Unit-6
Software Testing

- Software Testing Process
- Principle of testing
- Black-Box Testing (Boundary- value analysis, Equivalence class partitioning)
- White –Box testing (Statement coverage, Path coverage, Cyclomatic Complexity)
- Software Verification and validation

Introduction

- Software Testing is a method to assess the functionality of the software program.
- The process checks whether the actual software matches the expected requirements and ensures the software is bug-free.
- The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements.
- It mainly aims at measuring the specification, functionality, and performance of a software program or application.
- Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Software testing.....

Software testing can be divided into two steps

- **Verification:** It refers to the set of tasks that ensure that the software correctly implements a specific function. It means "Are we building the product right?".
- **Validation:** It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. It means "Are we building the right product?".

Importance of Software Testing

- **Defects can be identified early:** Software testing is important because if there are any bugs they can be identified early and can be fixed before the delivery of the software.
- **Improves quality of software:** Software Testing uncovers the defects in the software, and fixing them improves the quality of the software.
- **Increased customer satisfaction:** Software testing ensures reliability, security, and high performance which results in saving time, costs, and customer satisfaction.
- **Helps with scalability:** Software testing type non-functional testing helps to identify the scalability issues and the point where an application might stop working.
- **Saves time and money:** After the application is launched it will be very difficult to trace and resolve the issues, as performing this activity will incur more costs and time. Thus, it is better to conduct software testing at regular intervals during software development.

Software testing Process

- The software testing process in software engineering is a critical phase that ensures the quality, reliability, and performance of software products.
- It involves a series of systematic activities designed to identify defects, verify that the software meets specified requirements, and validate that it is fit for its intended use.

Software Testing Process or Software Testing Life Cycle (STLC)

- Following steps are involved in Software Testing Life Cycle (STLC). Each step is have its own Entry Criteria and deliverable.

- Requirement Analysis
- Test Planning
- Test Case Development
- Environment Setup
- Test Execution
- Test Cycle Closure

Software Testing Life Cycle (STLC)

Requirement Analysis:

- Requirement Analysis is the very first step in **Software Testing Life Cycle (STLC)**.
- In this step Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements.
- If any conflict, missing or not understood any requirement, then QA team follow up with the various stakeholders like Business Analyst, System Architecture, Client, Technical Manager/Lead etc to better understand the detail knowledge of requirement.
- From very first step QA involved in the where STLC which helps to prevent the introducing defects into Software under test.

The activities that take place during the Requirement Analysis stage include:

- Reviewing the software requirements document (SRD) and other related documents
- Interviewing stakeholders to gather additional information
- Identifying any ambiguities or inconsistencies in the requirements
- Identifying any missing or incomplete requirements
- Identifying any potential risks or issues that may impact the testing process

Test Planning:

- This phase also called as Test Strategy phase.
- In this phase typically Test Manager (or Test Lead based on company to company) involved to determine the effort and cost estimates for entire project.
- This phase will be kicked off once the requirement gathering phase is completed & based on the requirement analysis, start preparing the Test Plan.
- The Result of Test Planning phase will be Test Plan or Test strategy & Testing Effort estimation documents.

The activities that take place during the Test Planning stage include:

- Identifying the testing objectives and scope
- Developing a test strategy: selecting the testing methods and techniques that will be used
- Identifying the testing environment and resources needed
- Identifying the test cases that will be executed and the test data that will be used
- Estimating the time and cost required for testing
- Identifying the test deliverables and milestones
- Assigning roles and responsibilities to the testing team
- Reviewing and approving the test plan

Test Case Development:

- The test case development activity is started once the test planning activity is finished.
- This is the phase of STLC where testing team write down the detailed test cases.
- Along with test cases testing team also prepare the test data if any required for testing.
- Once the test cases are ready then these test cases are reviewed by peer members or QA lead.
- Also the Requirement Traceability Matrix (RTM) is prepared.
- The Requirement Traceability Matrix is an industry-accepted format for tracking requirements where each test case is mapped with the requirement.
- Using this RTM we can track backward & forward traceability.

The activities that take place during the Test Case Development stage include:

- Identifying the test cases that will be developed
- Writing test cases that are clear, concise, and easy to understand
- Creating test data and test scenarios that will be used in the test cases
- Identifying the expected results for each test case
- Reviewing and validating the test cases
- Updating the requirement traceability matrix (RTM) to map requirements to test cases

Test Environment Setup:

- Setting up the test environment is vital part of the STLC.
- Basically test environment decides on which conditions software is tested.
- This is independent activity and can be started parallel with Test Case Development.
- In process of setting up testing environment test team is not involved in it.
- Based on company to company may be developer or customer creates the testing environment.
- Mean while testing team should prepare the smoke test cases to check the readiness of the test environment setup.

Test Execution:

- Once the preparation of Test Case Development and Test Environment setup is completed then test execution phase can be kicked off.
- In this phase testing team start executing test cases based on prepared test planning & prepared test cases in the prior step.
- If any test case is failed then corresponding defect can be reported to developer team via bug tracking system & bug can be linked for corresponding test case for further analysis.
- Ideally every failed test case should be associated with at least single bug.
- Using this linking we can get the failed test case with bug associated with it.
- Once the bug fixed by development team then same test case can be executed based on your test planning.

The activities that take place during the test execution stage of the Software Testing Life Cycle (STLC) include:

- **Test execution:** The test cases and scripts created in the test design stage are run against the software application to identify any defects or issues.
- **Defect logging:** Any defects or issues that are found during test execution are logged in a defect tracking system, along with details such as the severity, priority, and description of the issue.
- **Test data preparation:** Test data is prepared and loaded into the system for test execution
- **Test environment setup:** The necessary hardware, software, and network configurations are set up for test execution
- **Test execution:** The test cases and scripts are run, and the results are collected and analyzed.
- **Test result analysis:** The results of the test execution are analyzed to determine the software's performance and identify any defects or issues.
- **Defect retesting:** Any defects that are identified during test execution are retested to ensure that they have been fixed correctly.
- **Test Reporting:** Test results are documented and reported to the relevant stakeholders.

Test Cycle Closure:

- Call out the testing team member meeting & evaluate cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software.
- Discuss what all went good, which area needs to be improve & taking the lessons from current STLC as input to upcoming test cycles, which will help to improve bottleneck in the STLC process.
- Test case & bug report will analyze to find out the defect distribution by type and severity.
- Once complete the test cycle then test closure report & Test metrics will be prepared.
- Test result analysis to find out the defect distribution by type and severity.

The main activities that take place during the test closure stage include:

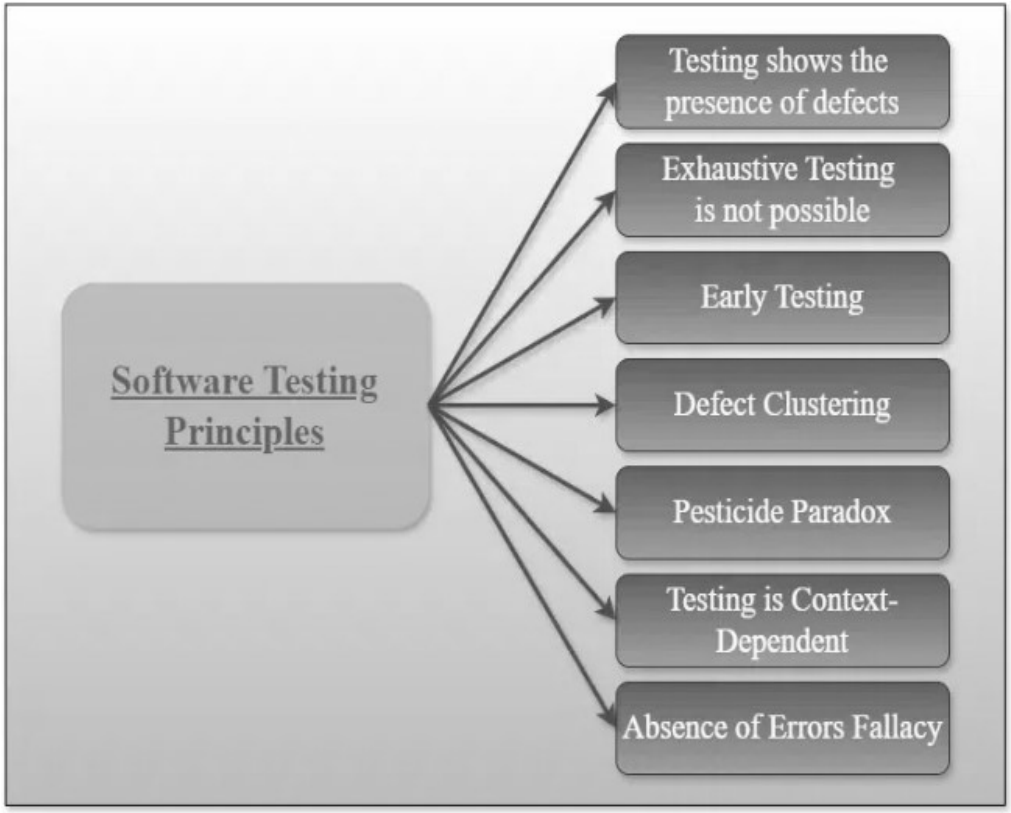
- **Test summary report:** A report is created that summarizes the overall testing process, including the number of test cases executed, the number of defects found, and the overall pass/fail rate.
- **Defect tracking:** All defects that were identified during testing are tracked and managed until they are resolved.
- **Test environment clean-up:** The test environment is cleaned up, and all test data and test artifacts are archived.
- **Test closure report:** A report is created that documents all the testing-related activities that took place, including the testing objectives, scope, schedule, and resources used.
- **Knowledge transfer:** Knowledge about the software and testing process is shared with the rest of the team and any stakeholders who may need to maintain or support the software in the future.
- **Feedback and improvements:** Feedback from the testing process is collected and used to improve future testing processes

Principle of Software Testing

- Software testing is a procedure of implementing software or the application to identify the defects or bugs.
- For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time.

Seven essential principles of software testing are:

- Testing shows the presence of defects
- Exhaustive testing is not possible
- Early testing
- Defect clustering
- Pesticide paradox
- Testing is Context-Dependent
- Absence of Errors fallacy



Principle of Software Testing.....

1. Testing shows the Presence of Defects

- The goal of software testing is to prevent the software fail.
- Software testing reduces the presence of defects.
- Software testing talks about the presence of defects and doesn't talk about the absence of defects.
- Software testing can ensure that defects are present but it can not prove that software is defect-free.
- Even multiple tests can never ensure that software is 100% bug-free.
- Testing can reduce the number of defects but not remove all defects.

2. Exhaustive Testing is not Possible

- It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing.
- Exhaustive testing is impossible means the software can never test at every test case.
- It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case.
- If the software will test every test case then it will take more cost, effort, etc., which is impractical.

3. Early Testing

- To find the defect in the software, early test activity shall be started.
- The defect detected in the early phases of SDLC will be very less expensive.
- For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

Principle of Software Testing.....

4. Defect Clustering

- In a project, a small number of modules can contain most of the defects.
- The Pareto Principle for software testing states that 80% of software defects come from 20% of modules.

5. Pesticide Paradox

- Repeating the same test cases, again and again, will not find new bugs.
- So it is necessary to review the test cases and add or update test cases to find new bugs.

6. Testing is Context-Dependent

- The testing approach depends on the context of the software developed.
- Different types of software need to perform different types of testing.
- For example, The testing of the e-commerce site is different from the testing of the Android application.

7. Absence of Errors Fallacy

- If a built software is 99% bug-free but does not follow the user requirement then it is unusable.
- It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

Test Case Design

- Test case design in software engineering is the process of creating detailed, structured scenarios or conditions under which a software application will be tested to verify its functionality, performance, and reliability.
- The goal of test case design is to ensure that the software meets its requirements and behaves as expected in various situations.

Purpose of Test Case Design

- **Identify defects:** To find bugs or errors in the software early in the development process.
- **Verify functionality:** To ensure that the software performs its intended tasks correctly.
- **Validate requirements:** To confirm that the software meets the user's needs and expectations.
- **Improve software quality:** To enhance the overall reliability, usability, and performance of the software.

Elements of a Test Case

- **Test Case ID:** A unique identifier for each test case.
- **Test Case Name:** A descriptive name that summarizes the test objective.
- **Preconditions:** The conditions that must be met before the test can be executed.
- **Test Steps:** The specific actions to be performed during the test.
- **Input Data:** The data values used as input for the test.
- **Expected Output:** The anticipated result of the test if the software functions correctly.
- **Actual Output:** The actual result obtained when the test is executed.
- **Postconditions:** The conditions that exist after the test has been executed.
- **Pass/Fail Criteria:** The criteria used to determine whether the test has passed or failed.

Test Case Design Techniques

1. Specification-Based or Black-Box techniques

- Specification-based testing, also known as black-box testing, is a testing technique that focuses on testing the software system based on its functional requirements and specifications without any knowledge about the underlying code or system structure.

- **Boundary Value Analysis (BVA)**

- Boundary value analysis identifies errors at the input domain's boundary.
- A simple example of boundary value analysis would be testing a text box that requires the user to enter a number between 1 and 10.
- In this case, the boundary values would be 1 and 10, and we would test with values that are just above, at, and just below these boundaries.
- **Example:** We would test with 0, 1, 2, 9, 10, and 11. We can expect that errors or defects are most likely to occur at or near the boundary values. Identifying these issues early can help prevent them from causing problems later in the software development process.

Specification-Based or Black-Box techniques.....

- **Equivalence Partitioning (EP)**
- Equivalence Partitioning is another technique that helps reduce the required test cases.
- By partitioning test input data into classes with an equivalent number of data, one can design test cases for each class or partition.
- This technique ensures that one thoroughly tests the software while minimizing the required test cases.
- **Example:** If a program requires an input of numbers between 1 and 100, an EP test would include a range of values, such as 1-50 and 51-100, and numbers outside that range, such as -1 or 101. Testing one value from each partition is sufficient to test all values within that partition.

Specification-Based or Black-Box techniques.....

- **Decision Table Testing**
- Decision Table Testing is a technique that involves designing test cases based on decision tables formulated using different combinations of inputs and their corresponding outputs based on various conditions and scenarios sticking to other business rules.
- This technique ensures that we test the software thoroughly and accurately.
- **Example:** If a program offers discounts based on the type of customer and the amount spent, a decision table would list all possible combinations of customer types and the amount paid to receive a discount. Each cell in the table would specify the value that should be applied. Testers can ensure the program behaves correctly under various scenarios by testing all combinations.

Specification-Based or Black-Box techniques.....

- **State Transition Diagrams(STD) :**
- Developers use State Transition Diagrams(STD) to test software with a finite number of states of different types.
- A set of rules that define the response to various inputs guides the transition from one state to another.
- This technique is handy for systems with specific workflows within them.
- **Example:** Consider an e-commerce website that has different states such as “logged out,” “logged in,” “cart empty,” “cart not empty,” and “order placed.” The transitions between the states would be triggered by login in and logout, adding the product to the cart, removing the product from the cart, proceeding to checkout, etc. An STD can help visualize and test such complex states and transitions in a system.

Specification-Based or Black-Box techniques.....

- **Use Case Testing**
  - Use Case Testing involves designing test cases to execute different business scenarios and end-user functionalities.
- **Example:** A use case could be a "student enrolling in a course" on an academic website. Test cases would simulate the enrollment process and verify the system's response from a student's perspective.

2. Structure-Based or White-Box techniques

- Structure-based testing, also known as white-box testing, is a testing technique that involves the testing of internal structures or components of software applications.
- In this approach, the tests interact with the code directly. These tests are designed to ensure the code works correctly and efficiently.

Statement Testing and Coverage

- Statement Testing and Coverage is a technique that involves executing all the executable statements in the source code at least once.
- We then calculate the percentage of executable statements as per the given requirement.
- **Example:** Consider code that inputs two numbers and checks if the first number is greater than or equal to the second. A statement coverage test would verify that both the "greater than" and "equal to" statements are executed during testing to ensure that all code branches are covered.

Structure-Based or White-Box techniques.....

Decision Testing Coverage

- Decision Testing Coverage, also known as branch coverage, validates all the branches in the code by executing each possible branch from each decision point at least once.
- This helps ensure that no branch leads to unexpected application behavior.
- **Example:** If a program requires an input of a number between 1 and 100 and uses an "if/else" statement to check if the number is even, decision testing coverage would ensure that both the even and odd outcomes have been tested to confirm all possible scenarios have been checked.

Structure-Based or White-Box techniques .....

Condition Testing

- Condition Testing, also known as Predicate coverage testing.
- It involves evaluating each Boolean expression in the code and checking its output values, TRUE or FALSE, against the expected outcomes.
- This test checks all outcomes at least once to achieve 100% code coverage. We design test cases that make it easy to execute the condition outcomes.

- **Example:** If a program determines whether a user is eligible for a discount based on age, condition testing would verify that the code handles each age group accurately. It would test age values such as one less than, one more than, and within the age range requirement to evaluate if the code performs as expected.

Structure-Based or White-Box techniques .....

Multiple Condition Testing

- Multiple Condition Testing aims to test different combinations of conditions to achieve 100% coverage.
- This technique requires two or more test scripts, which may require more effort.
- **Example:** If a program uses an "if/else" statement to check age and gender to provide a discount, multiple condition testing would verify that the program handles all possible scenarios correctly.
- It would test various age ranges and gender combinations to ensure the code performs accurately for all possibilities.

All Path Testing

- All Path Testing leverages the source code of a program to find every executable path.
- **Example:** If a program asks a user for two inputs (A and B) and has multiple conditions, All Path Testing would ensure that each condition is tested independently. The technique would test all combinations of A and B, including zero, negative, and positive Testings, to identify any potential errors in the code.

Statement Coverage Testing

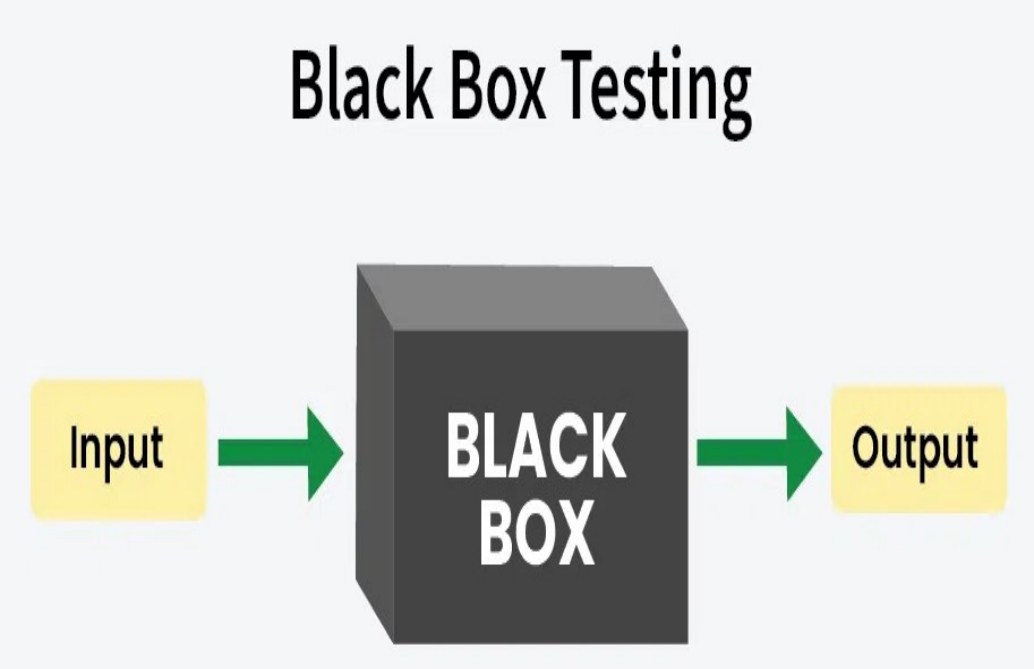
- A fundamental method of software testing called “statement coverage testing” makes sure that every statement in a piece of code is run at least once in order to gauge how thorough the testing was.
- This method offers useful insights into how thoroughly a program’s source code has been checked by monitoring the execution of each line of code.

3. Experience-Based techniques

- The experience-based techniques can be broadly categorized into the following types:
- **Error Guessing**
- Error Guessing is a testing technique that relies on the testers skill, intuition, and experience to anticipate potential errors in the product.
- **Exploratory Testing**
- On the other hand, Exploratory Testing is a technique used to test applications without any formal documentation.
- This method is recommended once scripted testing has been executed when working on the software.

Black Box Testing

- Black-box testing is a type of software testing in which the tester is not concerned with the software's internal knowledge or implementation details but rather focuses on validating the functionality based on the provided specifications or requirements.



Types Of Black Box Testing

- The following are the various categories of black box testing:
  - Functional Testing
  - Regression Testing
  - Non functional Testing (NFT)
- **Functional Testing**
- Functional testing is defined as a type of testing that verifies that each function of the software application works in conformance with the requirement and specification.
- This testing is not concerned with the source code of the application.
- Each functionality of the software application is tested by providing appropriate test input, expecting the output, and comparing the actual output with the expected output.
- This testing focuses on checking the user interface, APIs, database, security, client or server application, and functionality of the Application Under Test.
- Functional testing can be manual or automated.
- It determines the system's software functional requirements.

Types Of Black Box Testing.....

- **Regression Testing**

- Regression Testing is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made.
- Regression means the return of something and in the software field, it refers to the return of a bug. It ensures that the newly added code is compatible with the existing code.
- In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

Types Of Black Box Testing.....

- **Non-functional Testing**
- Non-functional testing is a software testing technique that checks the non-functional attributes of the system.
- It is designed to test the readiness of a system as per non-functional parameters which are never addressed by functional testing.
- It is as important as functional testing.
- It is also known as NFT. This testing is not functional testing of software. It focuses on the software's performance, usability, and scalability.

Advantages of Black Box Testing

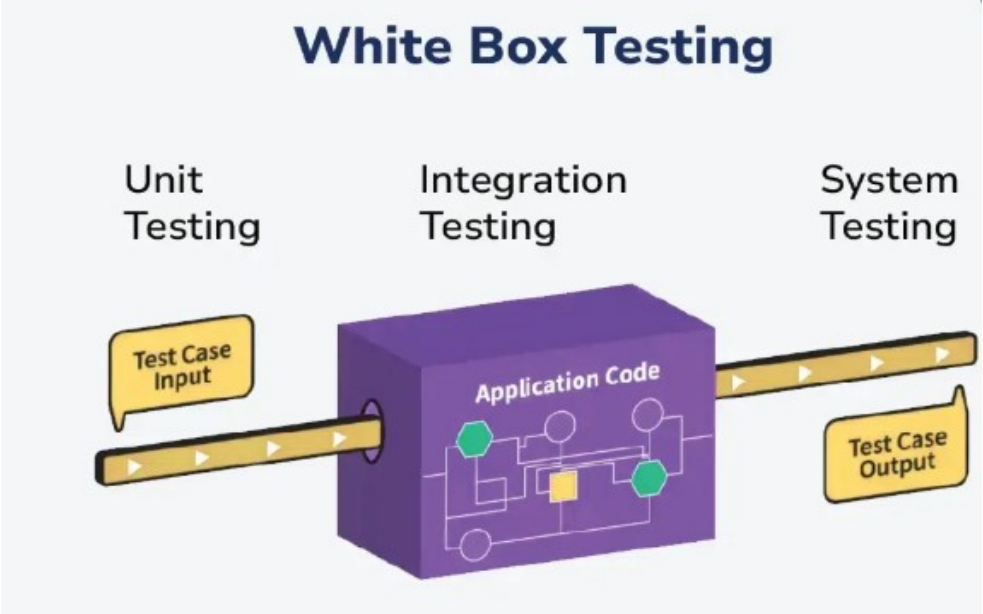
- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used to find the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

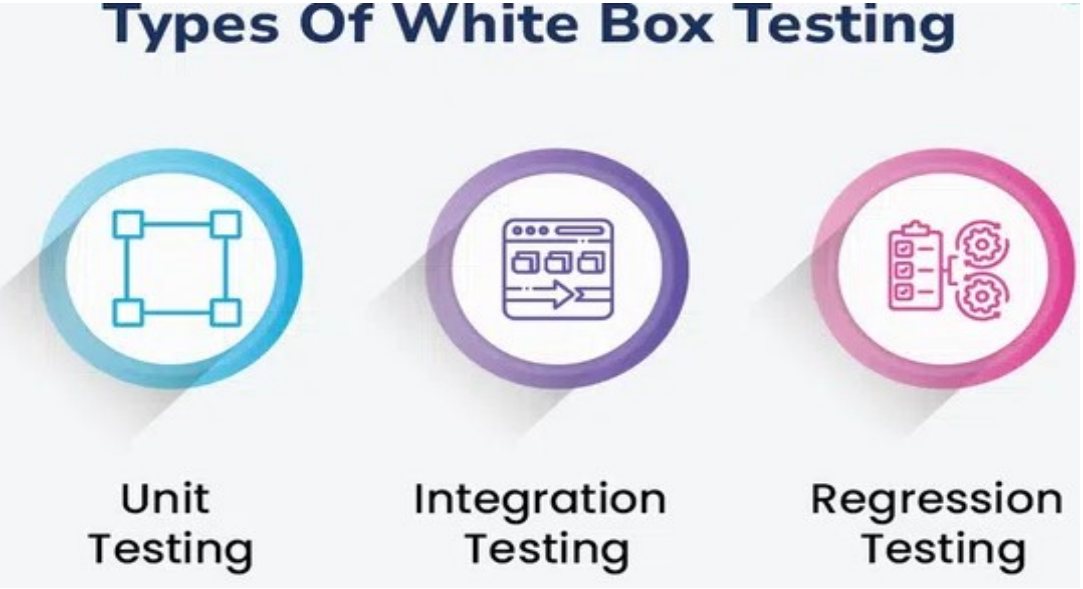
White Box Testing

- White box testing is a software testing technique that involves testing the internal structure and workings of a software application .
- The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.
- White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure.



Types Of White Box Testing

- White box testing can be done for different purposes.
- The three main types are:
  - Unit Testing
  - Integration Testing
  - Regression Testing



Types Of White Box Testing.....

Unit Testing

- Checks if each part or function of the application works correctly.
- Ensures the application meets design requirements during development.

Integration Testing

- Examines how different parts of the application work together.
- Done after unit testing to make sure components work well both alone and together.

Regression Testing

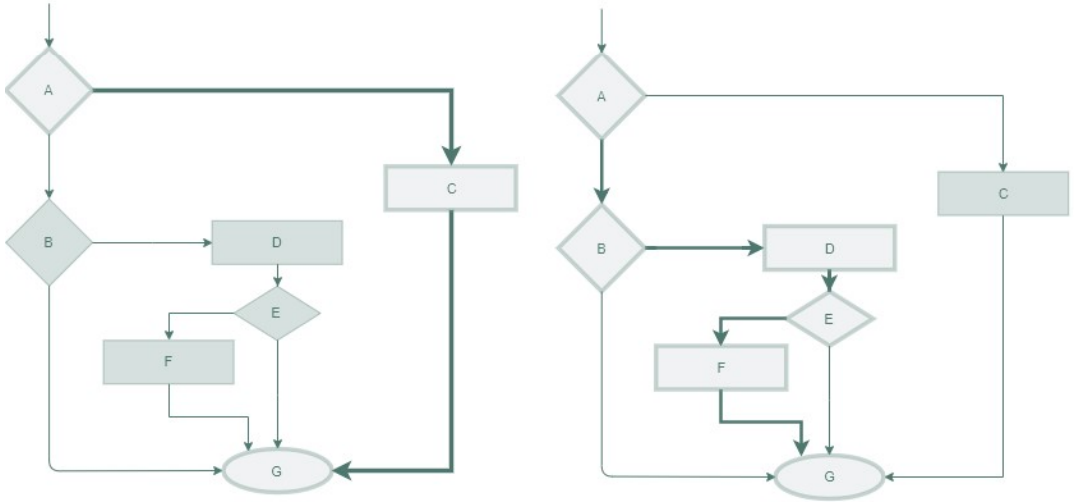
- Verifies that changes or updates don't break existing functionality.
- Ensures the application still passes all existing tests after updates.

White Box Testing Techniques

- One of the main benefits of white box testing is that it allows for testing every part of an application.
- To achieve complete code coverage, white box testing uses the following techniques:

1. Statement Coverage

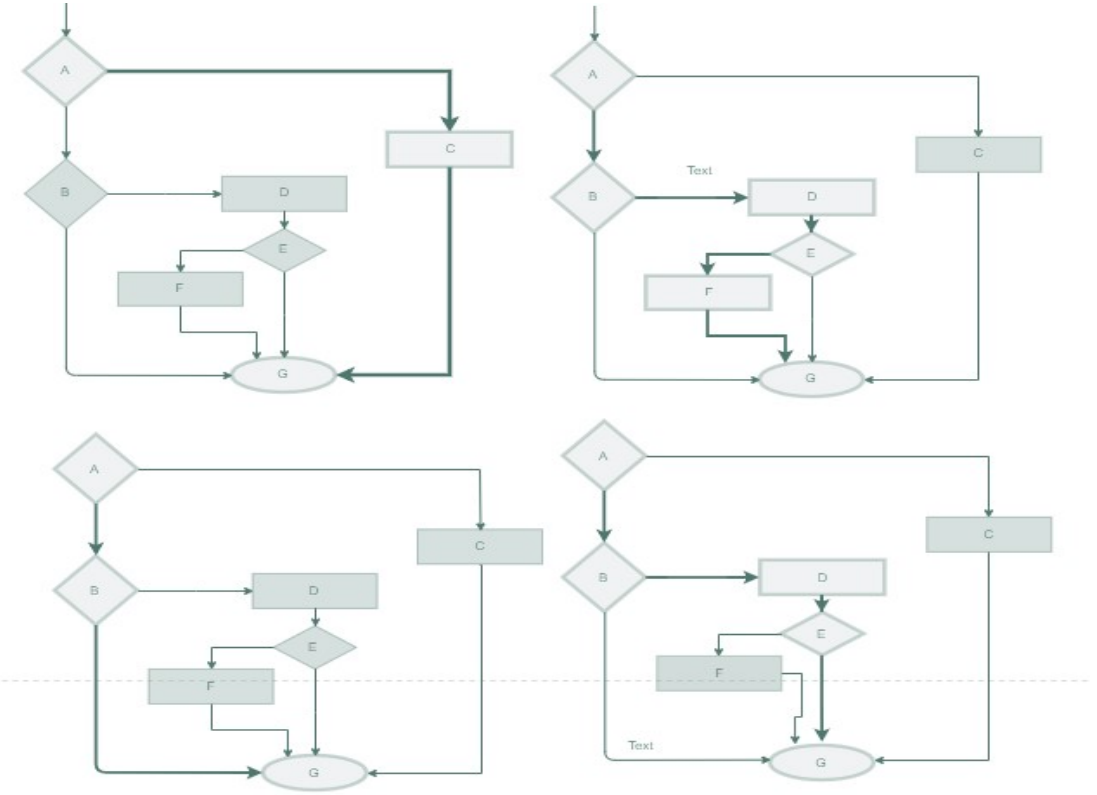
- In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested.
- In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.



White Box Testing Techniques.....

2. Branch Coverage

- In this technique, test cases are designed so that each branch from all decision points is traversed at least once.
- In a flowchart, all edges must be traversed at least once.



White Box Testing Techniques.....

- **3. Condition Coverage**
- In this technique, all individual conditions must be covered as shown in the following example:

```
READ X, Y
IF(X == 0 || Y == 0)
PRINT '0'
#TC1 – X = 0, Y = 55
#TC2 – X = 5, Y = 0
```

- **4. Multiple Condition Coverage**
- In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let's consider the following example:

```
READ X, Y
IF(X == 0 || Y == 0)
PRINT '0'
#TC1: X = 0, Y = 0
#TC2: X = 0, Y = 5
#TC3: X = 55, Y = 0
#TC4: X = 55, Y = 5
```

White Box Testing Techniques.....

5. Basis Path Testing

- In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

Steps:

- Make the corresponding control flow graph
- Calculate the cyclomatic complexity
- Find the independent paths
- Design test cases corresponding to each independent path
  - V(G) = P + 1, where P is the number of predicate nodes in the flow graph
  - V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
  - V(G) = Number of non-overlapping regions in the graph
  - #P1: 1 – 2 – 4 – 7 – 8
  - #P2: 1 – 2 – 3 – 5 – 7 – 8
  - #P3: 1 – 2 – 3 – 6 – 7 – 8
  - #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

White Box Testing Techniques.....

6. Loop Testing

- Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

Simple loops: For simple loops of size n, test cases are designed that:

- Skip the loop entirely
- Only one pass through the loop
- 2 passes
- m passes, where m < n
- n-1 ans n+1 passes

- **Nested loops:** For nested loops, all the loops are set to their minimum count, and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
- **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each. If they're not independent, treat them like nesting.

Statement Coverage

- It is one type of white box testing technique that ensures that all the statements of the source code are executed at least once.
- It covers all the paths, lines, and statements of a source code.
- It is used to design test box cases where it will find out the total number of executed statements out of the total statements present in the code.

Formula:

- Statement coverage = (Number of executed statements / Total number of statements in source code) * 100

Example 1:

```
Read A
Read B
if A > B
    Print "A is greater than B"
else
    Print "B is greater than A"
endif
```

- **Case 1:**

If A = 7, B= 3

No of statements Executed= 5

Total statements= 7

Statement coverage= 5 / 7 * 100

= 71.00 %

- **Case 2:**

If A = 4, B= 8

No of statements Executed= 6

Total statements= 7

Statement coverage= 6 / 7 * 100

= 85.20 %

Path Coverage Testing

- A structural white-box testing method called path coverage testing is used in software testing to examine and confirm that every possible path through a program's control flow has been tested at least once.
- This approach looks at the program's source code to find different paths, which are collections of statements and branches that begin at the entry point and end at the exit point of the program.
- In Path Coverage Testing, a **path** is defined as a unique sequence of statements in the program from the start to the end, passing through decision points (like if, while, for loops, etc.).
- Path coverage aims to execute every possible path in the code, including all combinations of branches.

Types of Paths:

- **Independent Path:** A path that traverses through a unique set of branches and decisions.
- **Composite Path:** A combination of independent paths that may share certain common parts.

- If you have a program with two decision points (if statements), there are multiple paths you could take:
  - Path 1: Both conditions are true
  - Path 2: The first condition is true, the second is false
  - Path 3: The first condition is false, the second is true
  - Path 4: Both conditions are false

Example:

```
Read A
Read B
if A > B:
    if A > 0:
        Print "A is positive and greater than B"
    else:
        Print "A is non-positive and greater than B"
else:
    Print "B is greater than A"
```

Identifying the Paths:

- **Path 1:** A > B is true, A > 0 is true — Print "A is positive and greater than B"
- **Path 2:** A > B is true, A > 0 is false — Print "A is non-positive and greater than B"
- **Path 3:** A > B is false — Print "B is greater than A"

Path Coverage Formula:

The formula for calculating path coverage can be described as:

- Path Coverage=(Number of Executed Paths/Total Number of Paths)×100

So, if you have **3 possible paths** (like in the example), but you've tested only **2** of them, your path coverage will be:

- Path Coverage=(2/3)×100=66.67

Cyclomatic Complexity

- The cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it.
- It is a software metric used to indicate the complexity of a program.
- It is computed using the control flow graph of the program.
- The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e. if the second command might immediately follow the first command.
- For example, if the source code contains no control flow statement then its cyclomatic complexity will be 1, and the source code contains a single path in it. Similarly, if the source code contains one **if condition** then cyclomatic complexity will be 2 because there will be two paths one for true and the other for false.

- **Formula for Calculating Cyclomatic Complexity**
  - Mathematically, for a structured program, the directed graph inside the control flow is the edge joining two basic blocks of the program as control may pass from first to second.

So, **cyclomatic complexity M** would be defined as,

M = E – N + 2P

where

E = the number of edges in the control flow graph
N = the number of nodes in the control flow graph
P = the number of connected components

- In case, when exit point is directly connected back to the entry point. Here, the graph is strongly connected, and cyclometric complexity is defined as

M = E – N + P

- In the case of a single method, P is equal to 1. So, for a single subroutine, the formula can be defined as

M = E – N + 2

```
def example(A, B, C):
    if A > B:
        if C > 0:
            print("A > B and C > 0")
        else:
            print("A > B and C <= 0")
    else:
        print("B > A")
```

Step 1: Draw the Control Flow Graph (CFG)

Start → Decision 1: if A > B

Yes → Decision 2: if C > 0

Yes → Print "A > B and C > 0"

No → Print "A > B and C <= 0"

No → Print "B > A"

Nodes (N):

Node 1: Start

Node 2: if A > B

Node 3: if C > 0

Node 4: Print "A > B and C > 0"

Node 5: Print "A > B and C <= 0"

Node 6: Print "B > A"

- So, N=6.
- **Edges (E):**

Edge 1: Start → if A > B

Edge 2: if A > B → if C > 0 (True branch)

Edge 3: if C > 0 → Print "A > B and C > 0" (True branch)

Edge 4: if C > 0 → Print "A > B and C <= 0" (False branch)

Edge 5: if A > B → Print "B > A" (False branch)

Edge 6: Print "A > B and C > 0" → End

Edge 7: Print "A > B and C <= 0" → End

Edge 8: Print "B > A" → End

- So, E=8.

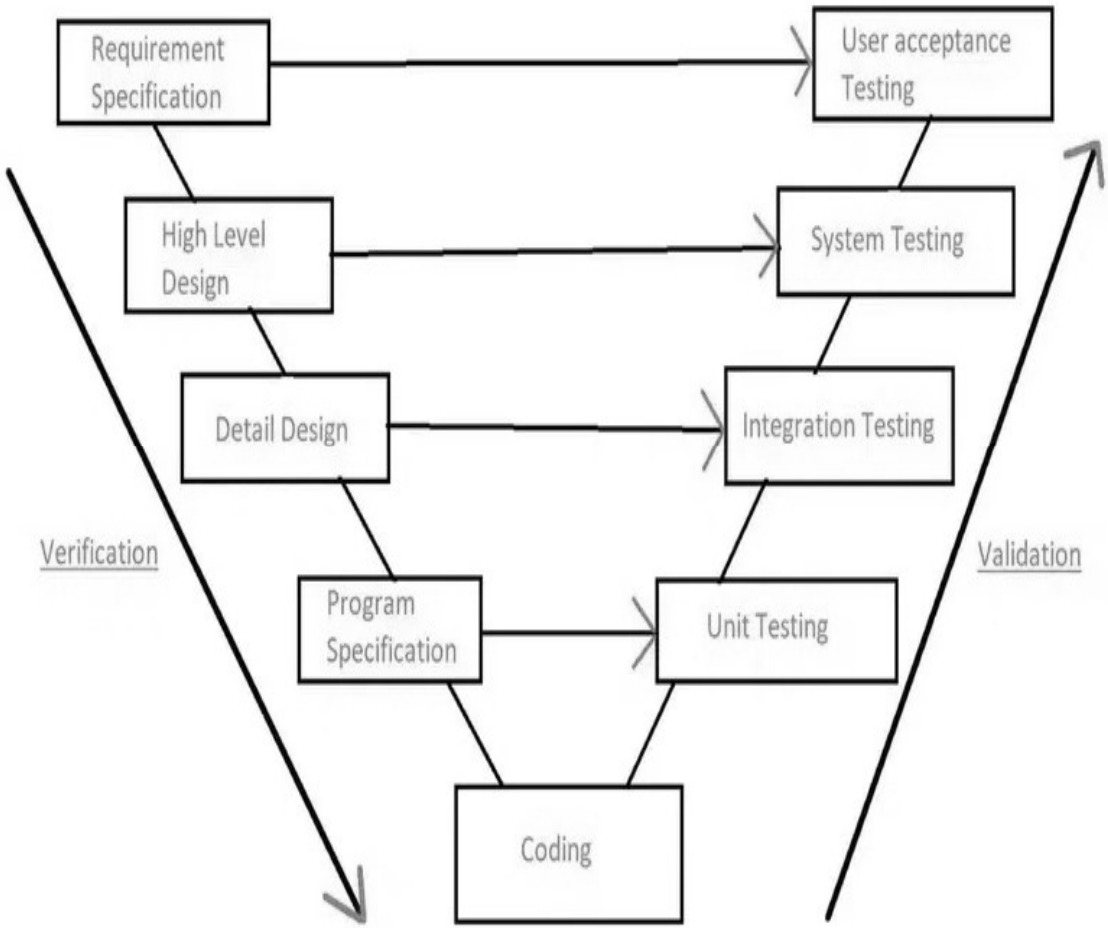
Step 2: Calculate Cyclomatic Complexity

- Using the formula for a single subroutine:
- M=E−N+2
- M=8−6+2=4

Thus, the cyclomatic complexity for this function is **4.**

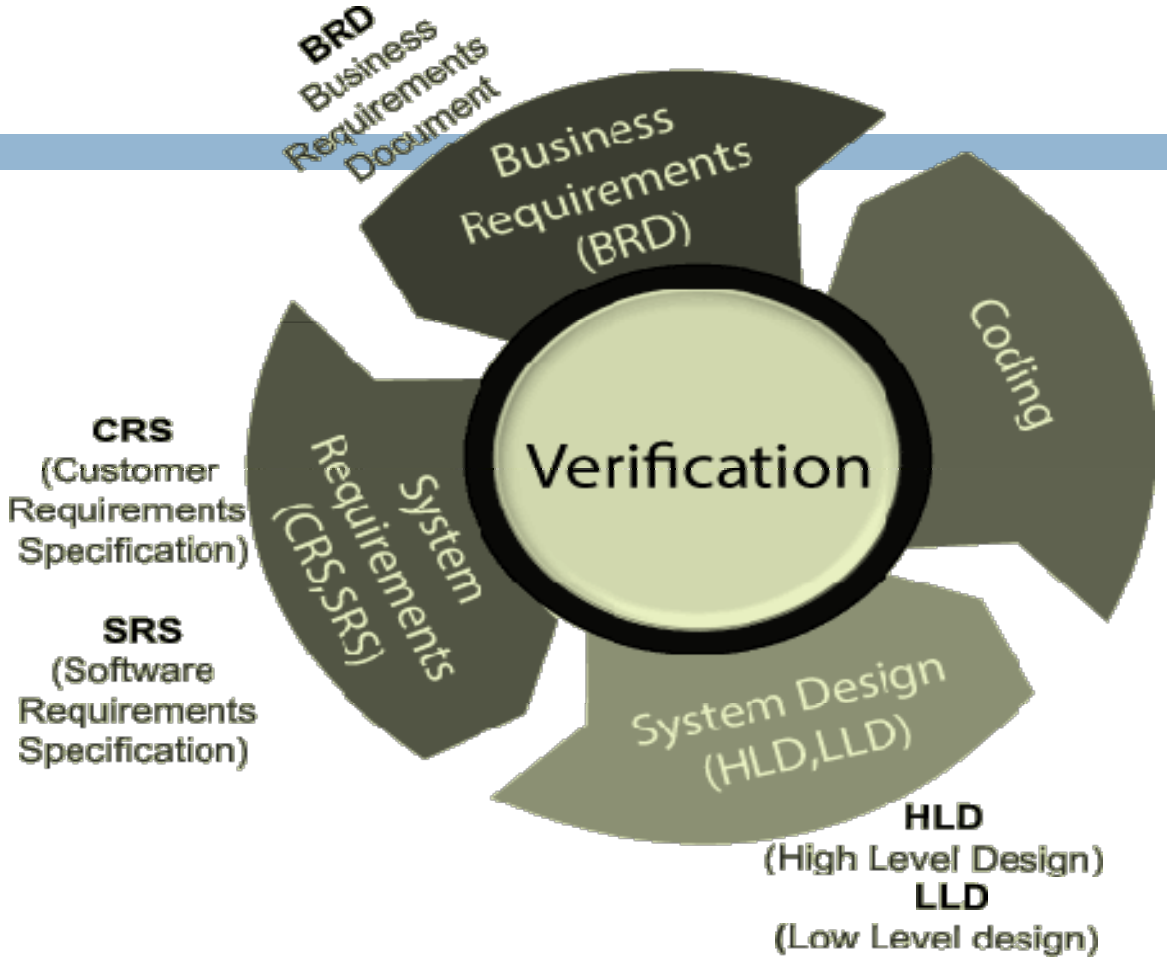
Verification and Validation

- **Verification and Validation** is the process of investigating whether a software system satisfies specifications and standards and fulfills the required purpose.
- **Barry Boehm** described verification and validation as the following:
  - **Verification:** Are we building the product right?
  - **Validation:** Are we building the right product?



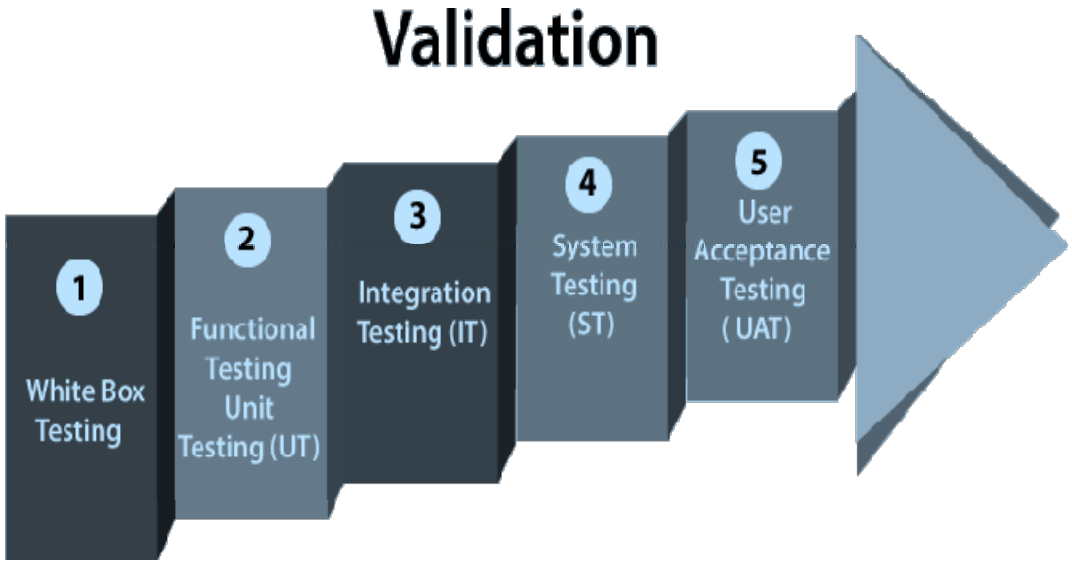
Verification

- Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.
- It is also known as static testing, where we are ensuring that "**we are developing the right product or not**".
- And it also checks that the developed application fulfilling all the requirements given by the client.

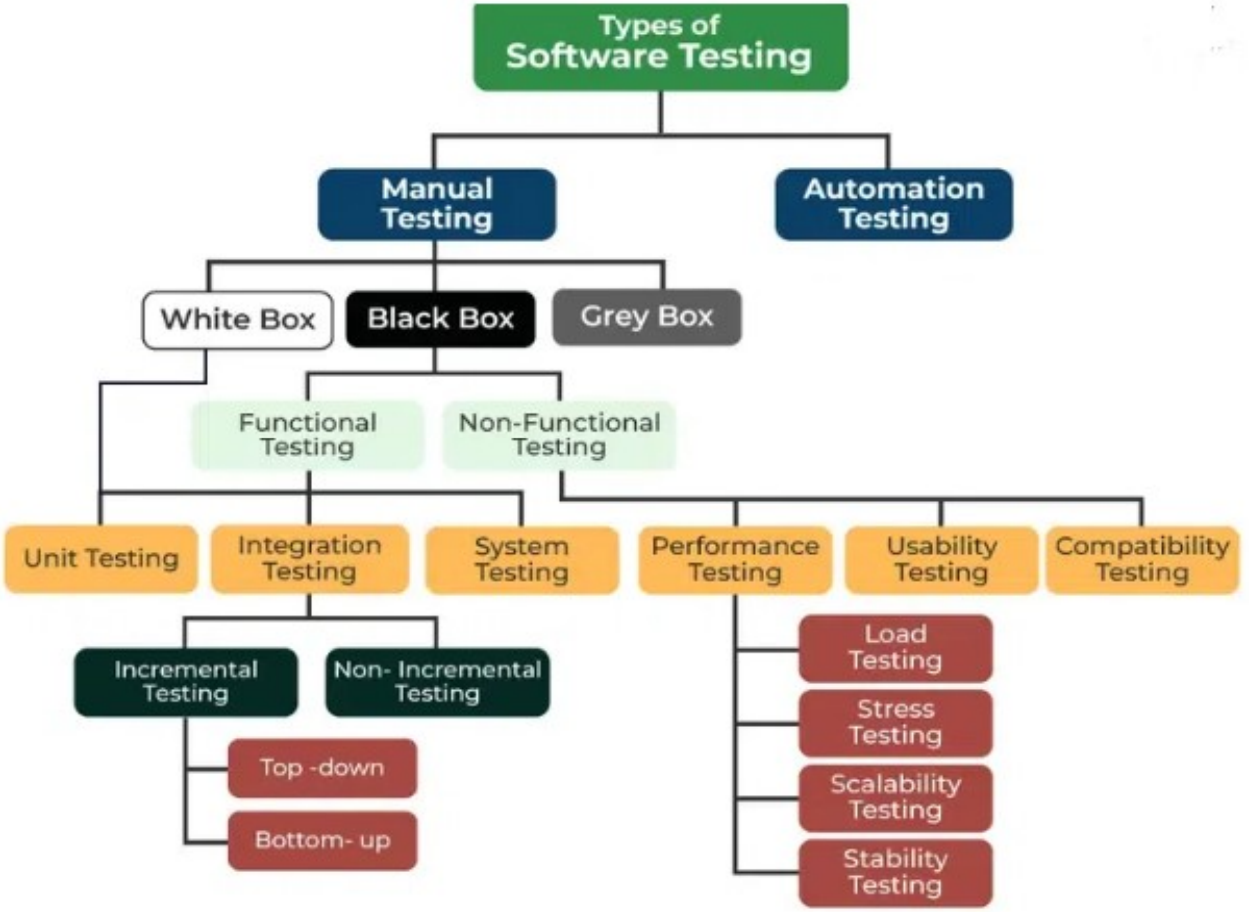


Validation

- Validation testing is testing where tester performed functional and non-functional testing.
- Here **functional testing** includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and **non-functional** testing includes User acceptance testing (UAT).
- Validation testing is also known as dynamic testing, where we are ensuring that **"we have developed the product right."**
- And it also checks that the software meets the business needs of the client.



Types of software testing



Manual Testing

- Manual testing is a technique to test the software that is carried out using the functions and features of an application.
- In manual software testing, a tester tests the software by following predefined test cases.
- In this testing, testers make test cases for the codes, test the software, and give the final report about that software.
- Manual testing is time-consuming because it is done by humans, and there is a chance of human errors.

Automation Testing

- It is also known as Test Automation, is when the tester writes scripts and uses another software to test the product.
- This process involves the automation of a manual process.
- Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.

- **Black box Testing:**
  - Testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without any concern with the internal logical structure of the software known as black-box testing.
- **White box Testing:**
  - Testing in which the tester is aware of the internal workings of the product, has access to its source code, and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.
- **Grey Box Testing:**
  - Testing in which the testers should have knowledge of implementation, however, they need not be experts.

Different Levels of Software Testing

- Software level testing can be majorly classified into 4 levels:
- **Unit Testing**: It a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
- **Integration Testing**: It is a level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
- **System Testing**: It is a level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
- **Acceptance Testing**: It is a level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.