Software Testing

**Testing:**

Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include preventing bugs, reducing development costs and improving performance.

**Types of Testing:**

* Manual testing
* Automation Testing

**Levels of Testing:**

Components Testing

Integration testing

System testing

Acceptance testing

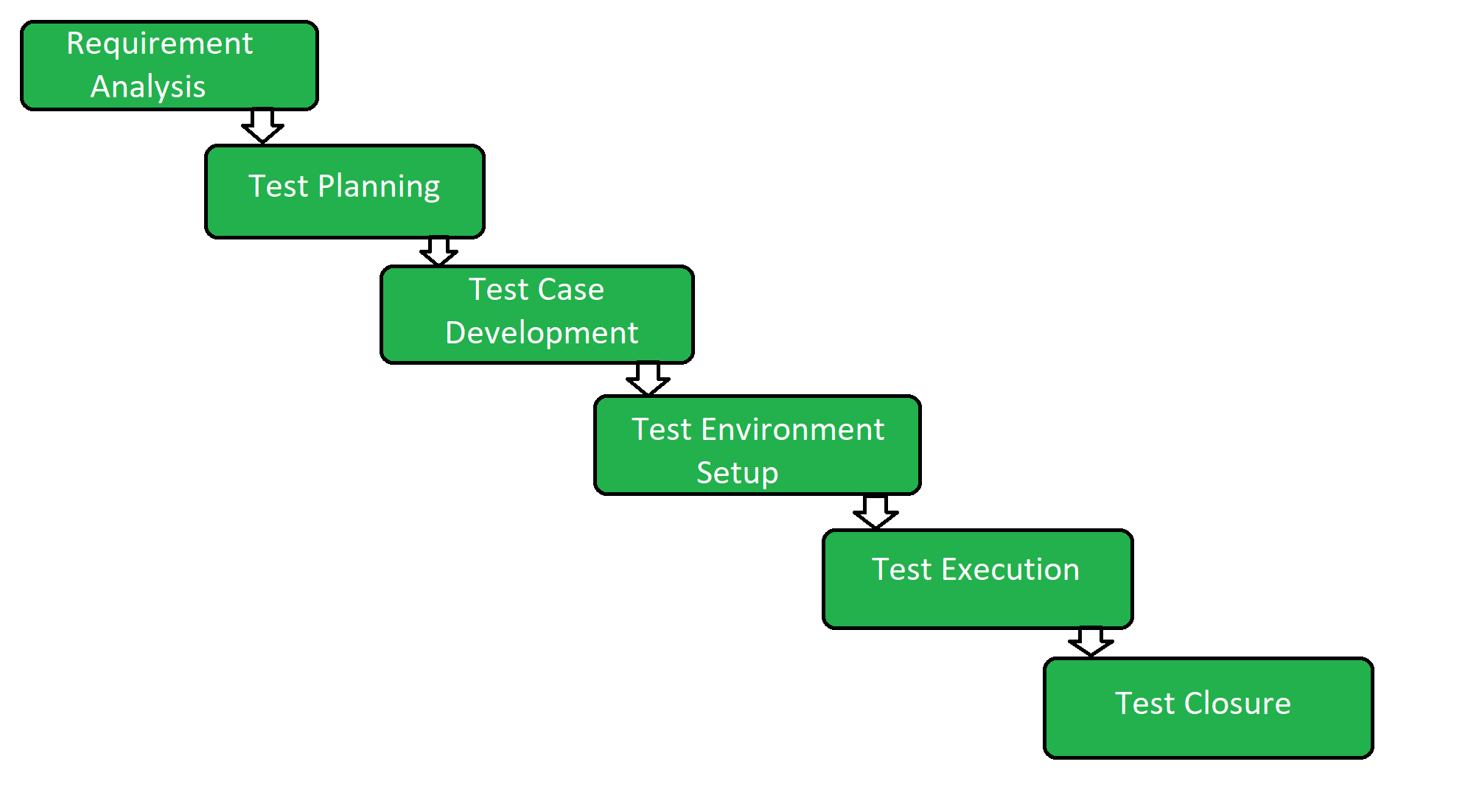
**Methods of Testing:**

1. Black box testing
2. White box testing
3. Gray box testing

**STLC**

Software Testing Life Cycle (STLC) is a process used to test software and ensure that quality standards are met. Tests are carried out systematically over several phases. During product development, phases of the STLC may be performed multiple times until a product is deemed suitable for release.

Diagram:



**Phases of STLC**

**1. Requirement Analysis:**Requirement Analysis is the first step of the 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 the quality assurance team meets with the stakeholders to better understand the detailed knowledge of requirements.

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

**3. Test Case Development:**The test case development phase gets started once the test planning phase is completed. In this phase testing team notes down the detailed test cases. The testing team also prepares the required test data for the testing. When the test cases are prepared then they are reviewed by the quality assurance team.

**4. Test Environment Setup:** Test environment setup is a vital part of the STLC. Basically, the 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.

**5. Test Execution:**After the test case development and test environment setup test execution phase gets started. In this phase testing team starts executing test cases based on prepared test cases in the earlier step.

**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.

Test Plan Document:

The Software Testing Life Cycle (STLC) test plan document is a comprehensive outline of the testing approach and strategies to be followed for a particular software project. It provides a roadmap for the testing activities to ensure the quality and reliability of the software under development.

1. Introduction:

- Purpose and objectives of the test plan.

- Scope and overview of the software project.

- Testing team members and their roles.

2. Test Strategy:

- Testing approach and methodologies to be employed.

- Test levels (unit, integration, system, etc.) and types (functional, performance, security, etc.) to be conducted.

- Test environments and tools to be used.

- Entry and exit criteria for each test level.

3. Test Schedule:

- Estimated timeline for each testing phase.

- Dependencies and milestones.

4. Test Deliverables:

- List of documents or artifacts to be produced during testing (test cases, test scripts, test data, etc.).

- Traceability matrix to map requirements to test cases.

5. Test Environment:

- Hardware and software requirements.

- Configuration management and version control.

6. Test Execution:

- Test case execution approach.

- Test data preparation and management.

- Defect reporting and tracking process.

- Test coverage metrics and criteria for test completion.

7. Risks and Mitigation:

- Identification and assessment of project risks.

- Contingency plans and mitigation strategies.

8. Test Sign-off:

- Criteria for test completion and acceptance.

- Approvals and sign-off process.

9. Test Team:

- Roles and responsibilities of testing team members.

- Communication and collaboration channels.

10. Appendix:

- Glossary of terms.

- References to additional supporting documents.

Remember, the test plan document should be tailored to the specific project and organization's needs. It serves as a reference for the testing team and stakeholders, ensuring a structured and organized approach to testing.

# Introduction

Brief introduction of the test strategies, process, workflow and methodologies used for the project

## Scope

### **In Scope**

Scope defines the features, functional or non-functional requirements of the software that **will be** tested

### **Out of Scope**

Out Of Scope defines the features, functional or non-functional requirements of the software that **will NOT be** tested

## Quality Objective

Here make a mention of the overall objevtie that you plan to achive withou your testing

Some objectives of your testing project could be

* Ensure the Application Under Test conforms to functional and non-functional requirements
* Ensure the AUT meets the quality specifications defined by the client
* Bugs/issues are identified and fixed before go live

## Roles and Responsibilities

Detail description of the Roles and responsibilities of different team members like

* QA Analyst
* Test Manager
* Configuration Manager
* Developers
* Installation Team

Amongst others

### 

# Test Methodology

## Overview

Mention the reason of adopting a particular test methodology for the project. The test methodology selected for the project could be

* WaterFall
* Iterative
* Agile
* Extreme Programming

The methodology selected depends on multiple factors. You can read about Test Methodology [here](http://www.guru99.com/testing-methodology.html)

## Test Levels

**Test Levels define the Types of Testing to be executed on the Application Under Test (AUT**). The Testing Levels primarily depends on the scope of the project, time and budget constraints.

## Bug Triage

The goal of the triage is to

* To define the type of resolution for each bug
* To prioritize bugs and determine a schedule for all “To Be Fixed Bugs’.

## Suspension Criteria and Resumption Requirements

Suspension criteria define the criteria to be used to suspend all or part of the testing procedure while Resumption criteria determine when testing can resume after it has been suspended

## Test Completeness

Here you define the criterias that will deem your testing complete.

For instance, a few criteria to check Test Completeness would be

* 100% test coverage
* All Manual & Automated Test cases executed
* All open bugs are fixed or will be fixed in next release

# Test Deliverables

Here mention all the Test Artifacts that will be delivered during different phases of the testing lifecycle.

Here are the sample deliverables

|  |
| --- |
| * Test Plan * Test Cases * Requirement Traceability Matrix * Bug Reports * Test Strategy * Test Metrics * Customer Sign Off |

# Resource & Environment Needs

## Testing Tools

Make a list of Tools like

* Requirements Tracking Tool
* Bug Tracking Tool
* Automation Tools

Required to test the project

## Test Environment

It mentions the minimum **hardware** requirements that will be used to test the Application.

Following **software’s** are required in addition to client-specific software.

Windows 8 and above

Office 2013 and above

MS Exchange, etc.

# Terms/Acronyms

Make a mention of any terms or acronyms used in the project

|  |  |
| --- | --- |
| TERM/ACRONYM | DEFINITION |
| API | Application Program Interface |
| AUT | Application Under Test |

Bug Life Cycle

**What is bug?**

Bug in software testing is flaw or default in a component or system or software that can cause the components or system to fail to perform its required functions, in other words, we can say that if the bug or defect encountered during the execution of the test, it may cause the failure of the components i.e. does not works as it expected from the components. For example, incorrect data definition, statements, input data, design, etc.

Bug life cycle or defect life cycle.

The number of states that a defect goes through varies from project to project. Below lifecycle diagram, covers all possible states

* **New:** When a new defect is logged and posted for the first time. It is assigned a status as NEW.
* **Assigned:** Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team
* **Open**: The developer starts analyzing and works on the defect fix
* **Fixed**: When a developer makes a necessary code change and verifies the change, he or she can make bug status as “Fixed.”
* **Pending retest**: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is “pending retest.”
* **Retest**: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to “Re-test.”
* **Verified**: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is “verified.”
* **Reopen**: If the bug persists even after the developer has fixed the bug, the tester changes the status to “reopened”. Once again the bug goes through the life cycle.
* **Closed**: If the bug is no longer exists then tester assigns the status “Closed.”
* **Duplicate**: If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to “duplicate.”
* **Rejected**: If the developer feels the defect is not a genuine defect then it changes the defect to “rejected.”
* **Deferred**: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status “Deferred” is assigned to such bugs
* **Not a bug**: If it does not affect the functionality of the application then the status assigned to a bug is “Not a bug”.



1. Tester finds the defect
2. Status assigned to defect- New
3. A defect is forwarded to Project Manager for analyze
4. Project Manager decides whether a defect is valid
5. Here the defect is not valid- a status is given “Rejected.”
6. So, project manager assigns a status **rejected**. If the defect is not rejected then the next step is to check whether it is in scope. Suppose we have another function- email functionality for the same application, and you find a problem with that. But it is not a part of the current release when such defects are assigned as a **postponed or deferred**status.
7. Next, the manager verifies whether a similar defect was raised earlier. If yes defect is assigned a status **duplicate**.
8. If no the defect is assigned to the developer who starts fixing the code. During this stage, the defect is assigned a status **in- progress.**
9. Once the code is fixed. A defect is assigned a status **fixed**

Next, the tester will re-test the code. In case, the test cases passes the defect is **closed.** If the test cases fail again, the defect is **re-opened** and assigned to the developer.

1. Consider a situation where during the 1st release of Flight Reservation a defect was found in Fax order that was fixed and assigned a status closed. During the second upgrade release the same defect again re-surfaced. In such cases, a closed defect will be **re-opened.**

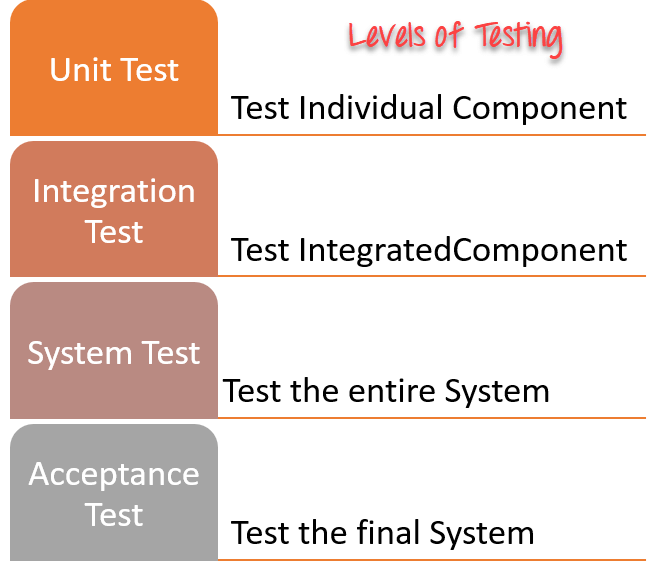
# **Different Levels of Testing in Software Testing**

## Overview

There are 4 levels of testing - unit testing, integration testing, system testing and acceptance testing. These levels are based on the extent of module testing. Unit testing is done by the developer, whereas integration testing and system testing are done by the testing team. Acceptance testing is done by the client to ensure the features are as per the requirements he listed.

## What are the Levels of Software Testing?

Software testing is a continuous process of finding bugs in the application and hence improving the quality of the product. This involves testing the application under various stages of development. In agile methodology, development and testing are done simultaneously. Hence, there are different levels of testing based on the module under the test. Through levels of testing, every feature of the product gets multiple rounds of testing.



## Different Levels of Testing

### **Unit Testing**

* Unit testing is when every module of the application gets tested respectively.
* Unit testing is done by the developer himself. After he has written code for a feature, he will ensure it is working fine.
* Unit tests are the smallest testable component of the application.
* Nowadays we have Junit, Pytest, and TestNg frameworks for unit testing the application.

### **Integration Testing**

* Integration testing is a testing technique where two or more independent components are tested together.
* Integration testing is done by the developer. Here test cases are written to ensure the data flowing between them is correct.
* For example, testing the signup form where UI validations are correct, data reaching API, and getting stored are all validated.
* Integration testing is done when the application is still developing to find bugs early on in the development process.

### **System Testing**

* System testing is done by the tester where the entire application is tested as a single unit.
* Hence, system testing test cases are also performance test cases, load testing, and stress testing test cases.
* System testing is done to find the errors which might have been overlooked during unit or integration testing.
* System testing evaluates both functional and non-functional test cases.

### **Acceptance Testing**

* Acceptance testing is done by the client where he evaluates whether the product is made by the requirement he listed out.
* Acceptance testing is done at the UAT server where a well-tested product is deployed by the team for the client's reference so he can track ongoing changes in the project
* There is a defined acceptance criterion that is laid at the time of requirement listing so that the client can validate that the product is meeting the acceptance criteria.
* Once the client completes acceptance testing the product goes to production where users can use the final application.

## Conclusion

1. There are four levels of software testing that an application undergoes. These levels are by the module which is being tested.
2. Unit testing is done by the developer. He ensures the modules he coded are defect-free.
3. Integration and system testing is done by the tester. In integration testing, different modules are tested together whereas in system testing the entire application is verified.
4. Acceptance testing is done by the client. Post the 4 levels of testing product goes live for users to use the application.

# Functional Vs Non-Functional Testing – Difference Between Them

## What is Functional Testing?

[Functional testing](https://www.guru99.com/functional-testing.html) is a type of testing which verifies that each **function** of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing, and it is not concerned about the source code of the application.

Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results. This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application Under Test. The testing can be done either manually or using automation

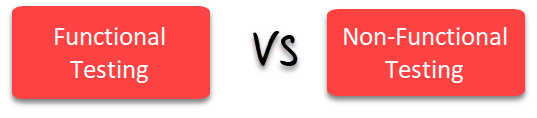
## What is Non-Functional Testing?

Non-functional testing is a type of testing to check non-functional aspects (performance, usability, reliability, etc.) of a software application. It is explicitly designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

A good example of non-functional test would be to check how many people can simultaneously login into a software.

Non-functional testing is equally important as functional testing and affects client satisfaction.

## Difference between Functional Testing and Non Functional Testing



| **Parameters** | **Functional** | **Non-functional testing** |
| --- | --- | --- |
| **Execution** | It is performed before non-functional testing. | It is performed after the functional testing. |
| **Focus area** | It is based on customer’s requirements. | It focusses on customer’s expectation. |
| **Requirement** | It is easy to define functional requirements. | It is difficult to define the requirements  for non-functional testing. |
| **Usage** | Helps to validate the behavior of the application. | Helps to validate the performance of the  application. |
| **Objective** | Carried out to validate software actions. | It is done to validate the performance  of the software. |
| **Requirements** | Functional testing is carried out using the functional specification. | This kind of testing is carried out by performance  specifications |
| **Manual testing** | Functional testing is easy to execute by manual testing. | It’s very hard to perform non-functional  testing manually. |
| **Functionality** | It describes what the product does. | It describes how the product works. |
| **Example Test Case** | Check login functionality. | The dashboard should load in 2 seconds. |
| **Testing Types** | Examples of Functional Testing Types   * Unit testing * Smoke testing * User Acceptance * Integration Testing * Regression testing * Localization * Globalization * Interoperability | Examples of Non-functional Testing Types   * Performance Testing * Volume Testing * Scalability * Usability Testing * Load Testing * Stress Testing * Compliance Testing * Portability Testing * Disaster Recover Testing |

#### **Key Difference Between Functional Testing and Non Functional Testing**

* Functional testing verifies each function/feature of the software whereas Non Functional testing verifies non-functional aspects like performance, usability, reliability, etc.
* Functional testing can be done manually whereas Non Functional testing is hard to perform manually.
* Functional testing is based on customer’s requirements whereas Non Functional testing is based on customer’s expectations.
* Functional testing has a goal to validate software actions whereas Non Functional testing has a goal to validate the performance of the software.
* A Functional Testing example is to check the login functionality whereas a Non Functional testing example is to check the dashboard should load in 2 seconds.
* Functional describes what the product does whereas Non Functional describes how the product works.
* Functional testing is performed before the non-functional testing.

**Let us explore some of the most common testing types:**

1. [Accessibility testing](https://www.perfecto.io/resources/types-of-testing#accessibility)
2. [Acceptance testing](https://www.perfecto.io/resources/types-of-testing#acceptance)
3. [Black box testing](https://www.perfecto.io/resources/types-of-testing#blackbox)
4. [End to end testing](https://www.perfecto.io/resources/types-of-testing#e2e)
5. [Functional testing](https://www.perfecto.io/resources/types-of-testing#functional)
6. [Interactive testing](https://www.perfecto.io/resources/types-of-testing#interactive)
7. [Integration testing](https://www.perfecto.io/resources/types-of-testing#integration)
8. [Load testing](https://www.perfecto.io/resources/types-of-testing#load)
9. [Non-functional testing](https://www.perfecto.io/resources/types-of-testing#non)
10. [Performance testing](https://www.perfecto.io/resources/types-of-testing#performance)
11. [Regression testing](https://www.perfecto.io/resources/types-of-testing#regression)
12. [Sanity testing](https://www.perfecto.io/resources/types-of-testing#sanity)
13. [Security testing](https://www.perfecto.io/resources/types-of-testing#security)
14. [Single user performance testing](https://www.perfecto.io/resources/types-of-testing#single)
15. [Smoke testing](https://www.perfecto.io/resources/types-of-testing#smoke)
16. [Stress testing](https://www.perfecto.io/resources/types-of-testing#stress)
17. [Unit testing](https://www.perfecto.io/resources/types-of-testing#unit)
18. [White-box testing](https://www.perfecto.io/resources/types-of-testing#whitebox)

This is just a sample of different methods of testing, but there are many others. Many of these types of testing can be done manually — or they can be automated.

### **1. Accessibility Testing**

[Accessibility testing](https://www.perfecto.io/accessibility-testing) is the practice of ensuring your mobile and web apps are working and usable for users without and with disabilities such as vision impairment, hearing disabilities, and other physical or cognitive conditions.

### **2. Acceptance Testing**

Acceptance testing ensures that the end-user (customers) can achieve the goals set in the business requirements, which determines whether the software is acceptable for delivery or not. It is also known as user acceptance testing (UAT).

### **3. Black Box Testing**

Black box testing involves testing against a system where the code and paths are invisible.

### **4. End to End Testing**

End to end testing is a technique that tests the application’s workflow from beginning to end to make sure everything functions as expected.

### **5. Functional Testing**

[Functional testing](https://www.perfecto.io/functional-testing-web-mobile-apps) checks an application, website, or system to ensure it’s doing exactly what it’s supposed to be doing.

### **6. Interactive Testing**

Also known as manual testing, [interactive testing](https://www.perfecto.io/solutions/interactive-testing) enables testers to create and facilitate manual tests for those who do not use automation and collect results from external tests.

### **7. Integration Testing**

Integration testing ensures that an entire, integrated system meets a set of requirements. It is performed in an integrated hardware and software environment to ensure that the entire system functions properly.

### **8. Load Testing**

This type of non-functional software testing process determines how the software application behaves while being accessed by multiple users simultaneously.

### **9. Non Functional Testing**

[Non functional testing](https://www.perfecto.io/blog/what-is-non-functional-testing) verifies the readiness of a system according to nonfunctional parameters (performance, accessibility, UX, etc.)  which are never addressed by functional testing.

### **10. Performance Testing**

[Performance testing](https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing) examines the speed, stability, reliability, scalability, and resource usage of a software application under a specified workload.

### **11. Regression Testing**

[Software regression testing](https://www.perfecto.io/resources/software-regression-testing) is performed to determine if code modifications break an application or consume resources.

### **12. Sanity Testing**

Performed after bug fixes, sanity testing determines that the bugs are fixed and that no further issues are introduced to these changes.

### **13. Security Testing**

Security testing unveils the vulnerabilities of the system to ensure that the software system and application are free from any threats or risks. These tests aim to find any potential flaws and weaknesses in the software system that could lead to a loss of data, revenue, or reputation per employees or outsides of a company.

### **14. Single User Performance Testing**

[Single user performance testing](https://www.perfecto.io/blog/single-user-performance-testing) checks that the application under test performs fine according to specified threshold without any system load. This benchmark can be then used to define a realistic threshold when the system is under load.

### **15. Smoke Testing**

This type of software testing validates the stability of a software application, it is performed on the initial software build to ensure that the critical functions of the program are working.

### **16. Stress Testing**

Stress testing is a software testing activity that tests beyond normal operational capacity to test the results.

### **17. Unit Testing**

Unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own, speeding up testing strategies and reducing wasted tests.

### **18. White Box Testing**

White box testing involves testing the product's underlying structure, architecture, and code to validate input-output flow and enhance design, usability, and security.

**19. Monkey Testing**

The user tests the application or system by providing random inputs and checking the behavior, or seeing whether the application or system will crash. Monkey testing is usually implemented as random, automated unit tests.

scrum

Scrum is a management framework that teams use to self-organize and work towards a common goal.

What is agile

Agile methodology is a project management approach that prioritizes cross-functional collaboration and continuous improvement. It divides projects into smaller phases and guides teams through cycles of planning, execution, and evaluation.

**Software Testing** is a method to check whether the actual software product matches expected requirements and to ensure that software product is[Defect](https://www.guru99.com/defect-management-process.html)free.

The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. in detail, the SDLC methodology focuses on the following phases of software development:

* Requirement analysis
* Planning
* Software design such as architectural design
* Software development
* Testing
* Deployment

**Software Testing Life Cycle (STLC)** is a sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities.

STLC Model Phases

1. Requirement Analysis
2. Test Planning
3. Test case development
4. Test Environment setup
5. Test Execution
6. Test Cycle closure

**Test basis**

Test basis is defined as the source of information or the document that is needed to write test cases and also for test analysis.

**Test case**

A test case refers to the actions required to verify a specific feature or functionality in software testing. The test case details the steps, data, prerequisites, and post conditions necessary to verify a feature.

**Test Suite**

Test suites are the logical grouping or collection of test cases to run a single job with different test scenarios. For instance, a test suite for product purchase has multiple test cases, like: Test Case 1: Login. Test Case 2: Adding Products.

**Test Scenario**

A test scenario is a set of manual or automated test cases that helps determine the positive and negative project characteristics. It provides an overview of what needs to be tested by a QA company. The main purpose of a test scenario is to check the entire system performance from the end-user point of view.

**BLACK BOX TESTING**

Black box testing, a form of testing that is performed with no knowledge of a system’s internals, can be carried out to evaluate the functionality, security, performance, and other aspects of an application. [Dynamic code analysis](https://www.checkpoint.com/cyber-hub/cloud-security/what-is-dynamic-code-analysis/) is an example of automated black box security testing. Black box evaluators define test cases and interact with the software like a user would to validate that it does what it should, how it should.

Black box testing

## Generic steps of black box testing

* The black box test is based on the specification of requirements, so it is examined in the beginning.
* In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
* In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
* The fourth phase includes the execution of all test cases.
* In the fifth step, the tester compares the expected output against the actual output.
* In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

## Test procedure

The test procedure of black box testing is a kind of process in which the tester has specific knowledge about the software's work, and it develops test cases to check the accuracy of the software's functionality.

It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function.A tester knows about the definite output of a particular input, but not about how the result is arising. There are various techniques used in black box testing for testing like decision table technique, boundary value analysis technique, state transition, All-pair testing, cause-effect graph technique, equivalence partitioning technique, error guessing technique, use case technique and user story technique. All these techniques have been explained in detail within the tutorial.

## Test cases

Test cases are created considering the specification of the requirements. These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications. For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalid input values to determine the correct output. Test cases are mainly designed for functional testing but can also be used for non-functional testing. Test cases are designed by the testing team, there is not any involvement of the development team of software.

## Types Of Black Box Testing

Black box testing is a methodology of performing tests. These tests can be designed to accomplish a few different goals, including:

* **Functional Testing:** Functional testing is intended to validate that an application does what it is supposed to do. For example, functional tests may test an application’s authentication mechanism to check that legitimate users can authenticate successfully while invalid login attempts are rejected. Common types of functional testing include sanity checks, integration testing, and system testing.
* **Non-Functional Testing:** Non-functional testing evaluates how well an application performs its core functions. Examples of tests include performance, usability, scalability, and security testing.
* **Regression Testing:** Regression testing is designed to ensure that a change to an application does not break functionality. For example, regression testing should be performed after patching a vulnerability in an application to ensure the patch has not caused the application to fail functional or non-functional tests.

## Black Box Testing Techniques

With no internal knowledge of an application, structure is important to ensure that the test covers all necessary cases. Some common techniques for performing a black box evaluation include:

**Equivalence Class Testing:** An application may follow the same control flow for certain types of inputs. For example, an application that should only be accessible to adults may terminate if a user enters an age under 18 or a tool with a limited service area may terminate for country or postal codes outside of that area. With equivalence class testing, testers identify these classes that produce the same results and only test for one value within that class.

**Boundary Value Evaluation:** Boundary values are inputs where an application’s changes from one control flow to another. For example, the ages 17 and 18 are boundary values for adulthood since a 17 year old may be rejected by an application, while an 18 year old would be accepted. Boundary value evaluation tests these inputs to ensure that the system is properly handling these edge cases.

**Decision Table Testing:** An application may be designed to make decisions based on a combination of inputs. For example, users over the age of 18 and living within a particular area may be able to access an application. Decision table testing involves enumerating each combination of inputs and its expected outcomes and developing a test case to validate each combination.

**State Transition Evaluation:** An application may be designed to change state under certain conditions, such as locking a user’s account after a certain number of failed authentication attempts. State transition evaluation involves identifying these situations and developing test cases to validate them.

**Error Checking:** This form of evaluation tests for common errors that a developer may have made when creating an application. This often revolves around input sanitization and ensuring that assumptions about an input are enforced. For example, testers may check to see if developers properly handled an input of zero in a numeric field or restricted the character set for a name to the letters and symbols that can appear in a name.

## Tools used for Black Box Testing:

Tools used for Black box testing largely depends on the type of black box testing you are doing.

* For Functional/ Regression Tests you can use – [QTP](https://www.guru99.com/quick-test-professional-qtp-tutorial.html), [Selenium](https://www.guru99.com/selenium-tutorial.html)
* For Non-Functional Tests, you can use – [LoadRunner](https://www.guru99.com/loadrunner-v12-tutorials.html), [Jmeter](https://www.guru99.com/jmeter-tutorials.html)

## Black Box vs White Box Testing

While black box testing is named for the fact that the tester has no internal knowledge of the application (i.e. it’s a “black box”), a white box evaluation takes the opposite approach. Some of the key differences between black box and white box testing include:

* **Black Box Testing:** The tester interacts with the application and attempts to validate that an application meets functional and non-functional requirements and specifications. The lack of internal knowledge can make these tests more time-consuming and may cause vulnerabilities in unvisited code paths to go undetected. However, it has the advantage of being language and platform-agnostic.
* **White Box Testing:** As opposed to black box testing, white box evaluations are performed with full knowledge of an application’s internals, including access to the source code. White box testing offers better test coverage than black box testing since all code can be evaluated. However, it requires expertise with the language in which the code was developed.

Black box and white box testing represent two extremes in how testing can be performed. Gray box testing falls in between. In a gray box evaluation, the tester has partial knowledge of the system’s internals, which can help to guide the evaluation.  [Runtime application self-protection](https://www.checkpoint.com/cyber-hub/cloud-security/what-is-runtime-application-self-protection-rasp/) (RASP) is a security tool that falls into the gray box testing category.

**Differences between Black Box Testing vs White Box Testing:**

| **Black Box Testing** | **White Box Testing** |
| --- | --- |
| It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it. | It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software. |
| Implementation of code is not needed for black box testing. | Code implementation is necessary for white box testing. |
| It is mostly done by software testers. | It is mostly done by software developers. |
| No knowledge of implementation is needed. | Knowledge of implementation is required. |
| It can be referred to as outer or external software testing. | It is the inner or the internal software testing. |
| It is a functional test of the software. | It is a structural test of the software. |
| This testing can be initiated based on the requirement specifications document. | This type of testing of software is started after a detail design document. |
| No knowledge of programming is required. | It is mandatory to have knowledge of programming. |
| It is the behavior testing of the software. | It is the logic testing of the software. |
| It is applicable to the higher levels of testing of software. | It is generally applicable to the lower levels of software testing. |
| It is also called closed testing. | It is also called as clear box testing. |
| It is least time consuming. | It is most time consuming. |
| It is not suitable or preferred for algorithm testing. | It is suitable for algorithm testing. |
| Can be done by trial and error ways and methods. | Data domains along with inner or internal boundaries can be better tested. |
| **Example:** Search something on google by using keywords | **Example:** By input to check and verify loops |
| **Black-box test design techniques-**   * Decision table testing * All-pairs testing * Equivalence partitioning * Error guessing | **White-box test design techniques-**   * Control flow testing * Data flow testing * Branch testing |
| **Types of Black Box Testing:**   * Functional Testing * Non-functional testing * Regression Testing | **Types of White Box Testing:**   * Path Testing * Loop Testing * Condition testing |
| It is less exhaustive as compared to white box testing. | It is comparatively more exhaustive than black box testing. |

**Black Box Security Testing with Check Point**

Check Point Professional Services offers a range of Cybersecurity Resilience/Penetration Testing services. This includes black box, gray box, and whitebox security assessments.

Learn more about [Check Point’s professional testing services](https://www.checkpoint.com/professional-services/). You’re also welcome to [contact us](https://www.checkpoint.com/about-us/contact-us/) to learn how we can help to identify and correct security issues within your organization

# **White Box Testing**

The white box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and transparent box testing**. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

White Box Testing Techniques

**Statement Coverage:** Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.

**Decision Coverage:** This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

**Control flow Coverage:** Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.

**Branch Coverage:** Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.

**Basic path Coverage:** **Basis Path Testing** is a white box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing.

# **Statement Coverage Testing**

Statement coverage is one of the widely used software testing. It comes under white box testing.

Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.

Statement coverage derives scenario of test cases under the white box testing process which is based upon the structure of the code.

Statement Coverage 

Source Code Structure:

* Take input of two values like a=0 and b=1.
* Find the sum of these two values.
* If the sum is greater than 0, then print "This is the positive result."
* If the sum is less than 0, then print "This is the negative result."

**Scenario 1:**  
**If a = 5, b = 4**

1. print (**int** a, **int** b) {
2. **int** sum = a+b;
3. **if** (sum>0)
4. print ("This is a positive result")
5. **else**
6. print ("This is negative result")
7. }

In scenario 1, we can see the value of sum will be 9 that is greater than 0 and as per the condition result will be "**This is a positive result.**" The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 5.\

1. Total number of statements = 7
2. Number of executed statements = 5

Statement Coverage link

Statement coverage = 5/7\*100

= 500/7

= 71%

**Scenario 2:**  
**If A = -2, B = -7**

1. print (**int** a, **int** b) {
2. **int** sum = a+b;
3. **if** (sum>0)
4. print ("This is a positive result")
5. **else**
6. print ("This is negative result")
7. }

In scenario 2, we can see the value of sum will be -9 that is less than 0 and as per the condition, result will be "**This is a negative result.**" The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 6.

Total number of statements = 7  
Number of executed statements = 6

Statement Coverage link

1. Statement coverage = 6/7\*100 <br>
2. = 600/7
3. = 85%

# **Decision Coverage Testing**

Decision coverage technique comes under white box testing which gives decision coverage to Boolean values. This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like **do while statement, if statement and case statement** (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Decision Coverage technique in whitebox testing link

**Let's understand it by an example.**

Consider the code to apply on decision coverage technique:

1. Test (**int** a)
2. {
3. If(a>4)
4. a=a\*3
5. Print (a)
6. }

**Scenario 1:**  
**Value of a is 7 (a=7)**

1. Test (**int** a=7)
2. { **if** (a>4)
3. a=a\*3
4. print (a)
5. }

The code highlighted in yellow is executed code. The outcome of this code is "True" if condition (a>4) is checked.

Control flow graph when the value of a is 7.

Decision Coverage technique in whitebox testing link

Calculation of Decision Coverage percent:

Decision Coverage technique in whitebox testing link

1. Decision Coverage = ½\*100  (Only "True" is exercised)
2. =100/2
3. = 50
4. Decision Coverage is 50%

**Scenario 2:**  
**Value of a is 3 (a=3)**

1. Test (**int** a=3)
2. { **if** (a>4)
3. a=a\*3
4. print (a)
5. }

The code highlighted in yellow will be executed. The outcome of this code is ?False? if condition (a>4) is checked.

Control flow graph when the value of a is 3

Decision Coverage technique in whitebox testing link

Calculation of Decision Coverage percent:

Decision Coverage technique in whitebox testing link

1. = ½\*100  (Only "False" is exercised) <br>
2. =100/2
3. = 50
4. Decision Coverage = 50%

Result table of Decision Coverage:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Value of A** | **Output** | **Decision Coverage** |
| 1 | 3 | 3 | 50% |
| 2 | 7 | 21 | 50% |

# **Control Flow Testing**

Control flow testing is a testing technique that comes under white box testing. The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. It is mostly used in unit testing. Test cases represented by the control graph of the program.

**Control Flow Graph** is formed from the node, edge, decision node, junction node to specify all possible execution path.

## Notations used for Control Flow Graph

1. Node
2. Edge
3. Decision Node
4. Junction node

## Node

Nodes in the control flow graph are used to create a path of procedures. Basically, it represents the sequence of procedures which procedure is next to come so, the tester can determine the sequence of occurrence of procedures.

## Edge

Edge in control flow graph is used to link the direction of nodes.

## Decision node

Decision node in the control flow graph is used to decide next node of procedure as per the value.

## Junction node

Junction node in control flow graph is the point where at least three links meet.

### **Example**

1. **public** **class** VoteEligiblityAge{
3. **public** **static** **void** main(String []args){
4. **int** n=45;
5. **if**(n>=18)
6. {
7. System.out.println("You are eligible for voting");
8. }  **else**
9. {
10. System.out.println("You are not eligible for voting");
11. }
12. }
13. }

**Diagram - control flow graph**

Control Flow Testing in white box testing Link

# **Branch Coverage Testing**

Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once. Branch coverage technique is a whitebox testing technique that ensures that every branch of each decision point must be executed.

However, branch coverage technique and decision coverage technique are very similar, but there is a key difference between the two. Decision coverage technique covers all branches of each decision point whereas branch testing covers all branches of every decision point of the code.

Branch Coverage

How to calculate Branch coverage?

There are several methods to calculate Branch coverage, but pathfinding is the most common method.

In this method, the number of paths of executed branches is used to calculate Branch coverage. Branch coverage technique can be used as the alternative of decision coverage. Somewhere, it is not defined as an individual technique, but it is distinct from decision coverage and essential to test all branches of the control flow graph.

**Let's understand it with an example:**

1. Read X
2. Read Y
3. IF X+Y > 100 THEN
4. Print "Large"
5. ENDIF
6. If X + Y<100 THEN
7. Print "Small"
8. ENDIF

This is the basic code structure where we took two variables X and Y and two conditions. If the first condition is true, then print "Large" and if it is false, then go to the next condition. If the second condition is true, then print "Small."

Control flow graph of code structure

Branch Coverage

In the above diagram, control flow graph of code is depicted. In the first case traversing through "Yes "decision, the path is **A1-B2-C4-D6-E8**, and the number of covered edges is 1, 2, 4, 5, 6 and 8 but edges 3 and 7 are not covered in this path. To cover these edges, we have to traverse through "No" decision. In the case of "No" decision the path is A1-B3-5-D7, and the number of covered edges is 3 and 7. So by traveling through these two paths, all branches have covered.

**Path 1** - A1-B2-C4-D6-E8

**Path 2** - A1-B3-5-D7

Branch Coverage (BC) = Number of paths

=2

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Covered Branches** | **Path** | **Branch coverage** |
| Yes | 1, 2, 4, 5, 6, 8 | A1-B2-C4-D6-E8 | 2 |
| No | 3,7 | A1-B3-5-D7 |

## What is Basis Path Testing?

Basis path testing, a structured testing or white box testing technique used for designing test cases intended to examine all possible paths of execution at least once. Creating and executing tests for all possible paths results in 100% statement coverage and 100% branch coverage.

## Example:

Function fn\_delete\_element (int value, int array\_size, int array[])

{

1 int i;

location = array\_size + 1;

2 for i = 1 to array\_size

3 if ( array[i] == value )

4 location = i;

end if;

end for;

5 for i = location to array\_size

6 array[i] = array[i+1];

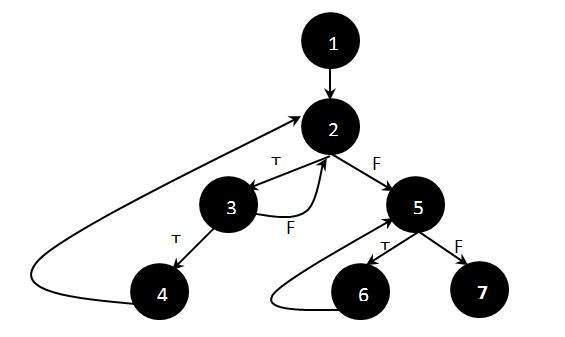
end for;

7 array\_size --;

}

## Steps to Calculate the independent paths

**Step 1 :**Draw the Flow Graph of the Function/Program under consideration as shown below:



**Step 2 :**Determine the independent paths.

Path 1: 1 - 2 - 5 - 7

Path 2: 1 - 2 - 5 - 6 - 7

Path 3: 1 - 2 - 3 - 2 - 5 - 6 - 7

Path 4: 1 - 2 - 3 - 4 - 2 - 5 - 6 - 7

# **Cyclomatic Complexity**

Cyclomatic complexity is a software metric used to measure the complexity of a program.McCabe interprets a computer program as a set of a strongly connected directed graph. Nodes represent parts of the source code having no branches and arcs represent possible control flow transfers during program execution. The notion of program graph has been used for this measure, and it is used to measure and control the number of paths through a program. The complexity of a computer program can be correlated with the topological complexity of a graph.

## How to Calculate Cyclomatic Complexity?

McCabe proposed the cyclomatic number, V (G) of graph theory as an indicator of software complexity. The cyclomatic number is equal to the number of linearly independent paths through a program in its graphs representation. For a program control graph G, cyclomatic number, V (G), is given as:

              V (G) = E - N + 2 \* P

E = The number of edges in graphs

N = The number of nodes in graphs G

P = The number of connected components in graph G.

## Example :

IF A = 10 THEN

IF B > C THEN

A = B

ELSE

A = C

ENDIF

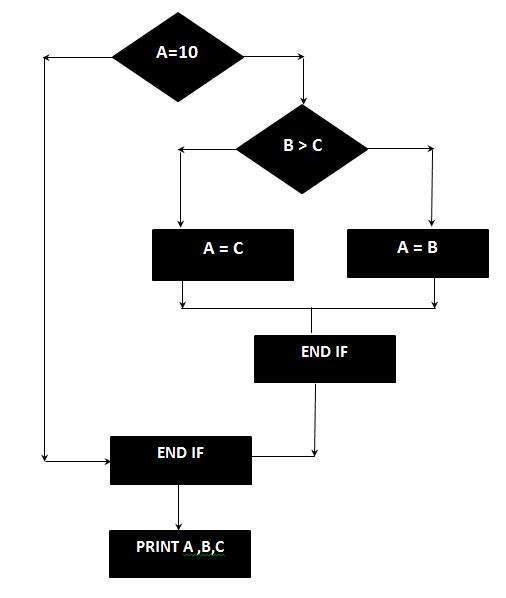
ENDIF

Print A

Print B

Print C

## FlowGraph:



The Cyclomatic complexity is calculated using the above control flow diagram that shows seven nodes(shapes) and eight edges (lines), hence the cyclomatic complexity is

8 - 7 + 2 = 3

PERFORMANCE TESTING

* It is the most important part of non-functional testing.
* *Checking the behaviour of an application by applying some load is known as performance testing.*
* Generally, this testing defines how quickly the server responds to the user's request.

While doing performance testing on the application, we will concentrate on the various factors like **Response time, Load, and Stability** of the application.

**Response time:** Response time is the time taken by the server to respond to the client's request.

**Load:** Here, Load means that when **N-number** of users using the application simultaneously or sending the request to the server at a time.

**Stability:** For the stability factor, we can say that, when N-number of users using the application simultaneously for a particular time.

## When we use performance testing?

We will do performance testing once the software is stable and moved to the production, and it may be accessed by the multiple users concurrently, due to this reason, some performance issues may occur. To avoid these performance issues, the tester performs one round of performance testing.

Since it is non-functional testing which doesn't mean that we always use performance testing, we only go for performance testing when the application is functionally stable.

**Note: Performance testing cannot be done manually since its costly and accurate result can't be maintained.**

## Performance testing process

The performance testing cannot be done manually since:

* We need a lot of resources, and it became a costlier approach.
* And the accuracy cannot maintain when we track response time manually.

**The Performance testing process will be completed in the following steps:**

* Identify performance scenarios
* Plan and design performance test script
* Configure the test environment & distribute the load
* Execute test scripts
* Result
* Analysis result
* Identify the Bottleneck
* Re-run test

## Performance Testing

## Performance Testing Attributes:

**Speed:**   
It determines whether the software product responds rapidly.

**Scalability:**   
It determines amount of load the software product can handle at a time.

**Stability:**   
It determines whether the software product is stable in case of varying workloads.

**Reliability:**   
It determines whether the software product is secure or not.

## Types of Performance Testing

**Load testing –** checks the application’s ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

Tools: Apache JMeter, webload, Load runner.

**Stress testing –** involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.

Tools: SmartMeter, Loadninja, LoadNinja

**Endurance testing –** is done to make sure the software can handle the expected load over a long period of time.

Tools:Loadstorm, Appvance

**Spike testing –** tests the software’s reaction to sudden large spikes in the load generated by users.

Tools: Apache JMeter

**Volume testing** – Under Volume Testing large no. of. Data is populated in a database, and the overall software system’s behaviour is monitored. The objective is to check software application’s performance under varying database volumes.

Tools: HammerDB, DbFit

**Scalability testing**– The objective of scalability testing is to determine the software application’s effectiveness in “scaling up” to support an increase in user load. It helps plan capacity addition to your software system.

ToolS: Gatling, HP LoadRunner

## Advantages of Performance Testing :

* Performance testing ensures the speed, load capability, accuracy and other performances of the system.
* It identifies, monitors and resolves the issues if anything occurs.
* It ensures the great optimization of the software and also allows large number of users to use it on same time.
* It ensures the client as well as end-customers satisfication. Performance testing has several advantages that make it an important aspect of software testing:

## Disadvantages of Performance Testing :

* Sometimes, users may find performance issues in the real time environment.
* Team members who are writing test scripts or test cases in the automation tool should have high-level of knowledge.
* Team members should have high proficiency to debug the test cases or test scripts.
* Low performances in the real environment may lead to lose large number of users
* Performance testing also has some disadvantages, which include:
* Resource-intensive: Performance testing can be resource-intensive, requiring significant hardware and software resources to simulate a large number of users or transactions. This can make performance testing expensive and time-consuming.

Security Testing

**Security Testing** is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss.

## Goal of Security Testing

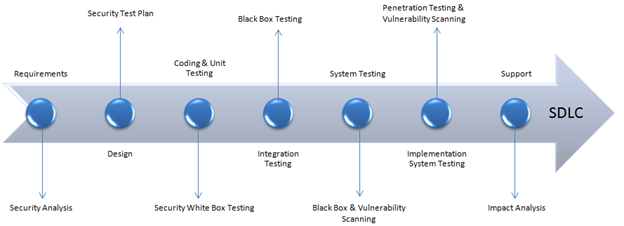
* To identify the threats in the system.
* To measure the potential vulnerabilities of the system.
* To help in detecting every possible security risks in the system.
* To help developers in fixing the security problems through coding.

**Principle of Security Testing:**

* Confidentiality
* Integrity
* Authentication
* Authorization
* Availability
* Non-repudiation

**Major Focus Areas in Security Testing:**

* Network Security
* System Software Security
* Client-side Application Security
* Server-side Application Security



## Types of Security Testing:

1. **Vulnerability Scanning:** Vulnerability scanning is performed with the help of automated software to scan a system to detect the known vulnerability patterns.

**Tools:** OpenVas, Intruder.

1. **Security Scanning:** Security scanning is the identification of network and system weaknesses. Later on it provides solutions for reducing these defects or risks. Security scanning can be carried out in both manual and automated ways.

**Tools:** Nmap, Qualys Guard.s

1. **Penetration Testing:** Penetration testing is the simulation of the attack from a malicious hacker. It includes an analysis of a particular system to examine for potential vulnerabilities from a malicious hacker that attempts to hack the system.

**Tools:** Metasploit, Wireshark.

1. **Risk Assessment:** In risk assessment testing security risks observed in the organization are analyzed. Risks are classified into three categories i.e., low, medium and high. This testing endorses controls and measures to minimize the risk.

**Tools:**Risk Matrix, Decision Tree.

1. **Security Auditing:** Security auditing is an internal inspection of applications and operating systems for security defects. An audit can also be carried out via line-by-line checking of code.

**Tools:**Burp suite, Netwrix.

1. **Ethical Hacking:** Ethical hacking is different from malicious hacking. The purpose of ethical hacking is to expose security flaws in the organization’s system.

**Tools:** Nmap, Kismet, Acunetix.

1. **Posture Assessment:** It combines security scanning, ethical hacking and risk assessments to provide an overall security posture of an

**Tools:** Goniometer

1. **Application security testing:** Application security testing is a type of testing that focuses on identifying vulnerabilities in the application itself. It includes testing the application’s code, configuration, and dependencies to identify any potential vulnerabilities.

**Tools:** OWASP ZAP, Sqlmap .

1. **Network security testing**: Network security testing is a type of testing that focuses on identifying vulnerabilities in the network infrastructure. It includes testing firewalls, routers, and other network devices to identify potential vulnerabilities.

**Tools**:Nessus,Wireshark.

1. **Social engineering testing**: Social engineering testing is a type of testing that simulates phishing, baiting, and other types of social engineering attacks to identify vulnerabilities in the system’s human element.

**Tool**:Maltego, Wifiphisher.

**Advantages**

1. Identifying vulnerabilities: Security testing helps identify vulnerabilities in the system that could be exploited by attackers, such as weak passwords, unpatched software, and misconfigured systems.
2. Improving system security: Security testing helps improve the overall security of the system by identifying and fixing vulnerabilities and potential threats.
3. Ensuring compliance: Security testing helps ensure that the system meets relevant security standards and regulations, such as HIPAA, PCI DSS, and SOC2.
4. Reducing risk: By identifying and fixing vulnerabilities and potential threats before the system is deployed to production, security testing helps reduce the risk of a security incident occurring in a production environment.
5. Improving incident response: Security testing helps organizations understand the potential risks and vulnerabilities that they face, enabling them to prepare for and respond to potential security incidents.

**Disadvantages:**

1. Resource-intensive: Security testing can be resource-intensive, requiring significant hardware and software resources to simulate different types of attacks.
2. Complexity: Security testing can be complex, requiring specialized knowledge and expertise to set up and execute effectively.
3. Limited testing scope: Security testing may not be able to identify all types of vulnerabilities and threats.
4. False positives and negatives: Security testing may produce false positives or false negatives, which can lead to confusion and wasted effort.
5. Time-consuming: Security testing can be time-consuming, especially if the system is large and complex.
6. Difficulty in simulating real-world attacks: It’s difficult to simulate real-world attacks, and it’s hard to predict how attackers will interact with the system.

**Tool for Security Testing**

**1. Unwelcome Visitor**

The intruder is a user-friendly enterprise-grade vulnerability scanner. It performs over 10,000 high-quality security checks throughout your IT infrastructure, including, but not limited to, configuration flaws, application flaws (such as SQL injection and cross-site scripting), and patches that are missing. Intruder saves time and keeps organizations of all sizes secure from hackers by providing intelligently prioritized results as well as proactive scans for the newest threats.

Features

Connectors for AWS, Azure, and Google Cloud

Results tailored to your perimeter to decrease your exterior attack surface

Reporting of exceptional quality

Integrations with Slack, Microsoft Teams, Jira, and Zapier

Integration with your CI/CD process using APIs

**2. The Owasp**

The Open Web Application Security Project (OWASP) is a non-profit organization dedicated to making software more secure. Multiple tools are available for pen testing different software environments and protocols as part of the project. The project's flagship tools include

Zed Attack Proxy (ZAP - an integrated penetration testing tool) is a program that allows you to test your network for vulnerabilities.

Check for OWASP Dependencies (it scans for project dependencies and checks against know vulnerabilities)

Web Testing Environment Project (OWASP) (collection of security tools and documentation)

**3. Acunetix**

Acunetix by Invicti is a simple and easy-to-use tool that helps small and medium-sized businesses protect their online applications against expensive data breaches. It does this by identifying a broad variety of online security concerns and assisting security and development experts in resolving them quickly.

**Features**

Scanning for over 7,000 online vulnerabilities, including OWASP Top 10 vulnerabilities like SQLi and XSS.

Automated online asset discovery can help you find websites that have been abandoned or forgotten.

Advanced web crawler with multi-form and password-protected regions for the most complicated online applications.

Using a combination of interactive and dynamic application security testing to find flaws that other technologies overlook

For a variety of vulnerabilities, proof of exploit is given.

Integrations with common issue tracking and CI/CD systems enable DevOps automation.

PCI DSS, NIST, HIPAA, ISO 27001, and other regulatory standards require compliance reporting.

**4. WireShark**

Wireshark, formerly known as Ethereal, is a network analysis tool. It catches packets in real-time and displays them in a way that is understandable to humans. It's essentially a network packet analyzer that gives you minute data about your network protocols, decryption, packet information, and so on. It's free and open-source, and it works with Linux, Windows, OS X, Solaris, NetBSD, FreeBSD, and a variety of other operating systems. The information acquired by this utility may be examined using a GUI or the TShark Utility in TTY mode.

**5. W3af**

W3af is a framework for web application attack and auditing. It has three types of plugins: discovery, audit, and attack, which communicate with one another to find any vulnerabilities in the site. For example, a discovery plugin in w3af looks for different urls to test for vulnerabilities and forwards them to the audit plugin, which then searches for vulnerabilities using

### 1. Vulnerability

Application vulnerability is a known or unknown weakness that hackers can use. Imagine a hole in the application that needs to be repaired and gives a chance to people that can get inside and access sensitive data. Insecure coding, unknown risks, updates, and business logic are considered as the top sources of application vulnerabilities.

**SQL injection:** It is a code injection approach where the destructive SQL Statements are implanted into some queries, and it is implemented by the server.

**Cross-site scripting (XSS):** This is the technique through which the user introduces client-side script or the HTML in the user-interface of a web application and those additions are visible to other users.

**TEST DATA MANAGEMENT**

Test Data management is very critical during the test life cycle. The amount of data that is generated is enormous for testing the application. Reporting the results it minimizes the time spent for processing the data and creating reports greatly contributes to the efficiency of an entire product.

## Test data Management - Checklist:

* identify common test data elements
* Aging, masking and archiving of test data
* Prioritization and allocation of test data
* Generating reports and dashboards for metrics
* Creating and implementing business rules
* Building an automation suite for master data preparation
* Masking, archiving and versioning aging of data

Test data management (TDM), a sister function of Test Environment Management, is the construction of test data sets that reliably represents an organization’s actual data so that IT teams (developers and testers) can effectively exercise software testing. Tests include Unit Testing, System Testing, Integration Testing, User Acceptance Testing, Performance Testing, and Security Testing.

Test Data Management Functions

Test Data Management is a broad subject and covers many facets often including:

**Test Data Requirements Gathering.** Gathering the needs of software testing i.e. identifying the data quality needed to build your test scenarios.

**Data Profiling / Data Discovery.** Understanding your Production Data. This is critical if you wish to understand what valid test data looks like, identify where personally identifiable information hides, and understand data security risk and data breach avoidance.

**Data Extraction** (often from Production data). Note: One benefit of using production data, as a base, is it will help ensure realistic test data.

**Data Transformation**

**+ Data Sub-setting.** That is condense or prepare test data so that it is both smaller & easier to handle.

**+ Data Masking / Data Privatization**. Protect data & protect customer PII through obfuscating sensitive data e.g. sensitive customer data. Note: Masking will usually use a copy of the original sensitive data i.e. production data. And by data masking, for example, data encryption, you are implementing data privacy to reduce the opportunity for a data breach.

**Test Data Provisioning** i.e. the deployment of test data.

**Data Cloning aka Data Virtualization** (the concept of snapshotting & deploying "tiny" replica databases). Note: Data cloning is rapidly displacing the need for test data subsetting. Cloning can also be used beyond software testing and against real data.

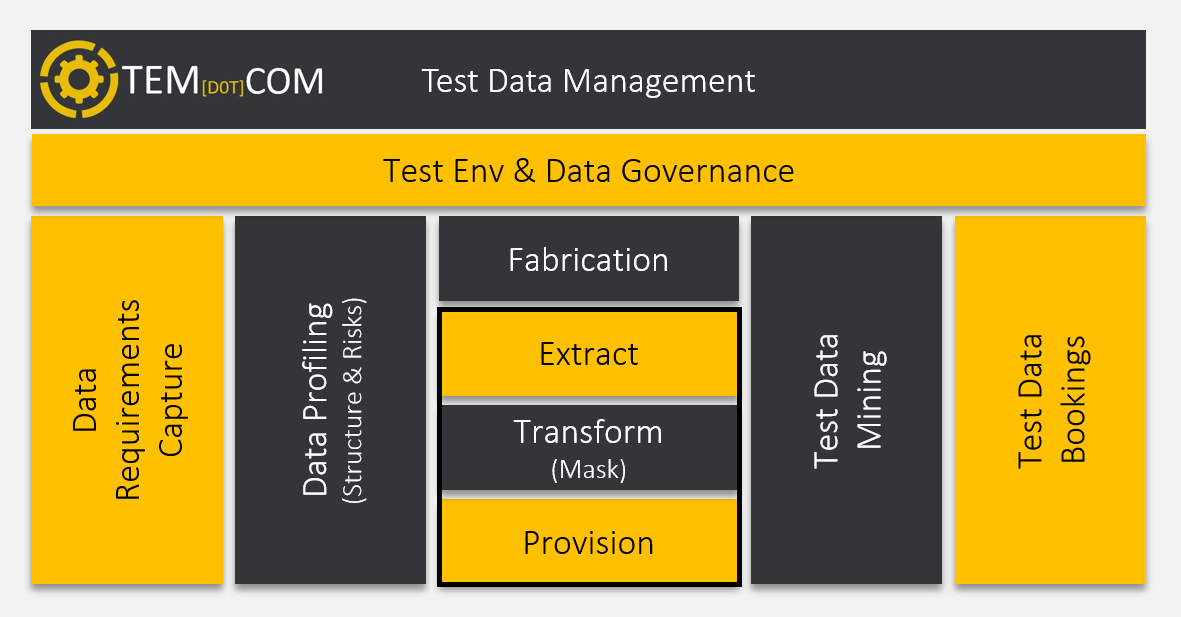
**Data Fabrication / Data Synthetics**. A method of test data preparation. Create test data using a data generation tool to create synthetic data (fake test data), from scratch.

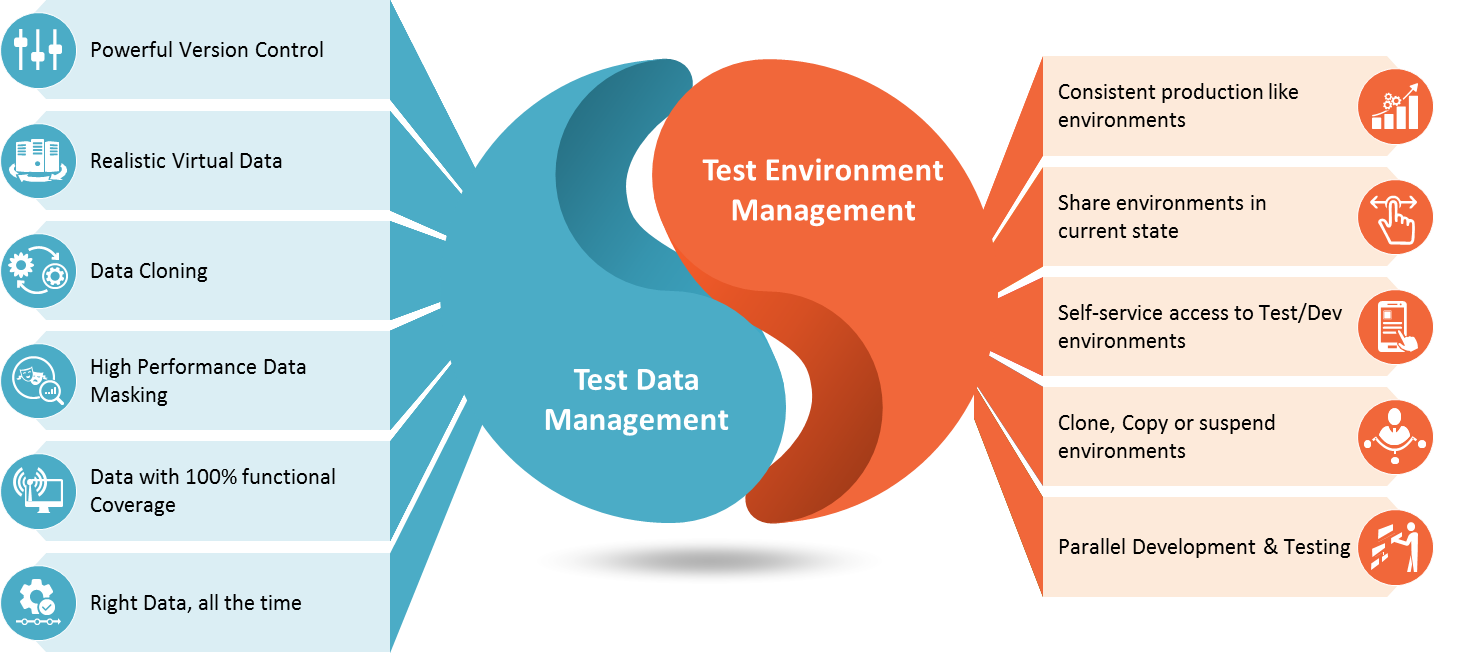
**DataOps** (any data operation that will help orchestrate data activities e.g. export, import, snapshot).

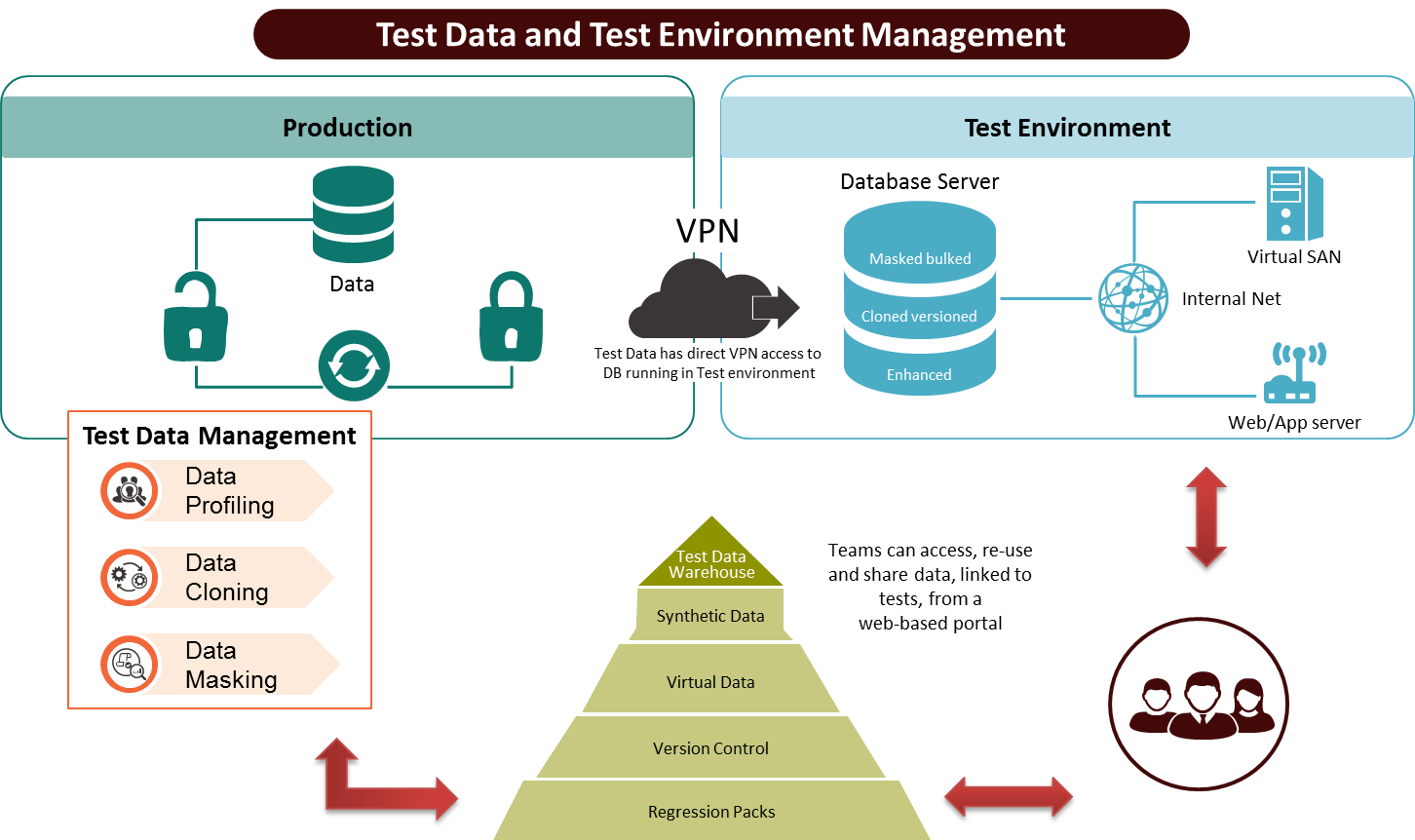
**Test Data Mining**. Methods to support ease of data access & helping software testing teams find sample data for testing. Note: Data mining can also be used to reduce the opportunity for data breaches by limiting what can be mind.

**Test Data Bookings**. Another method to simplify data access is Test Data Booking Management. Test Data Reservation methods help testing teams reserve test data to avoid test data contention.

**Test Data Reporting** e.g. Compliance, Size, Usage, Performance, etc.







**TEST ENVIRONMENT MANAGEMENT**

Test Environment Management (TEM) helps companies to speed up their software releases with close daily collaboration between all team members, to capture environment demands, to establish a simple and transparent environment utilization, to organize effective cooperation through planning and scheduling resources and control their lifecycles. TEM helps to streamline delivery by providing a validated, stable and available environment to execute any test scenarios or replicate bugs.

**Why is Test Environment Management important?**

Teams typically need a number of test environments for each production application, which depends on various circumstances. However, every business application may demand from as little as one to as many as, hundreds of test environments. IT architecture tendencies include distributed systems, various integrations, development, and testing across shared environments. All of these factors add additional complexity to managing projects, releases, and even achieving business goals.

With the rapid increase of ‘agile’ development environments, the necessity for regular and even more frequent software releases has also escalated, sometimes to the level of daily and hourly, or on-demand.

## What is the definition of a test environment?

A testing environment is a software and hardware configuration that allows testing teams to run test cases. In other words, it enables test execution with properly configured hardware, software, and network.

The test bed or test environment is set up according to the requirements of the Application Under Test. On rare occasions, a test bed may be a mix of the test environment and the test data with which it interacts.

Software testing success is ensured by setting up the correct test environment. Any errors in this procedure may result in the customer incurring more costs and effort.

This guide will teach you how to −

* What is the definition of a test environment?
* In the test environment, there are a few key locations to set up.
* Setup of a Software Testing Environment
* Management of the Testing Environment
* Checklist for the Testing Environment
* Setting up Test Environment Management presents a number of challenges.
* Setting up a Test Environment Management: Best Practices

## In the test environment, there are a few key locations to set up.

A critical place to set up for the test environment is −

* Applications and the system
* Data from the tests
* Server for databases
* Running environment for the front-end
* An operating system for the client
* Browser
* The server operating system is included in the hardware.
* Network
* Reference materials, setup guidelines, installation guides, and user manuals are all needed documentation.

## Setup of a Software Test Environment

What can be tested and what should not be examined are the limits of tests. The following personnel is engaged in the setup of the test environment −

* System Administrators,
* Developers
* Testers
* Occasionally, consumers or techies have a penchant for testing.

The test environment necessitates the creation of a variety of separate regions, such as,

## Configuration of the Test Server

Not every test can be run on a local computer. It may be necessary to set up a test server capable of supporting apps. For instance, Fedora configuration for PHP, Java-based apps with or without mail servers, cron configuration, Java-based applications, and so on.

## Network

Set up the network according to the test requirements. It contains the following items −

* Setting up the Internet
* Wifi setup on a LAN
* Setup of a private network

It assures that any congestion experienced while testing has no negative impact on other members. (Developers, designers, copywriters, and so on.)

## Setup a test PC

You may need to set up various browsers for different testers while doing web testing. For desktop applications, multiple kinds of operating systems are required for different testing PCs.

Windows Phone app testing, for example, may need

* Installation of Visual Studio
* Emulator for Windows Phone
* Assigning a Windows phone to the tester is another option.

## Reporting of Bugs

Testers should have access to bug reporting tools.

## Creating Test Data for the Test Environment

To test software products, many firms employ a distinct test environment. Copying production data to test is a popular practice. This allows the tester to discover the same faults as a live production server while avoiding data corruption.

The procedure for transferring production data to test data involves the following steps −

* Configure production tasks to transfer data to a shared test environment.
* All PII (Personally Identifiable Information) and other sensitive data are changed. The PII is replaced with non-personal data that is logically valid.
* Remove any information that isn't related to your exam.

This may be copied and pasted into each tester's or developer's test environment. They may change it to suit their needs.

In copy production data, privacy is the most pressing concern. You should look at obfuscated and anonymized test data to avoid privacy concerns.

There are two ways that may be used to anonymize data −

* All data fields are kept unmodified in the BlackList technique. Except for the fields that the users have chosen.
* WhiteList − This method anonymizes all data fields by default. With the exception of a list of fields that may be copied. A whitelisted field indicates that it is OK to replicate the data as is, with no need for anonymization.

Also, if you're utilizing production data, you'll need to think about how you're going to get it. Using SQL script to query the database is a good way to go.

## Management of the Testing Environment

Test Environment Management is in charge of the test bed's care and maintenance. The Test environment management function's list of activities includes −

* Maintenance of a central repository including all current test environment versions.
* Management of the test environment in accordance with the needs of the test team.
* Creating new environments in accordance with the new requirements
* Environmental monitoring is important.
* Test environments that are no longer in use are being updated or deleted.
* Environmental concerns are being investigated.
* Coordination until the problem is resolved.

## Checklist for the Testing Environment

**Hardware** −

* Find out whether the testing equipment you'll need is available. - If this isn't the case, look into the delivery time!
* Check to see if any more equipment is available- Scanners, special printers, handhelds, and other devices fall into this category.

**Connections / software** −

* Do you have a list of the apps you'll need?- A program such as Excel, Word, or Drawings, for example.
* Does the company have a test environment for the new software?- Has the company had any prior experience with the software's usage and upkeep?

**Information about the environment**−

* Do you have access to the normal test data sets?- Consider using the Defect administration to gather test data using the regression test set.
* Are there any agreements in place with the test data owners?- Consider the importance of effective upkeep.

**Tools and procedures for maintenance**−

* Is there a single point of contact for the upkeep of the test environment? If not, make a list of everyone who could be engaged in maintaining the test environment up and running. It should also provide their contact details.
* Is there a consensus on the preparedness and quality of the testing environment? Acceptance criteria, maintenance needs, and so forth. Also, see whether other/additional environmental quality criteria are in accord.
* Do you know who's engaged in the maintenance process?

Aside from this, there are a few additional questions to address before the test environment is set up.

* Developing an internal Test Environment or outsourcing?
* Which should you follow: an internal corporate standard or an external standard (IEE, ISO, etc.)?
* How long does the test environment have to be set up?
* Differences between the test and production systems must be established, as well as their influence on test validity.
* Is it possible to reuse an existing configuration for other corporate projects?

## Setting up Test Environment Management presents a number of challenges.

* Appropriate resource allocation planning - Ineffective resource utilization planning might have an impact on the final product. It may also rise to the rivalry between teams.
* In a remote location - It's feasible that a Test environment is physically separated from the rest of the world. The testing team will have to depend on the support team for different test assets in this situation. (This includes software, hardware, and other concerns.)
* Setup time is lengthy - When it comes to Integration Testing, the test setup may often be too complicated.
* Teams using it together - Test results will be compromised if the testing environment is utilized by both the development and testing teams at the same time.
* Configuration of a complex test - Certain tests need a complicated test environment setup. It might be a difficult task for the test crew.

## Setting up a Test Environment Management: Best Practices

* Understand the test requirements in detail and educate the members of the test team.
* Before starting the testing, make sure everything is connected.
* Check for necessary hardware and software, as well as licensing.
* Versions and browsers
* Scheduled usage of the test environment is being planned.
* The configurations of automation tools.

## In software testing, what is a test bed?

A software development environment is referred to as a test bed in software testing. It enables developers to test their modules without impacting the production servers in real-time. Test beds aren't only for developers; they're also utilized by testers. It's known as a test environment since it allows for thorough and transparent testing of new technologies.

## Summary

* A testing environment is a collection of software and hardware that the testing team will use to carry out the testing.
* A critical place to set up for the test environment is
  + Applications and the system
  + Data from the tests
  + Server for databases
  + The environment in which the front-end runs, and so on.
* The following are some of the difficulties encountered while setting up a test environment −
  + In a remote location
  + Collaboration between teams
  + Setup time is lengthy.
  + Ineffective resource utilization planning for integration
  + Configuration of a complex test