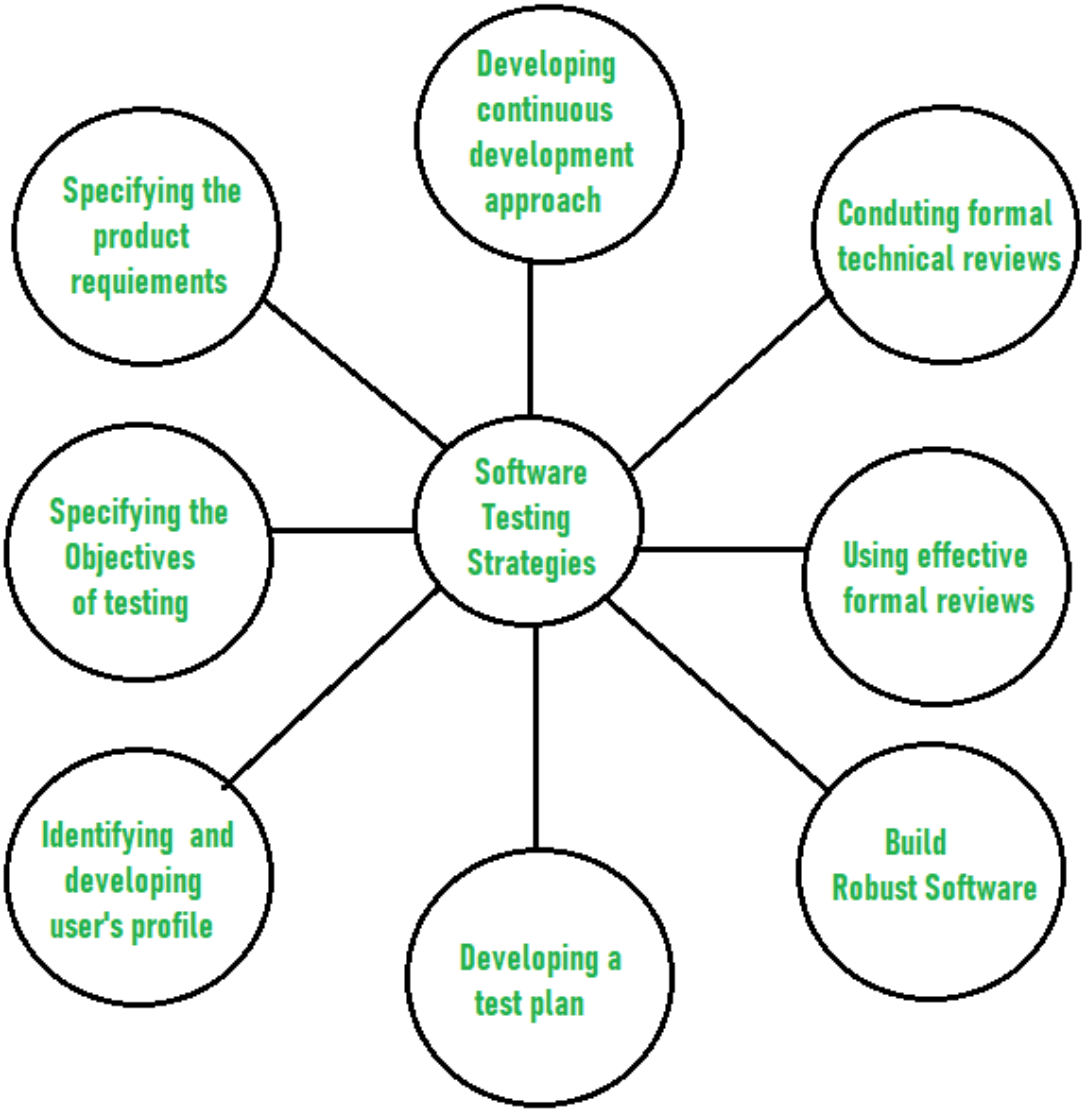
9. **Performance testing** – Tests the software to determine its performance characteristics such as speed, scalability, and stability.
10. **Security testing** – Tests the software to identify vulnerabilities and ensure it meets security requirements.

Software Testing is a type of investigation to find out if there is any default or error present in the software so that the errors can be reduced or removed to increase the quality of the software and to check whether it fulfills the specifies requirements or not. According to Glen Myers, software testing has the following objectives:

- The process of investigating and checking a program to find whether there is an error or not and does it fulfill the requirements or not is called testing.
- When the number of errors found during the testing is high, it indicates that the testing was good and is a sign of good test case.
- Finding an unknown error that wasn't discovered yet is a sign of a successful and a good test case.

The main objective of software testing is to design the tests in such a way that it systematically finds different types of errors without taking much time and effort so that less time is required for the development of the software. The overall strategy for testing software includes:



1. **Before testing starts, it's necessary to identify and specify the requirements of the product in a quantifiable manner.** Different characteristics quality of the software is there such as maintainability that means the ability to update and modify, the probability that means to find and estimate any risk, and usability that means how it can easily be used by the customers or end-users. All these characteristic qualities should be specified in a particular order to obtain clear test results without any error.
2. **Specifying the objectives of testing in a clear and detailed manner.** Several objectives of testing are there such as effectiveness that means how effectively the software can achieve the target, any failure that means inability to fulfill the requirements and perform functions, and the cost of defects or errors that mean the cost required to fix the error. All these objectives should be clearly mentioned in the test plan.
3. **For the software, identifying the user's category and developing a profile for each user.** Use cases describe the interactions and communication among different classes of users

and the system to achieve the target. So as to identify the actual requirement of the users and then testing the actual use of the product.

4. **Developing a test plan to give value and focus on rapid-cycle testing.** Rapid Cycle Testing is a type of test that improves quality by identifying and measuring the any changes that need to be required for improving the process of software. Therefore, a test plan is an important and effective document that helps the tester to perform rapid cycle testing.
5. **Robust software is developed that is designed to test itself.** The software should be capable of detecting or identifying different classes of errors. Moreover, software design should allow automated and regression testing which tests the software to find out if there is any adverse or side effect on the features of software due to any change in code or program.
6. **Before testing, using effective formal reviews as a filter.** Formal technical reviews is technique to identify the errors that are not discovered yet. The effective technical reviews conducted before testing reduces a significant amount of testing efforts and time duration required for testing software so that the overall development time of software is reduced.
7. **Conduct formal technical reviews to evaluate the nature, quality or ability of the test strategy and test cases.** The formal technical review helps in detecting any unfilled gap in the testing approach. Hence, it is necessary to evaluate the ability and quality of the test strategy and test cases by technical reviewers to improve the quality of software.
8. **For the testing process, developing a approach for the continuous development.** As a part of a statistical process control approach, a test strategy that is already measured should be used for software testing to measure and control the quality during the development of software.

Advantages or Disadvantages:

Advantages of software testing:

1. Improves software quality and reliability – Testing helps to identify and fix defects early in the development process, reducing the risk of failure or unexpected behavior in the final product.
2. Enhances user experience – Testing helps to identify usability issues and improve the overall user experience.
3. Increases confidence – By testing the software, developers and stakeholders can have confidence that the software meets the requirements and works as intended.
4. Facilitates maintenance – By identifying and fixing defects early, testing makes it easier to maintain and update the software.
5. Reduces costs – Finding and fixing defects early in the development process is less expensive than fixing them later in the life cycle.

Disadvantages of software testing:

1. Time-consuming – Testing can take a significant amount of time, particularly if thorough testing is performed.
2. Resource-intensive – Testing requires specialized skills and resources, which can be expensive.
3. Limited coverage – Testing can only reveal defects that are present in the test cases, and it is possible for defects to be missed.

4. Unpredictable results – The outcome of testing is not always predictable, and defects can be hard to replicate and fix.
5. Delays in delivery – Testing can delay the delivery of the software if testing takes longer than expected or if significant defects are identified.

Test Strategies for Conventional Software:

-

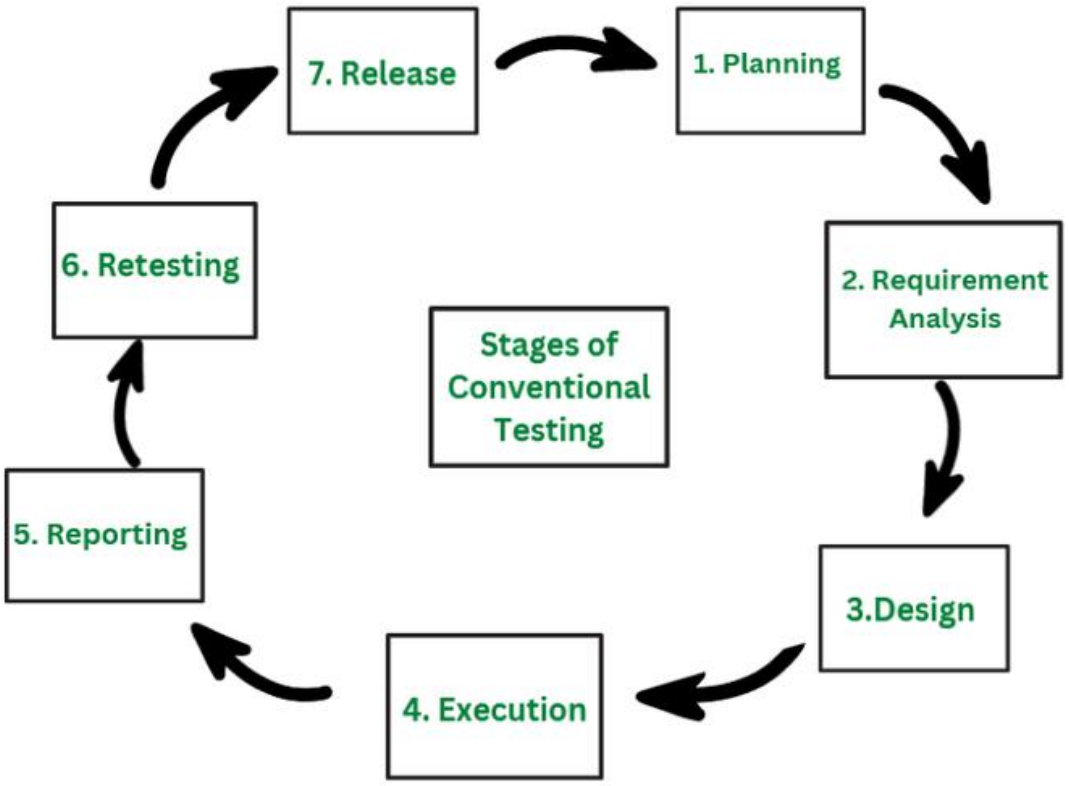
Conventional testing also known as the Traditional approach of software testing involves a series of activities that aim to identify the defects in the software and ensures that the software meets the specified requirements. The article focuses on discussing Conventional testing in detail.

What is Conventional Testing?

Conventional testing is defined as traditional testing where the main aim is to check whether all the requirements stated by the user are achieved.

- The difference between conventional testing and other testing approach is that it concentrates on checking all the requirements given by the user rather than following a software development life cycle.
- Conventional testing mainly focuses on functional testing.
- This testing is being performed by a dedicated team of software testers.

Stages of Conventional Testing



Conventional testing follows a sequential approach. It consists of various stages. Such as

1. Planning

Planning is the first stage of conventional testing. This stage consists of planning regarding the objective of testing developing a complete test plan and resources that will be required for performing testing.

2. Requirement Analysis

Software Requirements are being analyzed in this phase. These requirements help to identify the scope of testing and risks and for the preparation of test cases.

3. Design

In this stage, test cases are designed. If the test cases are successful it means that test cases are achieved. If not test cases are failed to achieve.

4. Execution

Execution is the process where test cases are executed. The errors encountered during execution are documented.

5. Reporting

In reporting phase, all the documented errors are sent to the development team for fixing.

6. Retesting

Retesting is the stage where all the test cases are performed again. It checks whether all the failed test cases meet. All the requirements specified by the user are achieved.

7. Release

In the last stage, the software is released for the users. It is verified that all the requirements stated by the user or client are successfully working before the release of the software product.

Types of Conventional Testing

1. Unit Testing

Unit Testing is defined as a type of testing where the various modules and units are being tested individually. Unit testing makes sure that each individual component of the system works well and eventually checks whether all the requirements stated by clients are achieved successfully.

2. Integration Testing

Integration Testing is defined as a type of testing where multiple modules or components are tested together in order to check that they work accordingly once integrated with each other. It makes sure that interaction and communication between different modules work well.

3. Performance Testing

Performance Testing is defined as a type of testing that checks for performance-related parameters for a software product. Performance testing helps to find out the loopholes in the system and improve performance.

4. Acceptance Testing

Acceptance Testing is defined as a type of testing that is used to check the requirements according to the user's point of view. It makes sure that all requirements specified by the user are achieved.

5. Regression Testing

Regression Testing is defined as a type of testing in which test cases are executed again in order to check that the changes made are being fixed and the system is working accordingly.

Benefits of Conventional Testing

1. **Cost Effective:** Conventional Testing is cost-effective as manual testing is being used. Manual testing requires less financial investment as compared to automation testing.
2. **Flexible:** Conventional testing has the advantage of flexibility. Manual testing has the ability to adopt the changes that take place while testing the product.

3. **Testing of Non-functional Requirements also:** Manual testing can test functional as well as non-functional requirements such as accessibility, and usability which is different from automation testing.
4. **Understanding User Experience more effectively:** Conventional Testing makes use of manual testing which helps to understand the user experience more effectively as the manual tester can test the requirements with multiple scenarios.
5. **Provides better communication between testers and developers:** Manual testing allows better communication between testers, developers, and other clients regarding issues and wrong outputs.

Limitations of Conventional Testing

1. **Time-consuming:** Conventional Testing can be time-consuming as with manual testing it can take more time for large applications and accordingly delay further deployment process of the project.
2. **Subjective:** The manual tester performing the testing can have their own views and opinions which can in turn result in the quality of testing that is being performed.
3. **Repetitive:** Manual testing can lead to repetition by performing the testing for the same test cases. It can consume more time than it is required.
4. **Limited Coverage:** Manual testing can miss some of the test cases and it will be not notified by the tester. This can result in delivering the software product with errors or un tasted test cases.

White box Testing – Software Engineering

- **White box testing** techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.

What is 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. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.

Before we move in depth of the white box testing do you known that their are many different type of testing used in industry and some automation testing tools are there which automate the most of testing so if you wish to learn the latest industry level tools then you check-out our manual to automation testing course in which you will learn all these concept and tools

What Does White Box Testing Focus On?

White box testing uses detailed knowledge of a software's inner workings to create very specific test cases.

- **Path Checking:** Examines the different routes the program can take when it runs. Ensures that all decisions made by the program are correct, necessary, and efficient.

- **Output Validation:** Tests different inputs to see if the function gives the right output each time.
- **Security Testing:** Uses techniques like static code analysis to find and fix potential security issues in the software. Ensures the software is developed using secure practices.
- **Loop Testing:** Checks the loops in the program to make sure they work correctly and efficiently. Ensures that loops handle variables properly within their scope.
- **Data Flow Testing:** Follows the path of variables through the program to ensure they are declared, initialized, used, and manipulated correctly.

Types Of White Box Testing

White box testing can be done for different purposes. The three main types are:

1. Unit Testing
2. Integration Testing
3. Regression 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

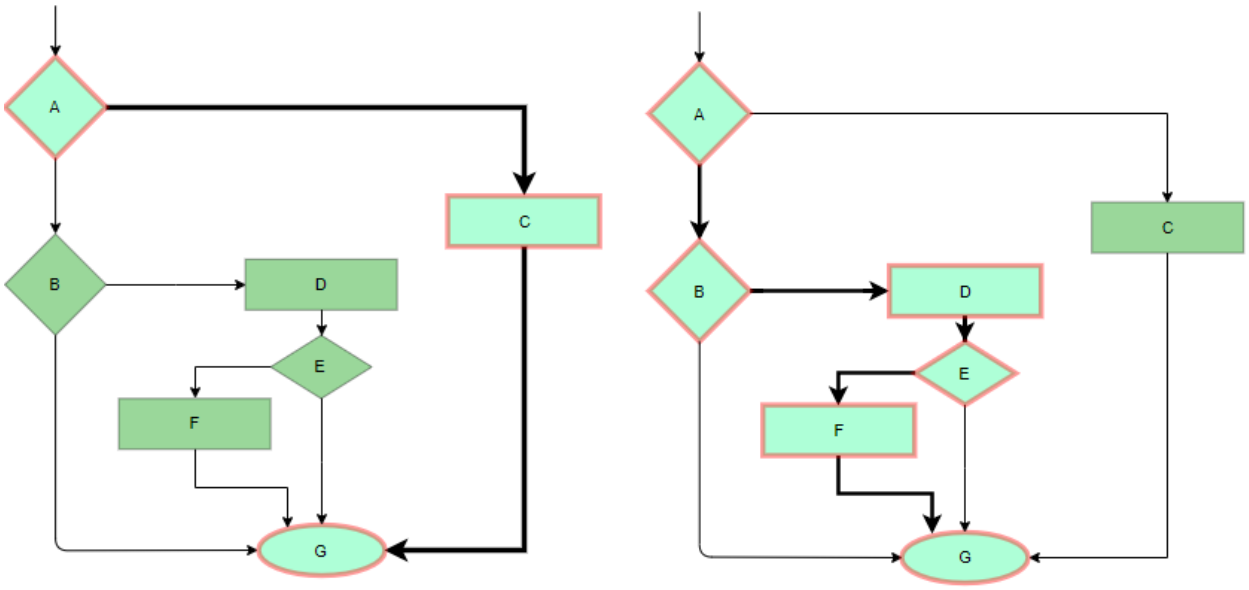
- Verifies that changes or updates don't break existing functionality.
- Ensures the application still passes all existing tests after updates.

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

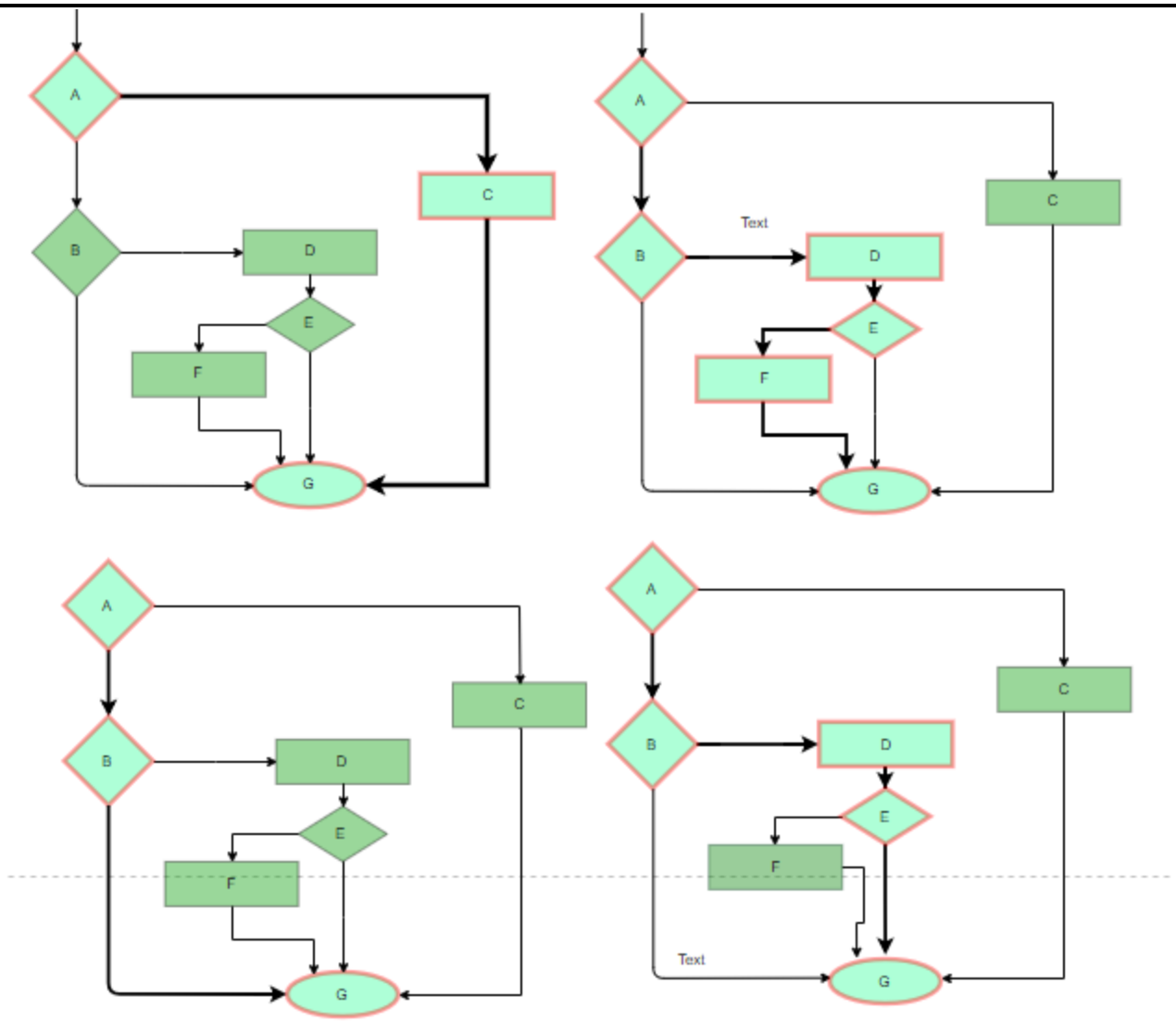
In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.



Statement Coverage Example

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.



4 test cases are required such that all branches of all decisions are covered, i.e, all edges of the flowchart are covered

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

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 cy clomatic 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

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:

1. Skip the loop entirely
2. Only one pass through the loop
3. 2 passes
4. m passes, where m < n
5. 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.

Black Box vs White Box vs Gray Box Testing

Here is a simple comparison of Black Box, White Box, and Gray Box testing, highlighting key aspects:

Aspect	Black Box Testing	White Box Testing	Gray Box Testing
Knowledge of Internal Code	Not required	Required	Partially required

Aspect	Black Box Testing	White Box Testing	Gray Box Testing
Other Names	Functional testing, data-driven testing, closed box testing	Structural testing, clear box testing, code-based testing, transparent testing	Translucent testing
Approach	Trial and error, based on external functionality	Verification of internal coding, system boundaries, and data domains	Combination of both black box and white box approaches
Test Case Input Size	Largest	Smaller compared to Black Box	Smaller than both Black Box and White Box
Finding Hidden Errors	Difficult	Easier due to internal code access	Challenging, may be found at user level
Algorithm Testing	Not suitable	Well-suited and recommended	Not suitable
Time Consumption	Depends on functional specifications	High due to complex code analysis	Moderate, faster than White Box

Process of White Box Testing

1. **Input:** Requirements, Functional specifications, design documents, source code.
2. **Processing:** Performing risk analysis to guide through the entire process.
3. **Proper test planning:** Designing test cases to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
4. **Output:** Preparing the final report of the entire testing process.

White Testing is performed in 2 Steps

1. Tester should understand the code well
2. Tester should write some code for test cases and execute them

Tools required for White box testing:

- Py Unit
- Sql map
- N map
- Para soft J test
- N unit
- Vera Unit
- Cpp Unit

- Bug zilla
- Fiddler
- JSUnit.net
- Open Grok
- Wire shark
- HP Fortify
- CS Unit

Features of White box Testing

1. **Code coverage analysis:** White box testing helps to analyze the code coverage of an application, which helps to identify the areas of the code that are not being tested.
2. **Access to the source code:** White box testing requires access to the application's source code, which makes it possible to test individual functions, methods, and modules.
3. **Knowledge of programming languages:** Testers performing white box testing must have knowledge of programming languages like Java, C++, Python, and PHP to understand the code structure and write tests.
4. **Identifying logical errors:** White box testing helps to identify logical errors in the code, such as infinite loops or incorrect conditional statements.
5. **Integration testing:** White box testing is useful for integration testing, as it allows testers to verify that the different components of an application are working together as expected.
6. **Unit testing:** White box testing is also used for unit testing, which involves testing individual units of code to ensure that they are working correctly.
7. **Optimization of code:** White box testing can help to optimize the code by identifying any performance issues, redundant code, or other areas that can be improved.
8. **Security testing:** White box testing can also be used for security testing, as it allows testers to identify any vulnerabilities in the application's code.
9. **Verification of Design:** It verifies that the software's internal design is implemented in accordance with the designated design documents.
10. **Check for Accurate Code:** It verifies that the code operates in accordance with the guidelines and specifications.
11. **Identifying Coding Mistakes:** It finds and fix programming flaws in your code, including syntactic and logical errors.
12. **Path Examination:** It ensures that each possible path of code execution is explored and test various iterations of the code.
13. **Determining the Dead Code:** It finds and remove any code that isn't used when the programme is running normally (dead code).

Advantages of White Box Testing

1. **Thorough Testing** : White box testing is thorough as the entire code and structures are tested.
2. **Code Optimization:** It results in the optimization of code removing errors and helps in removing extra lines of code.
3. **Early Detection of Defects:** It can start at an earlier stage as it doesn't require any interface as in the case of black box testing.
4. **Integration with SDLC:** White box testing can be easily started in Software Development Life Cycle.
5. **Detection of Complex Defects:** Testers can identify defects that cannot be detected through other testing techniques.

6. **Comprehensive Test Cases:** Testers can create more comprehensive and effective test cases that cover all code paths.
7. Testers can ensure that the code meets coding standards and is optimized for performance.

Disadvantages of White Box Testing

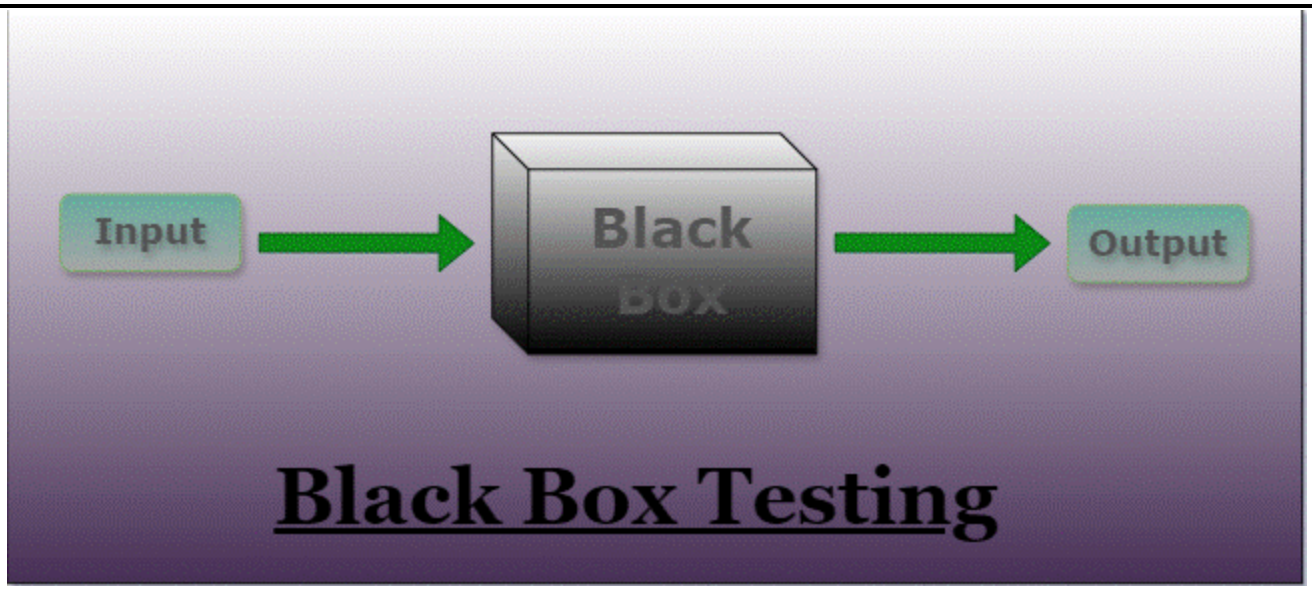
1. **Programming Knowledge and Source Code Access:** Testers need to have programming knowledge and access to the source code to perform tests.
2. **Overemphasis on Internal Workings:** Testers may focus too much on the internal workings of the software and may miss external issues.
3. **Bias in Testing:** Testers may have a biased view of the software since they are familiar with its internal workings.
4. **Test Case Overhead:** Redesigning code and rewriting code needs test cases to be written again.
5. **Dependency on Tester Expertise:** Testers are required to have in-depth knowledge of the code and programming language as opposed to black-box testing.
6. **Inability to Detect Missing Functionalities:** Missing functionalities cannot be detected as the code that exists is tested.
7. **Increased Production Errors:** High chances of errors in production.

Black Box Testing – Software Engineering

- Black Box Testing is an important part of making sure software works as it should. Instead of peeking into the code, testers check how the software behaves from the outside, just like users would. This helps catch any issues or bugs that might affect how the software works.
  This simple guide gives you an overview of what Black Box Testing is all about and why it matters in software development.

What is 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.



Black Box Testing

Types Of Black Box Testing

The following are the several categories of black box testing:

1. **Functional Testing**
2. **Regression Testing**
3. **Nonfunctional Testing (NFT)**

Before we move in depth of the Black box testing do you known that their are many different type of testing used in industry and some automation testing tools are there which automate the most of testing so if you wish to learn the latest industry level tools then you check-out our manual to automation testing course in which you will learn all these concept and tools

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.

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.

Nonfunctional Testing

- Non-functional testing is a software testing technique that checks the non-functional attributes of the system.
- Non-functional testing is defined as a type of software testing to check non-functional aspects of a software application.
- It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.
- Non-functional testing is as important as functional testing.
- Non-functional testing 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.

Black Box and White Box Testing

Black box testing is a testing technique in which the internal workings of the software are not known to the tester. The tester only focuses on the input and output of the software. Whereas, White box testing is a testing technique in which the tester has knowledge of the internal workings of the software, and can test individual code snippets, algorithms and methods.

- **Testing objectives:** Black box testing is mainly focused on testing the functionality of the software, ensuring that it meets the requirements and specifications. White box testing is mainly focused on ensuring that the internal code of the software is correct and efficient.
- **Knowledge level** : Black box testing does not require any knowledge of the internal workings of the software, and can be performed by testers who are not familiar with programming languages. White box testing requires knowledge of programming languages, software architecture and design patterns.
- **Testing methods:** Black box testing uses methods like equivalence partitioning, boundary value analysis, and error guessing to create test cases. Whereas, white box testing uses methods like control flow testing, data flow testing and statement coverage.
- **Scope** : Black box testing is generally used for testing the software at the functional level. White box testing is used for testing the software at the unit level, integration level and system level.

Grey Box Testing

Gray Box Testing is a software testing technique that is a combination of the **Black Box Testing** technique and the **White Box Testing** technique.

1. In the Black Box Testing technique, the tester is unaware of the internal structure of the item being tested and in White Box Testing the internal structure is known to the tester.
2. The internal structure is partially known in Gray Box Testing.
3. This includes access to internal data structures and algorithms to design the test cases.
4. Gray Box Testing is named so because the software program is like a semitransparent or gray box inside which the tester can partially see.
5. It commonly focuses on context-specific errors related to web systems.

Objectives of Gray Box Testing

- **To provide combined advantages of both black box testing and white box testing.**
- **To combine the input of developers as well as testers.**
- **To improve overall product quality.**

Ways of Black Box Testing Done

1. Syntax-Driven Testing – This type of testing is applied to systems that can be syntactically represented by some language. For example, language can be represented by context-free grammar. In this, the test cases are generated so that each grammar rule is used at least once.

2. Equivalence partitioning – It is often seen that many types of inputs work similarly so instead of giving all of them separately we can group them and test only one input of each group. The idea is to partition the input domain of the system into several equivalence classes such that each member of the class works similarly, i.e., if a test case in one class results in some error, other members of the class would also result in the same error.

The technique involves two steps:

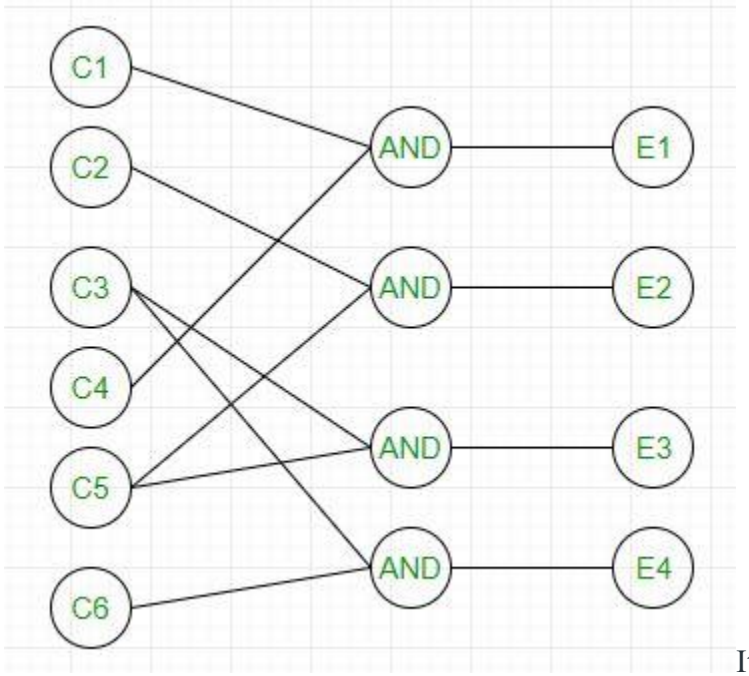
1. **Identification of equivalence class –** Partition any input domain into a minimum of two sets: **valid values** and **invalid values** . For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.
2. **Generating test cases –** (i) To each valid and invalid class of input assign a unique identification number. (ii) Write a test case covering all valid and invalid test cases considering that no two invalid inputs mask each other. To calculate the square root of a number, the equivalence classes will be **(a) Valid inputs:**
   - The whole number which is a perfect square-output will be an integer.
   - The entire number which is not a perfect square-output will be a decimal number.
   - Positive decimals
   - Negative numbers(integer or decimal).
   - Characters other than numbers like “a”,”!”,”;”, etc.

3. Boundary value analysis – Boundaries are very good places for errors to occur. Hence, if test cases are designed for boundary values of the input domain then the efficiency of testing improves and the probability of finding errors also increases. For example – If the valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.

4. Cause effect graphing – This technique establishes a relationship between logical input called causes with corresponding actions called the effect. The causes and effects are represented using Boolean graphs. The following steps are followed:

1. Identify inputs (causes) and outputs (effect).
2. Develop a cause-effect graph.
3. Transform the graph into a decision table.
4. Convert decision table rules to test cases.

For example, in the following cause graph:



It can be converted into a decision table like:

		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	x	-	-	-
	E2	-	x	-	-
	E3	-	-	x	-
	E4	-	-	-	x

Each column corresponds to a rule which will become a test case for testing. So there will be 4 test cases.

5. Requirement-based testing – It includes validating the requirements given in the SRS of a software system.

6. Compatibility testing – The test case results not only depends on the product but is also on the infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly. Some parameters that generally affect the compatibility of software are:

1. Processor (Pentium 3, Pentium 4) and several processors.
2. Architecture and characteristics of machine (32-bit or 64-bit).
3. Back-end components such as database servers.
4. Operating System (Windows, Linux, etc).

Tools Used for Black Box Testing:

1. **Appium**

2. **Selenium**
3. **Microsoft Coded UI**
4. **Applitools**
5. **HP QTP .**

What can be identified by Black Box Testing

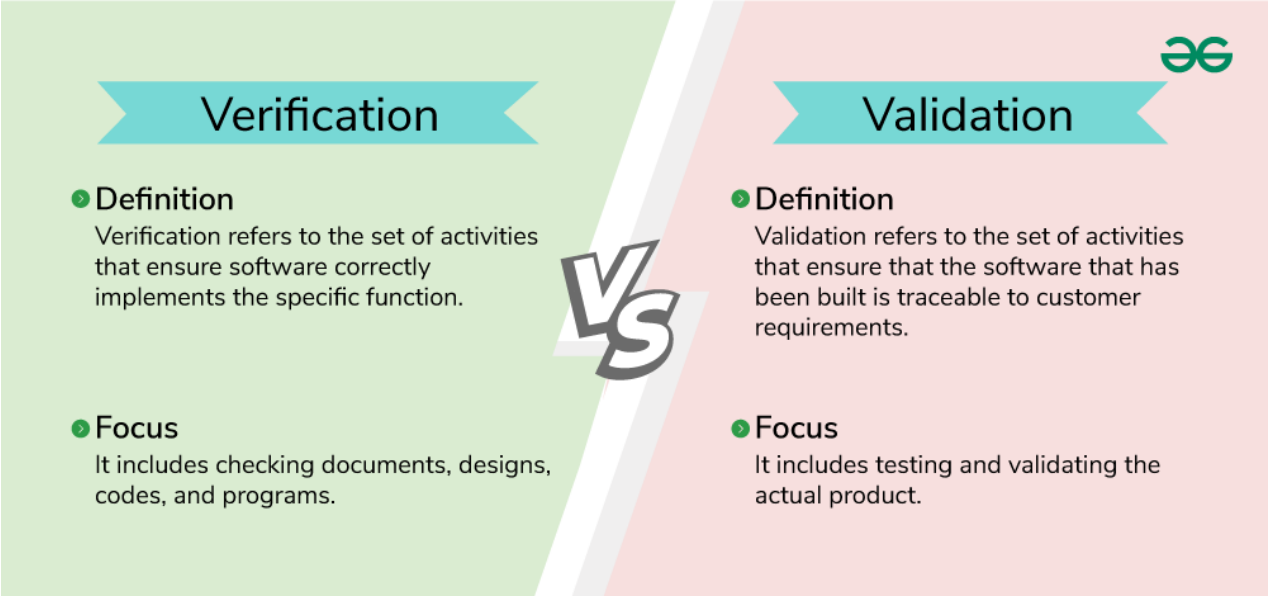
1. Discovers missing functions, incorrect function & interface errors
2. Discover the errors faced in accessing the database
3. Discovers the errors that occur while initiating & terminating any functions.
4. Discovers the errors in performance or behaviour of software.

Features of black box testing

1. **Independent testing:** Black box testing is performed by testers who are not involved in the development of the application, which helps to ensure that testing is unbiased and impartial.
2. **Testing from a user's perspective:** Black box testing is conducted from the perspective of an end user, which helps to ensure that the application meets user requirements and is easy to use.
3. **No knowledge of internal code:** Testers performing black box testing do not have access to the application's internal code, which allows them to focus on testing the application's external behaviour and functionality.
4. **Requirements-based testing:** Black box testing is typically based on the application's requirements, which helps to ensure that the application meets the required specifications.
5. **Different testing techniques:** Black box testing can be performed using various testing techniques, such as functional testing, usability testing, acceptance testing, and regression testing.
6. **Easy to automate:** Black box testing is easy to automate using various automation tools, which helps to reduce the overall testing time and effort.
7. **Scalability:** Black box testing can be scaled up or down depending on the size and complexity of the application being tested.
8. **Limited knowledge of application:** Testers performing black box testing have limited knowledge of the application being tested, which helps to ensure that testing is more representative of how the end users will interact with the application.

Differences between Verification and Validation testing:

- **Verification and Validation** is the process of investigating whether a software system satisfies specifications and standards and fulfills the required purpose. Verification and Validation both play an important role in developing good software development. Verification helps in examining whether the product is built right according to requirements, while validation helps in examining whether the right product is built to meet user needs. In this article, we will learn the difference between Verification and Validation.



Differences between Verification and Validation

What is Verification?

Verification is the process of checking that software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing. Verification means **Are we building the product right?**

What is Validation?

Validation is the process of checking whether the software product is up to the mark or in other words product has high-level requirements. It is the process of checking the validation of the product i.e. it checks what we are developing is the right product. It is validation of the actual and expected products. Validation is dynamic testing. Validation means **Are we building the right product?**

Differences between Verification and Validation

	Verification	**Validation**
Definition	Verification refers to the set of activities that ensure software correctly implements the specific function	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.
Focus	It includes checking documents, designs, codes, and programs.	It includes testing and validating the actual product.
Type of Testing	Verification is the static testing.	Validation is dynamic testing.
Execution	It does *not* include the execution of the code.	It includes the execution of the code.

	Verification	Validation
Methods Used	Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
Purpose	It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
Bug	It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
Goal	The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Responsibility	Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
Timing	It comes before validation.	It comes after verification.
Human or Computer	It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.
Lifecycle	After a valid and complete specification the verification starts.	Validation begins as soon as project starts.
Error Focus	Verification is for prevention of errors.	Validation is for detection of errors.
Another Terminology	Verification is also termed as white box testing or static testing as work product goes through reviews.	Validation can be termed as black box testing or dynamic testing as work product is executed.
Performance	Verification finds about 50 to 60% of the defects.	Validation finds about 20 to 30% of the defects.

	Verification	Validation
Stability	Verification is based on the opinion of reviewer and may change from person to person.	Validation is based on the fact and is often stable.

Real-World Example of Verification vs Validation

- **Verification Example:** Imagine a team is developing a new mobile banking app. During the verification phase, they review the requirements and design documents. They check if all the specified features like fund transfer, account balance check, and transaction history are included and correctly detailed in the design. They also perform peer reviews and inspections to ensure the design aligns with the requirements. This step ensures that the app is being built according to the initial plan and specifications without actually running the app.
- **Validation Example:** In the validation phase, the team starts testing the mobile banking app on actual devices. They check if users can log in, transfer money, and view their transaction history as intended. Testers perform usability tests to ensure the app is user-friendly and functional tests to ensure all features work correctly. They might also involve real users to provide feedback on the app's performance. This phase ensures that the app works as expected and meets user needs in real-world scenarios.

Advantages of Differentiating Verification and Validation

Differentiating between verification and validation in software testing offers several advantages:

1. **Clear Communication:** It ensures that team members understand which aspects of the software development process are focused on checking requirements (verification) and which are focused on ensuring functionality (validation).
2. **Efficiency:** By clearly defining verification as checking documents and designs without executing code, and validation as testing the actual software for functionality and usability, teams avoid redundant efforts and streamline their testing processes.
3. **Minimized Errors:** It reduces the chances of overlooking critical requirements or functionalities during testing, leading to a more thorough evaluation of the software's capabilities.
4. **Cost Savings:** Optimizing resource allocation and focusing efforts on the right testing activities based on whether they fall under verification or validation helps in managing costs effectively.
5. **Client Satisfaction:** Ensuring that software meets or exceeds client and user expectations by conducting both verification and validation processes rigorously improves overall software quality and user satisfaction.
6. **Process Improvement:** By distinguishing between verification and validation, organizations can refine their testing methodologies, identify areas for improvement, and enhance the overall software development lifecycle.

In essence, clear differentiation between verification and validation in software testing contributes to a more structured, efficient, and successful software development process.

System Testing – Software Engineering

- System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified

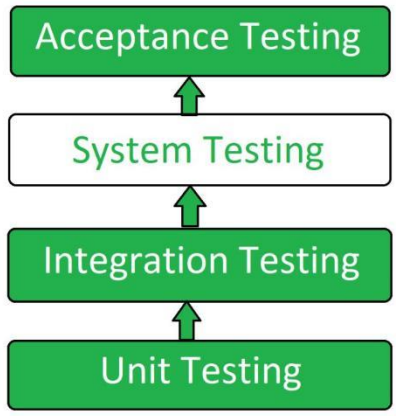
requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.

What is System Testing ?

System Testing is a type of software testing that is performed on a completely integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input.

- The goal of integration testing is to detect any irregularity between the units that are integrated. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.
- **System Testing** is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or the context of both. System testing tests the design and behavior of the system and also the expectations of the customer.
- It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS) . System Testing is performed by a testing team that is independent of the development team and helps to test the quality of the system impartial.
- It has both functional and non-functional testing. **System Testing is a black-box testing** . System Testing is performed after the integration testing and before the acceptance testing.

System testing is evergreen role in software engineering because every software is needed to test and very update is needed to test so the demand of the software tester is always needed. If you wish to learn software testing from the scratch and want to grab a good grip on testing tools and concept you can check our new software testing course



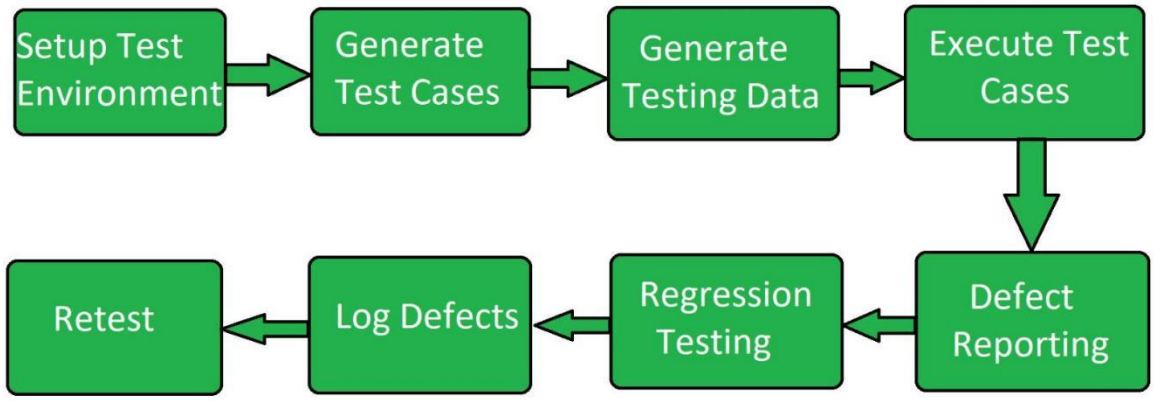
System Testing

System Testing Process

System Testing is performed in the following steps:

- **Test Environment Setup:** Create testing environment for the better quality testing.
- **Create Test Case:** Generate test case for the testing process.
- **Create Test Data:** Generate the data that is to be tested.
- **Execute Test Case:** After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:** Defects in the system are detected.
- **Regression Testing:** It is carried out to test the side effects of the testing process.
- **Log Defects:** Defects are fixed in this step.

- **Retest:** If the test is not successful then again test is performed.



System Testing Process

Types of System Testing

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Tools used for System Testing

1. J Meter
2. Gallen Framework
3. HP Quality Center/ALM
4. IBM Rational Quality Manager
5. Microsoft Test Manager
6. Selenium
7. Appium
8. Load Runner
9. Gatling
10. J Meter
11. Apache JServ
12. Soap UI

Note: *The choice of tool depends on various factors like the technology used, the size of the project, the budget, and the testing requirements.*

Advantages of System Testing

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.

- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing
- Verifies the overall functionality of the system.
- Detects and identifies system-level problems early in the development cycle.
- Helps to validate the requirements and ensure the system meets the user needs.
- Improves system reliability and quality.
- Facilitates collaboration and communication between development and testing teams.
- Enhances the overall performance of the system.
- Increases user confidence and reduces risks.
- Facilitates early detection and resolution of bugs and defects.
- Supports the identification of system-level dependencies and inter-module interactions.
- Improves the system's maintainability and scalability.

Disadvantages of System Testing

- This testing is time consuming process than another testing techniques since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of entire software.
- It needs good debugging tool otherwise the hidden errors will not be found.
- Can be time-consuming and expensive.
- Requires adequate resources and infrastructure.
- Can be complex and challenging, especially for large and complex systems.
- Dependent on the quality of requirements and design documents.
- Limited visibility into the internal workings of the system.
- Can be impacted by external factors like hardware and network configurations.
- Requires proper planning, coordination, and execution.
- Can be impacted by changes made during development.
- Requires specialized skills and expertise.
- May require multiple test cycles to achieve desired results.

What is Debugging in Software Engineering?

-

Debugging in **Software Engineering** is the process of identifying and resolving **errors** or **bugs** in a software system. It's a critical aspect of software development, ensuring **quality**, **performance**, and **user satisfaction**. Despite being time-consuming, effective **debugging** is essential for reliable and competitive software products.

Here are we discussing the points related to Debugging in detail:

What is Debugging?

In the context of software engineering, debugging is the process of fixing a bug in the software. When there's a problem with software, programmers analyze the code to figure out why things aren't working correctly. They use different debugging tools to carefully go through the code, step by step, find the issue, and make the necessary corrections.

Process of Debugging

Debugging is a crucial skill in programming. Here's a **simple, step-by-step explanation** to help you understand and execute the **debugging process** effectively:

Step 1: Reproduce the Bug

- To start, you need to **recreate the conditions** that caused the bug. This means making the error happen again so you can see it firsthand.
- Seeing the bug in action helps you understand the problem better and gather important details for fixing it.

Step 2: Locate the Bug

- Next, **find where the bug is in your code**. This involves looking closely at your code and checking any error messages or logs.
- Developers often use debugging tools to help with this step.

Step 3: Identify the Root Cause

- Now, figure out **why the bug happened**. Examine the logic and flow of your code and see how different parts interact under the conditions that caused the bug.
- This helps you understand what went wrong.

Step 4: Fix the Bug

- Once you know the cause, **fix the code**. This involves making changes and then testing the program to ensure the bug is gone.
- Sometimes, you might need to try several times, as initial fixes might not work or could create new issues.
- Using a version control system helps track changes and undo any that don't solve the problem.

Step 5: Test the Fix

After fixing the bug, **run tests** to ensure everything works correctly. These tests include:

- **Unit Tests:** Check the specific part of the code that was changed.
- **Integration Tests:** Verify the entire module where the bug was found.
- **System Tests:** Test the whole system to ensure overall functionality.
- **Regression Tests:** Make sure the fix didn't cause any new problems elsewhere in the application.

Step 6: Document the Process

- Finally, **record what you did**. Write down what caused the bug, how you fixed it, and any other important details.
- This documentation is helpful if similar issues occur in the future.

Why is debugging important?

Fixing mistakes in computer programming, known as bugs or errors, is necessary because programming deals with abstract ideas and concepts. Computers understand machine language, but we use programming languages to make it easier for people to talk to computers. Software has many layers of abstraction, meaning different parts must work together for an application to function properly. When errors happen, finding and fixing them can be tricky. That's where debugging tools and strategies come in handy. They help solve problems faster, making developers more efficient. This not only improves the quality of the software but also makes the experience better for the people using it. In simple terms, debugging is important because it makes sure the software works well and people have a good time using it.

Debugging Approaches/Strategies

1. **Brute Force:** Study the system for a longer duration to understand the system. It helps the debugger to construct different representations of systems to be debugged depending on the need. A study of the system is also done actively to find recent changes made to the software.
2. **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
3. **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
4. **Using** A debugging **experience** with the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
5. **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.
6. **Static analysis:** Analyzing the code without executing it to identify potential bugs or errors. This approach involves analyzing code syntax, data flow, and control flow.
7. **Dynamic analysis:** Executing the code and analyzing its behavior at runtime to identify errors or bugs. This approach involves techniques like runtime debugging and profiling.
8. **Collaborative debugging:** Involves multiple developers working together to debug a system. This approach is helpful in situations where multiple modules or components are involved, and the root cause of the error is not clear.
9. **Logging and Tracing:** Using logging and tracing tools to identify the sequence of events leading up to the error. This approach involves collecting and analyzing logs and traces generated by the system during its execution.
10. **Automated Debugging:** The use of automated tools and techniques to assist in the debugging process. These tools can include static and dynamic analysis tools, as well as tools that use machine learning and artificial intelligence to identify errors and suggest fixes.

Examples of error during debugging

Some common example of error during debugging are:

- Syntax error
- Logical error
- Runtime error
- Stack overflow
- Index Out of Bound Errors
- Infinite loops
- Concurrency Issues
- I/O errors
- Environment Dependencies
- Integration Errors
- Reference error
- Type error

Debugging Tools

Debugging tools are essential for software development, helping developers locate and fix coding errors efficiently. With the rapid growth of software applications, the demand for advanced debugging tools has increased significantly. Companies are investing heavily in these tools, and researchers are developing innovative solutions to enhance debugging capabilities, including AI-driven debuggers and autonomous debugging for specialized applications.

Debugging tools vary in their functionalities, but they generally provide command-line interfaces to help developers identify and resolve issues. Many also offer remote debugging features and tutorials, making them accessible to beginners. Here are some of the most commonly used debugging tools:

1. Integrated Development Environments (IDEs)

IDEs like Visual Studio, Eclipse, and Py Charm offer features for software development, including built-in debugging tools. These tools allow developers to:

- Execute code line-by-line (**step debugging**)
- Stop program execution at specific points (**breakpoints**)
- Examine the state of variables and memory

IDEs support many programming languages and scripting languages, often through open-source plugins.

2. Standalone Debuggers

Standalone debuggers like GDB (GNU Debugger) provide advanced debugging features:

- **Conditional breakpoints** and watchpoints
- **Reverse debugging** (running a program backwards)

These tools are powerful but have a steeper learning curve compared to IDE debuggers.

3. Logging Utilities

Logging utilities log a program's state at various points in the code, which can then be analyzed to find problems. Logging is particularly useful for debugging issues that only occur in production environments.

4. Static Code Analyzers

Static code analysis tools examine code without executing it to find potential errors and deviations from coding standards. They focus on the semantics of the source code, helping developers catch common mistakes and maintain consistent coding styles.

5. Dynamic Analysis Tools

Dynamic analysis tools monitor software as it runs to detect issues like resource leaks or concurrency problems. These tools help catch bugs that static analysis might miss, such as memory leaks or buffer overflows.

6. Performance Profilers

Performance profilers help developers identify performance bottlenecks in their code. They measure:

- **CPU usage**
- **Memory usage**
- **I/O operations**

Difference Between Debugging and Testing

Debugging is different from testing. Testing focuses on finding bugs, errors, etc whereas debugging starts after a bug has been identified in the software. Testing is used to ensure that the program is correct and it was supposed to do with a certain minimum success rate. Testing can be manual or automated. There are several different types of testing unit testing, integration testing, alpha, and beta testing, etc.

Aspects	Testing	Debugging
Definition	Testing is the process to find bugs and errors.	Debugging is the process of correcting the bugs found during testing.
Purpose	The purpose of testing is to identify defects or errors in the software system	The purpose of debugging is to fix those defects or errors.
Focus	It is the process to identify the failure of implemented code.	It is the process to give absolution to code failure.
Timing	Testing is done before debugging	Debugging Differences between Testing and Debugging is done after testing
Approach	Testing involves executing the software system with test cases	Debugging involves analyzing the symptoms of a problem and identifying the root cause of the problem
Tools and Technique	Testing can involve using automated or manual testing tools	Debugging typically involves using tools and techniques such as logging, tracing, and code inspection.

Advantages of Debugging

Several advantages of debugging in software engineering:

1. **Improved system quality:** By identifying and resolving bugs, a software system can be made more reliable and efficient, resulting in improved overall quality.
2. **Reduced system downtime:** By identifying and resolving bugs, a software system can be made more stable and less likely to experience downtime, which can result in improved availability for users.
3. **Increased user satisfaction**: By identifying and resolving bugs, a software system can be made more user-friendly and better able to meet the needs of users, which can result in increased satisfaction.
4. **Reduced development costs:** Identifying and resolving bugs early in the development process, can save time and resources that would otherwise be spent on fixing bugs later in the development process or after the system has been deployed.
5. **Increased security**: By identifying and resolving bugs that could be exploited by attackers, a software system can be made more secure, reducing the risk of security breaches.
6. **Facilitates change:** With debugging, it becomes easy to make changes to the software as it becomes easy to identify and fix bugs that would have been caused by the changes.
7. **Better understanding of the system:** Debugging can help developers gain a better understanding of how a software system works, and how different components of the system interact with one another.

8. **Facilitates testing:** By identifying and resolving bugs, it makes it easier to test the software and ensure that it meets the requirements and specifications.

In summary, debugging is an important aspect of software engineering as it helps to improve system quality, reduce system downtime, increase user satisfaction, reduce development costs, increase security, facilitate change, a better understanding of the system, and facilitate testing.

Disadvantages of Debugging

While debugging is an important aspect of software engineering, there are also some disadvantages to consider:

1. **Time-consuming:** Debugging can be a time-consuming process, especially if the bug is difficult to find or reproduce. This can cause delays in the development process and add to the overall cost of the project.
2. **Requires specialized skills:** Debugging can be a complex task that requires specialized skills and knowledge. This can be a challenge for developers who are not familiar with the tools and techniques used in debugging.
3. **Can be difficult to reproduce:** Some bugs may be difficult to reproduce, which can make it challenging to identify and resolve them.
4. **Can be difficult to diagnose:** Some bugs may be caused by interactions between different components of a software system, which can make it challenging to identify the root cause of the problem.
5. **Can be difficult to fix:** Some bugs may be caused by fundamental design flaws or architecture issues, which can be difficult or impossible to fix without significant changes to the software system.
6. **Limited insight**: In some cases, debugging tools can only provide limited insight into the problem and may not provide enough information to identify the root cause of the problem.
7. **Can be expensive:** Debugging can be an expensive process, especially if it requires additional resources such as specialized debugging tools or additional development time.

Metrics for Process and products:

Software Measurement

- **Software Measurement:** A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process.

It is an authority within software engineering. The software measurement process is defined and governed by ISO Standard.

Software Measurement Principles

The software measurement process can be characterized by five activities-

1. **Formulation:** The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
2. **Collection:** The mechanism used to accumulate data required to derive the formulated metrics.
3. **Analysis:** The computation of metrics and the application of mathematical tools.
4. **Interpretation:** The evaluation of metrics results in insight into the quality of the representation.
5. **Feedback:** Recommendation derived from the interpretation of product metrics transmitted to the software team.

Need for Software Measurement

Software is measured to:

- Create the quality of the current product or process.
- Anticipate future qualities of the product or process.
- Enhance the quality of a product or process.
- Regulate the state of the project concerning budget and schedule.
- Enable data-driven decision-making in project planning and control.
- Identify bottlenecks and areas for improvement to drive process improvement activities.
- Ensure that industry standards and regulations are followed.
- Give software products and processes a quantitative basis for evaluation.
- Enable the ongoing improvement of software development practices.

Classification of Software Measurement

There are 2 types of software measurement:

1. **Direct Measurement:** In direct measurement, the product, process, or thing is measured directly using a standard scale.
2. **Indirect Measurement:** In indirect measurement, the quantity or quality to be measured is measured using related parameters i.e. by use of reference.

Software Metrics

A metric is a measurement of the level at which any impute belongs to a system product or process. Software metrics are a quantifiable or countable assessment of the attributes of a software product. There are 4 functions related to software metrics:

1. **Planning**
2. **Organizing**
3. **Controlling**
4. **Improving**

Characteristics of software Metrics

1. **Quantitative:** Metrics must possess a quantitative nature. It means metrics can be expressed in numerical values.
2. **Understandable:** Metric computation should be easily understood, and the method of computing metrics should be clearly defined.
3. **Applicability:** Metrics should be applicable in the initial phases of the development of the software.
4. **Repeatable:** When measured repeatedly, the metric values should be the same and consistent.
5. **Economical:** The computation of metrics should be economical.
6. **Language Independent:** Metrics should not depend on any programming language.

Types of Software Metrics:

1. product Metrics
2. Process Metrics
3. Project Metrics

1. **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated. Examples include lines of code, cyclomatic complexity, code coverage, defect density, and code maintainability index.
2. **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization. These metrics are used to optimize the development process and

maintenance activities of software. Examples include effort variance, schedule variance, defect injection rate, and lead time.

3. **Project Metrics:** The project metrics describes the characteristic and execution of a project. Examples include effort estimation accuracy, schedule deviation, cost variance, and productivity. Usually measures-
   - Number of software developer
   - Staffing patterns over the life cycle of software
   - Cost and schedule
   - Productivity

Advantages of Software Metrics

1. Reduction in cost or budget.
2. It helps to identify the particular area for improvising.
3. It helps to increase the product quality.
4. Managing the workloads and teams.
5. Reduction in overall time to produce the product,.
6. It helps to determine the complexity of the code and to test the code with resources.
7. It helps in providing effective planning, controlling and managing of the entire product.

Disadvantages of Software Metrics

1. It is expensive and difficult to implement the metrics in some cases.
2. Performance of the entire team or an individual from the team can't be determined. Only the performance of the product is determined.
3. Sometimes the quality of the product is not met with the expectation.
4. It leads to measure the unwanted data which is wastage of time.
5. Measuring the incorrect data leads to make wrong decision making.

Metrics for Software Quality:

- In Software Engineering, Software Measurement is done based on some Software Metrics where these software metrics are referred to as the measure of various characteristics of a Software.
  In Software engineering Software Quality Assurance (SAQ) assures the quality of the software. A set of activities in SAQ is continuously applied throughout the software process. Software Quality is measured based on some software quality metrics.
  There is a number of metrics available based on which software quality is measured. But among them, there are a few most useful metrics which are essential in software quality measurement. They are –

1. Code Quality
2. Reliability
3. Performance
4. Usability
5. Correctness
6. Maintainability
7. Integrity
8. Security

Now let's understand each quality metric **in detail –**

1. Code Quality – Code quality metrics measure the quality of code used for software project development. Maintaining the software code quality by writing Bug-free and semantically correct code is very important for good software project development. In code quality, both Quantitative metrics like the number of lines, complexity, functions, rate of bugs generation, etc, and Qualitative metrics like readability, code clarity, efficiency, and maintainability, etc are measured.

2. Reliability – Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).

3. Performance – Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.

4. Usability – Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.

5. Correctness – Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.

6. Maintainability – Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include the time required to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc.

7. Integrity – Software integrity is important in terms of how much it is easy to integrate with other required software which increases software functionality and what is the control on integration from unauthorized software's which increases the chances of cyber attacks.

8. Security – Security metrics measure how secure the software is. In the age of cyber terrorism, security is the most essential part of every software. Security assures that there are no unauthorized changes, no fear of cyber attacks, etc when the software product is in use by the end-user.