate test case
   - It consider only the actual software code to generate test case and is not concern with the user requirement
   - Specially used for performing unit test
   - These test cases can easily test statement, branches, special values and symbol present in the unit being tested.
2. Specification –based test case generation
   - Uses specification, which indicate the function that are produced by the software to generate test case
   - It considers only external view of the software to generate test case but does not test the design decision and may not cover all statement of a program.

- Specially used for integration testing and system testing to ensure that the software is performing the required task.
- As it is derived from specification, the error present in these specification may remain uncovered.

6.5 White Box testing

- White box testing is also called as glass box testing.
- It is a test case design method that uses the control structure of the procedural design to the derive test cases.
- White box testing is the detailed investigation of internal logic and structure of the code.
- White box testing is also called glass testing or open box testing.
- In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code.
- The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.



Benefits of white box testing:

1. **Focused testing**: The programmer can test the program in pieces. It's much easier to give an individual suspect module a through workout in glass box testing than in black box testing.

2. **Testing coverage:** The programmer can also find out which parts of the program are exercised by any test. It is possible to find out which lines of code, which branches, or which paths haven't yet been tested.

3. **Control flow:** The programmer knows what the program is supported to do next, as a function of its current state.

4. **Data integrity:** The programmer knows which parts of the program modify any item of data. By tracking a data item through the system.

5. **Internal boundaries:** The programmer can see internal boundaries in the code that are completely invisible to the outside tester.

6. **Algorithmic specific:** The programmer can apply standard numerical analysis techniques to predict the results.

Various white box testing techniques:

1. Basis Path Testing:

The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining and use this measure as a guide for defining a basis set of execution paths.

Flow graph notation: Flow graph is a simple notation for the representation of control flow. Each structured construct has a corresponding flow graph symbol.
Flow graph node: Represents one or more procedural statements.
Edges or links: Represent flow control.
Regions: These are areas bounded by edges and nodes.
Each node that contains a condition is called a predicate node and is characterized by two or more edges emanating from it.

2. Cyclomatic Complexity:

Cyclomatic complexity is software metric that provide a quantitative measure of the logical complexity of a program.

The value computed for cyclomatic defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

Three ways of computing cyclomatic complexity:

1. The number of regions of the flow graph corresponds 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 the flow graph edges; N is the number of flow graph nodes.

3. Cyclomatic complexity V (G) for a flow graph G is also called as

V (G) = P+1

Where, P is the number of predicate nodes contained in the flow graph G.

Advantages	**Disadvantages**

As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. It helps in optimizing the code. Extra lines of code can be removed which can bring in hidden defects. Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased. Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested. It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

6.4 BLACK BOX TESTING

- Black box testing is also called as behavioral testing. This focuses on the functional requirements of the s/w. Black box testing enables the s/w engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
- The technique of testing without having any knowledge of the interior workings of the application is Black Box testing.
- The tester is oblivious to the system architecture and does not have access to the source code.
- Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

➢ Errors found by black box testing:

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures or external data base access.
4. Behavior or performance errors.
5. Initialization and termination errors.

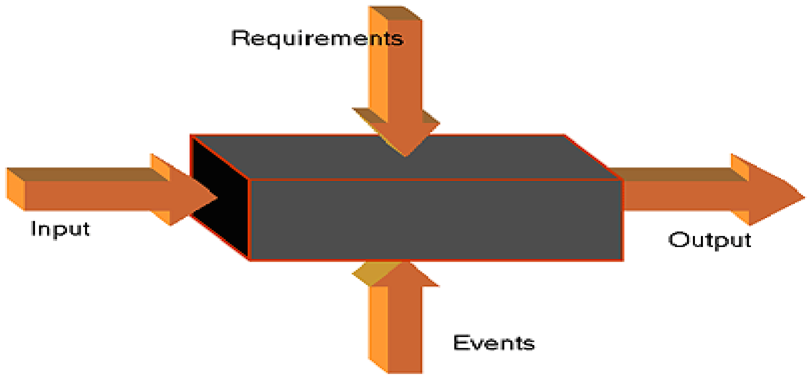


Figure: Black-box Testing

<u>**Various black boxes testing method:**</u>

1. Equivalent partitioning
2. Boundary value analysis

1. Equivalence Partitioning:

➢ It is a black box testing method that divides the inputs domain of a program into classes of data from which test cases can be derived.

➢ Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.

➢ Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition. An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is a specific numeric value, a range of values, a set of related values, or a boolean condition.

- The user can access the bank using a personal computer, provide a six-digit password, and follow with a series of typed commands that trigger various banking functions. During the log-on sequence, the software supplied for the banking application accepts data in the form:
  - area code—blank or three-digit number
  - prefix—three-digit number not beginning with 0 or 1
  - suffix—four-digit number
  - password—six digit alphanumeric string
  - commands—check, deposit, bill pay, and the like
- The input conditions associated with each data element for the banking application can be specified as area code:
  - Input condition, Boolean—the area code may or may not be present.
  - Input condition, range—values defined between 200 and 999, with specific exceptions.
  - prefix: Input condition, range—specified value >200
  - Input condition, value—four-digit length
  - Password: Input condition, Boolean—a password may or may not be present.
  - Input condition, value—six-character string.
  - Command: Input condition, set—containing commands noted previously.
- Applying the guidelines for the derivation of equivalence classes, test cases for each input domain data item can be developed and executed.
- Test cases are selected so that the largest numbers of attributes of an equivalence class are exercised at once.

- Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition.
- The input data to a program usually fall into number of different classes. These classes have common characteristics, for example positive numbers, negative numbers strings without blanks and so on. Programs normally behave in a comparable way for all members of a class. Because of this equivalent behavior, these classes are sometimes called equivalent partitions or domains.
- A systematic approach to defect testing is based on identifying a set of equivalence partitions which must be handled by a program.

<u>Guidelines for defining equivalence classes:</u>

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, one valid and two invalid equivalence classes are defined.
3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
4. If an input condition is Boolean, one valid and one invalid class are defined.

- Test cases for each input domain data item can be developed and executed by applying the guidelines for the derivation of equivalence classes.

2. Boundary Value Analysis:

- A great number of errors tend to occur at the boundaries of the input domain rather than in the center. So boundary value analysis (BVA) derives test cases from the output domain as well.
- BVA extends equivalence partitioning by focusing on data at the "edges" of an equivalence class.

- For reasons that are not completely clear, a greater number of errors tends to occur at the boundaries of the input domain rather than in the "center". It is for this reason that boundary value analysis (BVA) has been developed as a testing technique. Boundary value analysis leads to a selection of test cases that exercise bounding values.
- Boundary value analysis is a test case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.
- Most software engineers intuitively perform BVA to some degree. By applying these guidelines, boundary testing will be more complete, thereby having a higher likelihood for error detection

Guidelines for boundary value analysis:

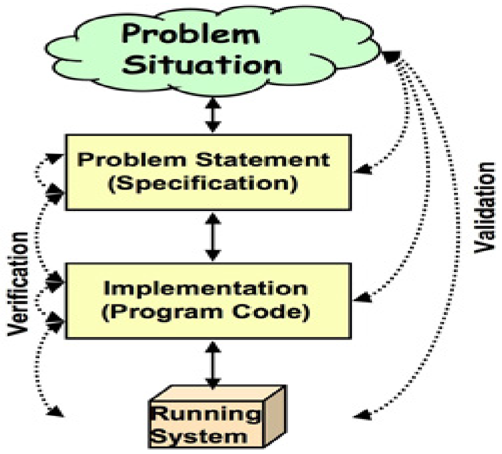
1. If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and just below a and b.
2. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers.
3. Apply guidelines 1 and 2 to output conditions.
4. If internal program data structures have prescribed boundaries, be certain to design a test case to exercise the data structure at its boundary.

Advantages	**Disadvantages**
1. Well suited and efficient for large code segments. 2. Code Access not required. 3. Clearly separates user's perspective from the	1. Limited Coverage since only a selected number of test scenarios are actually performed. 2. Inefficient testing, due to the

developer's perspective through visibly defined roles. 4. Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.	fact that the tester only has limited knowledge about an application. 3. Blind Coverage, since the tester cannot target specific code segments or error prone areas. 4. The test cases are difficult to design.

6.6 Verification & Validation

- The terms Verification and Validation are commonly used in software engineering to mean two different types of analysis. The usual definitions are:
    - Validation: Are we building the right system?
    - Verification: Are we building the system right?


Problem Situation
Problem Statement (Specification)
Implementation (Program Code)
Running System
Verification
Validation

- In other words, validation is concerned with checking that the system will meet the customer's actual needs, while verification is concerned with whether the system is well-engineered, error-free, and so on. Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful.

- The distinction between the two terms is largely to do with the role of specifications. Validation is the process of checking whether the specification captures the customer's needs, while verification is the process of checking that the software meets the specification.
- Verification includes all the activities associated with the producing high quality software: testing, inspection, design analysis, specification analysis, and so on.
- In contrast, validation is an extremely subjective process. It involves making subjective assessments of how well the (proposed) system addresses a real-world need. Validation includes activities such as requirements modeling, prototyping and user evaluation.
- In a traditional phased software lifecycle, verification is often taken to mean checking that the products of each phase satisfy the requirements of the previous phase. Validation is relegated to just the beginning and ending of the project: requirements analysis and acceptance testing.
- It assumes that the customer's requirements can be captured completely at the start of a project, and that those requirements will not change while the software is being developed. In practice, the requirements change throughout a project, partly in reaction to the project itself: the development of new software makes new things possible. Therefore both validation and verification are needed throughout the lifecycle.

Criteria	**Verification**	**Validation**
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
Objective	To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified	To ensure that the product actually meets the user's needs, and that the specifications were correct in the first place. In other words, to demonstrate that

	requirements.	the product fulfills its intended use when placed in its intended environment.
Question	Are we building the product *right*?	Are we building the *right* product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
Activities	▪ Reviews ▪ Walkthroughs ▪ Inspections	▪ Testing

These two terms are very confusing for people, who use them interchangeably. Let's discuss about them briefly.

S.N.	**Verification**	**Validation**
1	Are you building it right?	Are you building the right thing?
2	Ensure that the software system meets all the functionality.	Ensure that functionalities meet the intended behavior.
3	Verification takes place first and includes the checking for documentation, code etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by Testers.
5	Have static activities as it includes the reviews, walkthroughs, and inspections to verify that software is correct or not.	Have dynamic activities as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify the	It is a subjective process and involves subjective decisions on how well the

	Software.	Software works.