Software Testing Techniques

Software testing techniques are methods used to design and execute tests to evaluate software applications. The following are common testing techniques:

1. **Manual testing** – Involves manual inspection and testing of the software by a human tester.
2. **Automated testing** – Involves using software tools to automate the testing process.
3. **Functional testing** – Tests the functional requirements of the software to ensure they are met.
4. **Non-functional testing** – Tests non-functional requirements such as performance, security, and usability.
5. **Unit testing** – Tests individual units or components of the software to ensure they are functioning as intended.
6. **Integration testing** – Tests the integration of different components of the software to ensure they work together as a system.
7. **System testing** – Tests the complete software system to ensure it meets the specified requirements.
8. **Acceptance testing** – Tests the software to ensure it meets the customer's or end-user's expectations.
9. **Regression testing** – Tests the software after changes or modifications have been made to ensure the changes have not introduced new defects.
10. **Performance testing** – Tests the software to determine its performance characteristics such as speed, scalability, and stability.
11. **Security testing** – Tests the software to identify vulnerabilities and ensure it meets security requirements.
12. **Exploratory testing** – A type of testing where the tester actively explores the software to find defects, without following a specific test plan.
13. **Boundary value testing** – Tests the software at the boundaries of input values to identify any defects.
14. **Usability testing** – Tests the software to evaluate its user-friendliness and ease of use.
15. **User acceptance testing (UAT)** – Tests the software to determine if it meets the end-user's needs and expectations.

Software testing techniques are the ways employed to test the application under test against the functional or non-functional requirements gathered from business. Each testing technique helps to find a specific type of defect. For example, Techniques which may find structural defects might not be able to find the defects against the end-to-end business flow. Hence, multiple testing techniques are applied in a testing project to conclude it with acceptable quality.

Principles Of Testing

Below are the principles of software testing:

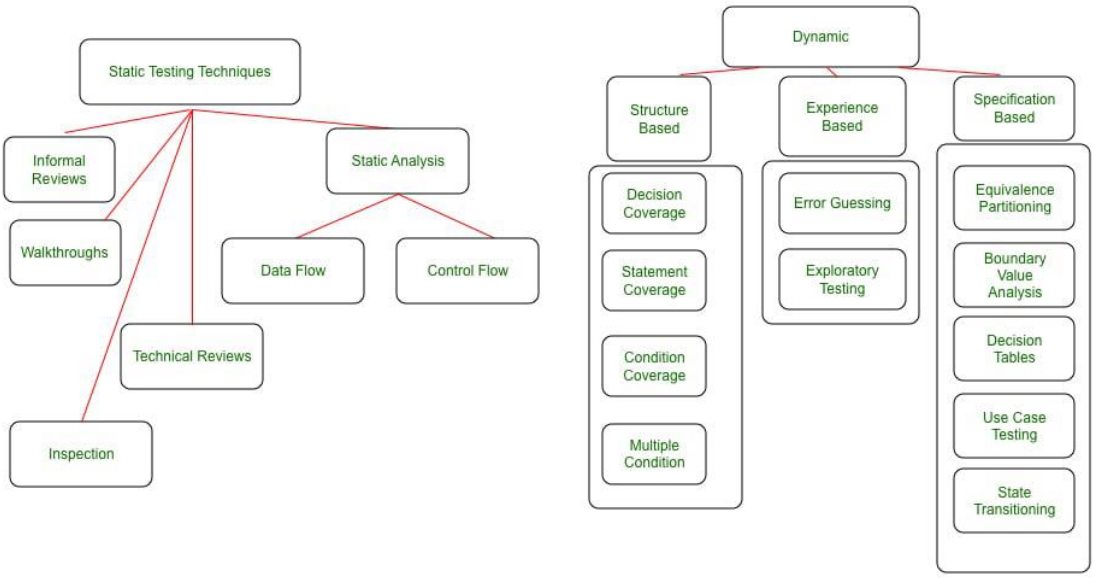
1. All the tests should meet the customer requirements.
2. To make our software testing should be performed by a third party
3. Exhaustive testing is not possible. As we need the optimal amount of testing based on the risk assessment of the application.
4. All the test to be conducted should be planned before implementing it
5. It follows the Pareto rule(80/20 rule) which states that 80% of errors come from 20% of program components.
6. Start testing with small parts and extend it to large parts.

Types Of Software Testing Techniques

There are two main categories of software testing techniques:

1. **Static Testing Techniques** are testing techniques which are used to find defects in Application under test without executing the code. Static Testing is done to avoid errors at an early stage of the development cycle and thus reducing the cost of fixing them.
2. **Dynamic Testing Techniques** are testing techniques that are used to test the dynamic behavior of the application under test, that is by the execution of the code base. The main purpose of dynamic testing is to test the application with dynamic inputs- some of which may be allowed as per requirement (Positive testing) and some are not allowed (Negative Testing).

Each testing technique has further types as showcased in the below diagram. Each one of them will be explained in detail with examples below.



Testing Techniques

Static Testing Techniques

As explained earlier, Static Testing techniques are testing techniques that do not require the execution of a code base. Static Testing Techniques are divided into two major categories:

1. **Reviews:** They can range from purely informal peer reviews between two developers/testers on the artifacts (code/test cases/test data) to totally formal **Inspections** which are led by moderators who can be internal/external to the organization.
    1. **Peer Reviews:** Informal reviews are generally conducted without any formal setup. It is between peers. For Example- Two developers/Testers review each other's artifacts like code/test cases.
    2. **Walkthroughs**: Walkthrough is a category where the author of work (code or test case or document under review) walks through what he/she has done and the logic behind it to the stakeholders to achieve a common understanding or for the intent of feedback.
    3. **Technical review:** It is a review meeting that focuses solely on the technical aspects of the document under review to achieve a consensus. It has less or no focus on the identification of defects based on reference documentation. Technical experts like architects/chief designers are required for doing the review. It can vary from Informal to fully formal.

4. **Inspection:** Inspection is the most formal category of reviews. Before the inspection, The document under review is thoroughly prepared before going for an inspection. Defects that are identified in the Inspection meeting are logged in the defect management tool and followed up until closure. The discussion on defects is avoided and a separate discussion phase is used for discussions, which makes Inspections a very effective form of reviews.

2. **Static Analysis:** Static Analysis is an examination of requirement/code or design with the aim of identifying defects that may or may not cause failures. For Example- Reviewing the code for the following standards. Not following a standard is a defect that may or may not cause a failure. There are many tools for Static Analysis that are mainly used by developers before or during Component or Integration Testing. Even **Compiler** is a Static Analysis tool as it points out incorrect usage of syntax, and it does not execute the code per se. There are several aspects to the code structure – Namely Data flow, Control flow, and Data Structure.
    1. **Data Flow:** It means how the data trail is followed in a given program – How data gets accessed and modified as per the instructions in the program. By Data flow analysis, You can identify defects like a variable definition that never got used.
    2. **Control flow:** It is the structure of how program instructions get executed i.e conditions, iterations, or loops. Control flow analysis helps to identify defects such as Dead code i.e a code that never gets used under any condition.
    3. **Data Structure:** It refers to the organization of data irrespective of code. The complexity of data structures adds to the complexity of code. Thus, it provides information on how to test the control flow and data flow in a given code.

Dynamic Testing Techniques

Dynamic techniques are subdivided into three categories:

1. Structure-based Testing:
These are also called White box techniques. Structure-based testing techniques are focused on how the code structure works and test accordingly. To understand Structure-based techniques, We first need to understand the concept of code coverage.
Code Coverage is normally done in Component and Integration Testing. It establishes what code is covered by structural testing techniques out of the total code written. One drawback of code coverage is that- it does not talk about code that has not been written at all (Missed requirement), There are tools in the market that can help measure code coverage.

There are multiple ways to test code coverage:

1. Statement coverage: Number of Statements of code exercised/Total number of statements. For Example, If a code segment has 10 lines and the test designed by you covers only 5 of them then we can say that statement coverage given by the test is 50%.
2. Decision coverage: Number of decision outcomes exercised/Total number of Decisions. For Example, If a code segment has 4 decisions (If conditions) and your test executes just 1, then decision coverage is 25%
3. Conditional/Multiple condition coverage: It has the aim to identify that each outcome of every logical condition in a program has been exercised.
2. Experience-Based Techniques:
These are techniques of executing testing activities with the help of experience gained over the years. Domain skill and background are major contributors to this type of testing. These techniques are used majorly for UAT/business user testing. These work on top of structured techniques like Specification-based and Structure-based, and it complements them. Here are the types of experience-based techniques:
1. Error guessing: It is used by a tester who has either very good experience in testing or with the application under test and hence they may know where a system might have a weakness. It cannot be an effective technique when used stand-alone but is really helpful when used along with structured techniques.
2. Exploratory testing: It is hands-on testing where the aim is to have maximum execution coverage with minimal planning. The test design and execution are carried out in parallel without documenting the test design steps. The key aspect of this type of testing is the tester's learning about the strengths and weaknesses of an application under test. Similar to error guessing, it is used along with other formal techniques to be useful.
3. Specification-based Techniques:
This includes both functional and nonfunctional techniques (i.e. quality characteristics). It basically means creating and executing tests based on functional or non-functional specifications from the business. Its focus is on identifying defects corresponding to given specifications. Here are the types of specification-based techniques:

1. Equivalence partitioning: It is generally used together and can be applied to any level of testing. The idea is to partition the input range of data into valid and non-valid sections such that one partition is considered "equivalent". Once we have the partitions identified, it only requires us to test with any value in a given partition assuming that all values in the partition will behave the same. For example, if the input field takes the value between 1-999, then values between 1-999 will yield similar results, and we need NOT test with each value to call the testing complete.
2. Boundary Value Analysis (BVA): This analysis tests the boundaries of the range- both valid and invalid. In the example above, 0,1,999, and 1000 are

boundaries that can be tested. The reasoning behind this kind of testing is that more often than not, boundaries are not handled gracefully in the code.

3. Decision Tables: These are a good way to test the combination of inputs. It is also called a Cause-Effect table. In layman's language, One can structure the conditions applicable for the application segment under test as a table and identify the outcomes against each one of them to reach an effective test.

1. It should be taken into consideration that there are not too many combinations, so that table becomes too big to be effective.
2. Take an example of a Credit Card that is issued if both credit score and salary limit are met. This can be illustrated in below decision table:

	Rule 1	Rule 2	Rule 3	Rule 4
Credit Score Met	FALSE	FALSE	TRUE	TRUE
Salary Limit Met	FALSE	TRUE	FALSE	TRUE
Issue Credit Card	No	No	No	Yes

Decision Table

4. Use case-based Testing: This technique helps us to identify test cases that execute the system as a whole- like an actual user (Actor), transaction by transaction. Use cases are a sequence of steps that describe the interaction between the Actor and the system. They are always defined in the language of the Actor, not the system. This testing is most effective in identifying the integration defects. Use case also defines any preconditions and postconditions of the process flow. ATM machine example can be tested via use case:

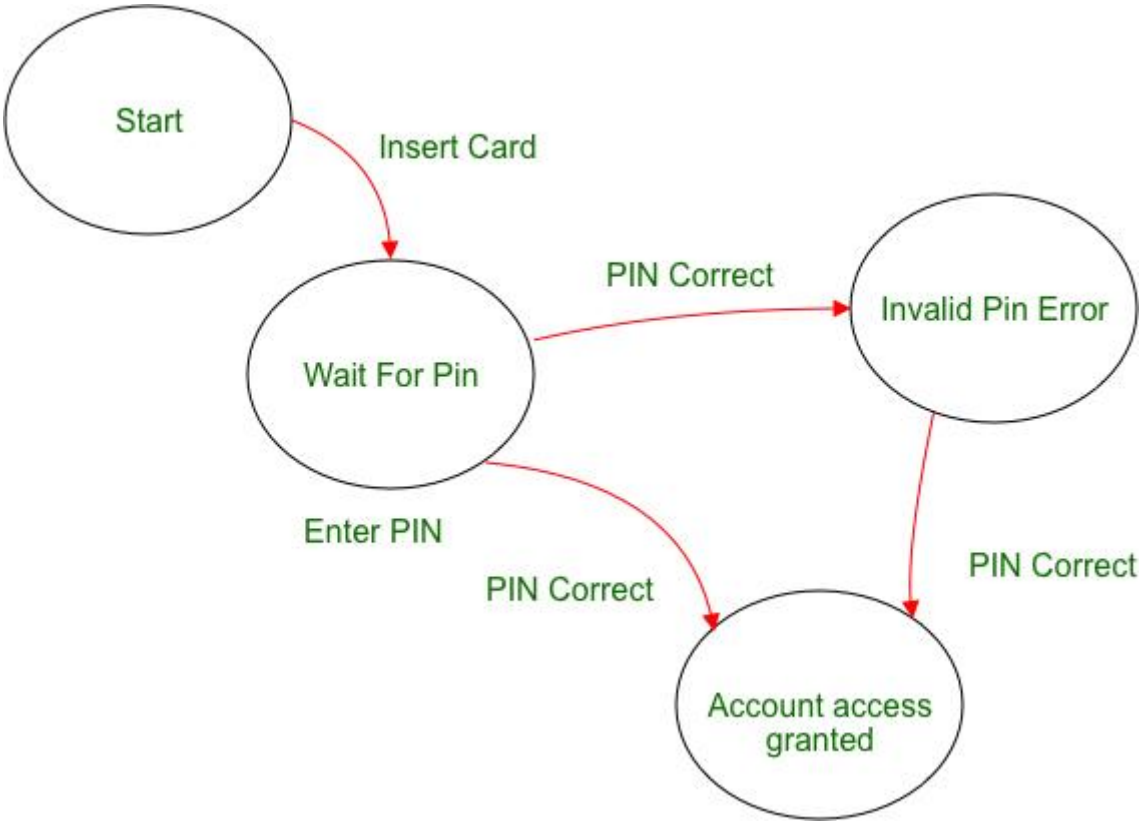
Happy	1	A: Insert Card
Flow	2	A: Enter PIN
A: Actor	3	S: Validte PIN
S: System	4	S: Allow to withdraw

Use case-based Testing

5. State Transition Testing: It is used where an application under test or a part of it can be treated as FSM or finite state machine. Continuing the simplified ATM example above, We can say that ATM flow has finite states and hence can be tested with the State transition technique. There are 4 basic things to consider –

1. States a system can achieve
2. Events that cause the change of state
3. The transition from one state to other
4. Outcomes of change of state

A state event pair table can be created to derive test conditions – both positive and negative.



State Transition

Advantages of software testing techniques:

1. Improves software quality and reliability – By using different testing techniques, software developers can 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 – Techniques like usability testing can help 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 techniques:

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.
6. Automated testing limitations – Automated testing tools may have limitations, such as difficulty in testing certain aspects of the software, and may require significant maintenance and updates.

Difference between Black Box Vs White Vs Grey Box Testing

- Last Updated : 15 Sep, 2022

- Read

- Discuss

1. Black Box Testing :
Black box testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products. It is also called Functional testing. Black-box testing focuses on software's external attributes and behavior. This type of testing looks at an application's software expected behavior from the user's point of view.

2. White Box Testing :
White-box testing or glass-box testing is a software testing technique that tests the software by using the knowledge of internal data structures, physical logic flow, and architecture at the level of source code. This testing works by looking at testing from the developer's point of view. This testing is also known as glass box testing, clear box testing, structural testing, or non-functional testing.

3. Gray Box Testing :
Gray Box Testing is a combination of the Black Box Testing technique and the White Box Testing technique in software testing. The gray-box testing involves inputs and outputs of a program for the testing purpose but test design is tested by using the information about the code. Gray-box testing is well suited for web application testing because it factors in a high-level design environment and the inter-operability conditions.
Let's see the tabular differences between them.

S. No.	Black Box Testing	Gray Box Testing	White Box Testing

1.	This testing has Low granularity.	This testing has a medium level of granularity.	This testing has high-level granularity.
2.	It is done by end-users and also done by the tester, developers.	It is done by end-users (called user acceptance testing), also done by testers and developers.	It is generally done by testers and developers.
3.	Here, Internals are not required to be known.	Here, Internals relevant to the testing is known.	Here, the Internal code of the application and database is known.
4.	It is likely to be less exhaustive than the other two.	It is kind of in-between.	Most exhaustive among all three.
5.	It is based on requirements, and test cases on the functional specifications, as the internals are not known.	It provides better variety/depth in test cases on account of high-level knowledge of the internals.	It has the ability to exercise code with a relevant variety of data.
6.	If used algorithm testing, is not suited best for that.	If used algorithm testing is also not suited best for that.	If used algorithm testing, it is suited best for that.
7.	It is suited for functional or business testing.	It is suited for functional or business domain testing deeply.	It is used for all.
8.	This testing involves validating the outputs for given inputs, the application being tested as a black-box technique.	Herein, we have a better variety of inputs and the ability to extract test results from the database for comparison with expected results.	It involves structural testing and enables logic coverage, decisions, etc. within the code.
9.	This is also called Opaque-box testing, Closed-box testing, input-output testing, Data-	This is also called translucent box testing	This is also called Glass-box testing, Clear-box testing, Design-based testing,

	driven testing, Behavioral, Functional testing		Logic-based testing, Structural testing, Code-based testing.
10.	**Black-box test design techniques-** • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	**Gray box test design techniques-** • Matrix testing • Regression testing • Pattern testing • Orthogonal Array Testing	**White-box test design techniques-** • Control flow testing • Data flow testing • Branch testing
11.	Black Box testing provides resilience and security against viral attacks.	Gray Box testing does not provide resilience and security against viral attacks.	White Box testing does not provide resilience and security against viral attacks.

Index	Black Box Testing	White Box Testing	Grey Box Testing
1	Knowledge of internal working structure (Code) is not required for this type of testing. Only GUI (Graphical User Interface) is required for test cases.	Knowledge of internal working structure (Coding of software) is necessarily required for this type of testing.	Partially Knowledge of the internal working structure is required.
2	Black Box Testing is also known as functional testing, data-driven testing, and closed box testing.	White Box Testing is also known as structural testing, clear box testing, code-based testing, and transparent testing.	Grey Box Testing is also known as translucent testing as the tester has limited knowledge of coding.
3	The approach towards testing includes trial techniques and error guessing method because tester does not need knowledge of internal coding of the software.	White Box Testing is proceeded by verifying the system boundaries and data domains inherent in the software as there is no lack of internal coding knowledge.	If the tester has knowledge of coding, then it is proceeded by validating data domains and internal system boundaries of the software.

4	The testing space of tables for inputs (inputs to be used for creating test cases) is pretty huge and largest among all testing spaces.	The testing space of tables for inputs (inputs to be used for creating test cases) is less as compared to Black Box testing.	The testing space of tables for inputs (inputs to be used for creating test cases) is smaller than Black Box and White Box testing.
5	It is very difficult to discover hidden errors of the software because errors can be due to internal working which is unknown for Black Box testing.	It is simple to discover hidden errors because it can be due to internal working which is deeply explored in White Box testing.	Difficult to discover the hidden error. Might be found in user level testing.
6	It is not considered for algorithm testing.	It is well suitable and recommended for algorithm testing.	It is not considered for algorithm testing.
7	Time consumption in Black Box testing depends upon the availability of the functional specifications.	White Box testing takes a long time to design test cases due to lengthy code.	Test cases designing can be done in a short time period.
8	Tester, developer and the end user can be the part of testing.	Only tester and developer can be a part of testing; the end user can not involve.	Tester, developer and the end user can be the part of testing.
9	It is the least time-consuming process among all the testing processes.	The entire testing process is the most time consuming among all the testing processes.	less time consuming than White Box testing.
10	Resilience and security against viral attacks are covered under Black Box testing.	Resilience and security against viral attacks are not covered under White Box testing.	Resilience and security against viral attacks are not covered under Grey Box testing.
11	The base of this testing is external expectations internal behavior is unknown.	The base of this testing is coding which is responsible for internal working.	Testing based on high-level database diagrams and dataflow diagrams.

12	It is less exhaustive than White Box and Grey Box testing methods.	It is most exhaustive between Black Box and Grey Box testing methods.	Partly exhaustive; depends upon the type of test cases are coding based or GUI based.

Introduction of Software Testing Life Cycle (STLC):
The Software Testing Life Cycle (STLC) is a systematic approach to testing a software application to ensure that it meets the requirements and is free of defects. It is a process that follows a series of steps or phases, and each phase has specific objectives and deliverables. The STLC is used to ensure that the software is of high quality, reliable, and meets the needs of the end-users.

The main goal of the STLC is to identify and document any defects or issues in the software application as early as possible in the development process. This allows for issues to be addressed and resolved before the software is released to the public.

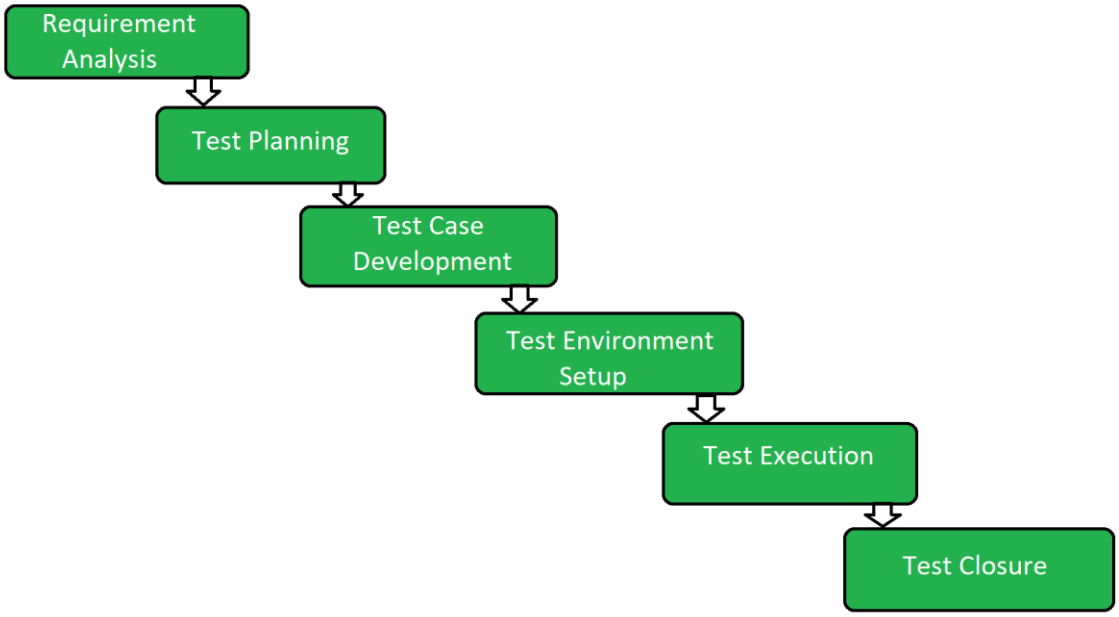
The stages of the STLC include: Test Planning, Test Analysis, Test Design, Test Environment Setup, Test Execution, Test Closure, and Defect Retesting. Each of these stages includes specific activities and deliverables that help to ensure that the software is thoroughly tested and meets the requirements of the end-users.

Overall, the STLC is an important process that helps to ensure the quality of software applications and provides a systematic approach to testing. It allows organizations to release high-quality software that meets the needs of their customers, ultimately leading to customer satisfaction and business success.

Characteristics of STLC:

- STLC is a fundamental part of Software Development Life Cycle (SDLC) but STLC consists of only the testing phases.
- STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.
- STLC yields a step-by-step process to ensure quality software.

In the initial stages of STLC, while the software product or the application is being developed, the testing team analyzes and defines the scope of testing, entry and exit criteria and also the test cases. It helps to reduce the test cycle time and also enhance the product quality. As soon as the development phase is over, testing team is ready with test cases and start the execution. This helps in finding bugs in the early phase. **Phases of STLC:**



1. Requirement Analysis:

Requirement Analysis is the first step of Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement.

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
- dentifying any potential risks or issues that may impact the testing process

Creating a requirement traceability matrix (RTM) to map requirements to test cases

At the end of this stage, the testing team should have a clear understanding of the software requirements and should have identified any potential issues that may impact the testing process. This will help to ensure that the testing process is focused on the most important areas of the software and that the testing team is able to deliver high-quality results.

2. Test Planning:

Test Planning is most efficient phase of software testing life cycle where all testing plans are defined. In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.

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

At the end of this stage, the testing team should have a detailed plan for the testing activities that will be performed, and a clear understanding of the testing objectives, scope, and deliverables. This will help to ensure that the testing process is well-organized and that the testing team is able to deliver high-quality results.

3. Test Case Development:

The test case development phase gets started once the test planning phase is completed. In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing. When the test cases are prepared then they are reviewed by quality assurance team.

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

At the end of this stage, the testing team should have a set of comprehensive and accurate test cases that provide adequate coverage of the software or application. This will help to ensure that the testing process is thorough and that any potential issues are identified and addressed before the software is released.

4. Test Environment Setup:

Test environment setup is the vital part of the STLC. Basically test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process the testing team is not involved. either the developer or the customer creates the testing environment.

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 a 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.

It is important to note that test execution is an iterative process and may need to be repeated multiple times until all identified defects are fixed and the software is deemed fit for release.

5. Test Execution:

After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.

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 a 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.

It is important to note that test execution is an iterative process and may need to be repeated multiple times until all identified defects are fixed and the software is deemed fit for release.

6. Test Closure:

Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. The main objective of the test closure stage is to ensure that all testing-related activities have been completed, and that the software is ready for release.

At the end of the test closure stage, the testing team should have a clear understanding of the software's quality and reliability, and any defects or issues that were identified during testing should have been resolved. The test closure stage also includes documenting the testing process and any lessons learned, so that they can be used to improve future testing processes

Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. 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

It is important to note that test closure is not just about documenting the testing process, but also about ensuring that all relevant information is shared and any lessons learned are captured for future reference. The goal of test closure is to ensure that the software is ready for release and that the testing process has been conducted in an organized and efficient manner.