School of Computing

Department of Computer Science and Engineering

and

Department of Information Technology

Quality As

UNIT - III

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

UNIT III

Software Testing Strategies and Approaches

1. Static Testing Strategy

A static test evaluates the quality of a system without actually running the system. While that may seem impossible, it can be accomplished in a few ways.

The static test looks at portions of or elements related to the system in order to detect problems as early as possible. For example, developers review their code after writing and before pushing it. This is called desk-checking, a form of static testing. Another example of a static test would be a review meeting held for the purpose of evaluating requirements, design, and code.

Static tests offer a decided advantage: If a problem is detected in the requirements before it develops into a bug in the system, it will save time and money. If a preliminary code review leads to bug detection, it saves the trouble of building, installing, and running a system to find and fix the bug.

It is possible to perform automated static tests with the right tools. C programmers can use the lint program to identify potential bugs while Java users can utilize JTest to check their scripts against a coding standard.

Static tests must be performed at the right time. For example, if requirements are reviewed after developers have finished coding the entire software it can help testers design test cases. But testers cannot detect bugs in already written code without running the system, thus defeating the purpose of static tests. In this case, the code must be reviewed by individual developers as soon as it is created, and before it is integrated.

Additionally, static tests must be run not just by technical personnel, but other stakeholders. Business domain experts must review requirements, system architects must review design, and so on. Testers' feedback is also imperative since they are trained to spot inconsistencies, missing details, vague functionality, etc.

2. Structural Testing Strategy

While static tests are quite useful, they are not adequate. The software needs to be operated on real devices, and the system has to be run in its entirety to find all bugs. Structural tests are among the most important of these tests.

Structural tests are designed on the basis of the software structure. They can also be called white-box tests because they are run by testers with thorough knowledge of the software as well as the devices and systems it is functioning on. Structural tests are most often run on individual components and interfaces in order to identify localized errors in data flows.

A good example of this would be using reusable, automated test harnesses for the system being tested. With this harness in place, coders can create structural test cases for components right after they have written the code for each component. Then, they register the tests into the source code repository along with the main component during integration. A well-crafted test harness will run the tests every time new code is added, thus serving as a regression test suite.

Since creating structural tests require a thorough understanding of the software being tested, it is best that they are executed by developers or highly skilled testers. In the best-case scenario, developers and testers work in tandem to set up test harnesses and run them at regular intervals. Testers are especially helpful when it comes to developing reusable and shareable test scripts and cases, which cut down on time and effort in the long run.

3. Behavioral Testing Strategy

Behavioral Testing focuses on how a system acts rather than the mechanism behind its functions. It focuses on workflows, configurations, performance, and all elements of the user journey. The point of these tests, often called "black box" tests, is to test a website or app from the perspective of an end-user.

Behavioral testing must cover multiple user profiles as well as usage scenarios. Most of these tests focus on fully integrated systems rather than individual components. This is because it is possible to effectively gauge system behavior from a user's eyes, only after it has been assembled and integrated to a significant extent.

Behavioral tests are most frequently run manually, though some of them can be automated. Manual testing requires careful planning, design, and meticulous checking of results to detect what goes wrong. Skilled manual testers are known for being able to follow a trail of bugs and ascertain their effect on user experience.

Automation testing helps primarily to run repetitive actions, such as regression tests which check that new code has not disrupted already existing features that are working well. For example, a website needs to be tested by filling 50 fields in a form. Now this action needs to be repeated with multiple sets of values. Obviously, it is smarter to let a machine handle this rather than risk wasting time, human effort, and human error.

Behavioral testing does require some understanding of the system's technicality. Testers need some measure of insight into the business side of the software, especially with regard to what target users want. In order to plan test scenarios, they must know what users are likely to do once they access a website or app.

What to consider when choosing a software testing strategy?

A strategic approach to software testing must take the following into account:

- **Risks-** Risk management is very important during testing to figure out the risks and the risk level. For example, for an app that is well-established and slowly evolving, regression is a critical risk.
- **Objectives-** Testing should satisfy the requirements and needs of stakeholders to succeed. The objective is to look for as many defects as possible with less up-front time and effort invested.
- **Skills-** It is important to consider the skills of the testers since strategies should not only be chosen but executed as well. A standard-compliant strategy is a smart option when lacking skills and time in the team to create an approach.
- **Product-** Some products have specified requirements. This could lead to synergy with an analytical strategy that is requirements-based.
- **Business-** Business considerations and strategy are often important. If using a legacy system as a model for a new one, you could use a model-based strategy.

- **Regulations-** At some instances, one needs to satisfy the regulators along with the stakeholders. In this case, you would need a methodical strategy which satisfies these regulators.

The role of Real Devices: An Accurate Software Testing Approach

The point of all software testing is to identify bugs. Testers must be perfectly clear on how frequently a bug occurs and how it affects the software.

The best way to detect all bugs is to run software through real devices and browsers. When it comes to a website, ensure that it is under the purview of both manual testing and automation testing. Automated Selenium testing should supplement manual tests so that testers do not miss any bugs in the Quality Assurance process.

Websites must also be put through extensive cross browser testing so that they function consistently, regardless of the browser they are being accessed by. Using browsers installed on real devices is the only way to guarantee cross-browser compatibility and not alienate users of any browser.

The best option is to opt for a cloud-based testing service that provides real device browsers and operating systems. BrowserStack offers **2000+ real browsers and devices** for manual and automated testing. Users can sign up for free, log in, choose desired device-browser-OS combinations and start testing.

The same applies to apps. BrowserStack offers real devices for mobile app testing and automated app testing. Simply upload the app to the required device-OS combination and check to see how it functions in the real world.

Additionally, BrowserStack offers a wide range of debugging tools that make it easy to share and resolve bugs. This includes text and video logs to identify exactly where and why a test failed, thus letting testers zero in on what issue to work on.

A clear comprehension of test automation strategy is essential to building test suites, scripts and timelines that offer fast and accurate results. This is equally true for manual tests. Don’t start testing without knowing what techniques to use, what approach to follow and how the software is expected to perform. The information in this article intends to provide a starting point for building constructive testing plans, by detailing what strategies exist for testers to explore in the first place.

Strategy of testing

A strategy of software testing is shown in the context of spiral.

Following figure shows the testing strategy:

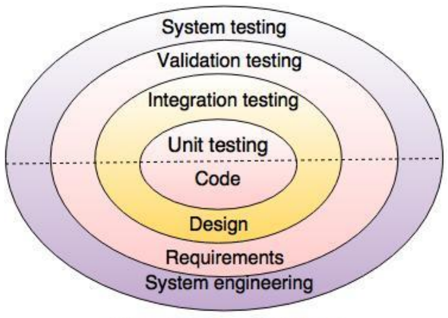


Fig. - Testing Strategy

Unit testing

Unit testing starts at the centre and each unit is implemented in source code.

Integration testing

An integration testing focuses on the construction and design of the software.

Validation testing

Check all the requirements like functional, behavioral and performance requirement are validate against the construction software.

System testing

System testing confirms all system elements and performance are tested entirely.

Testing strategy for procedural point of view

As per the procedural point of view the testing includes following steps.

1) Unit testing
2) Integration testing
3) High-order tests
4) Validation testing

These steps are shown in following figure:

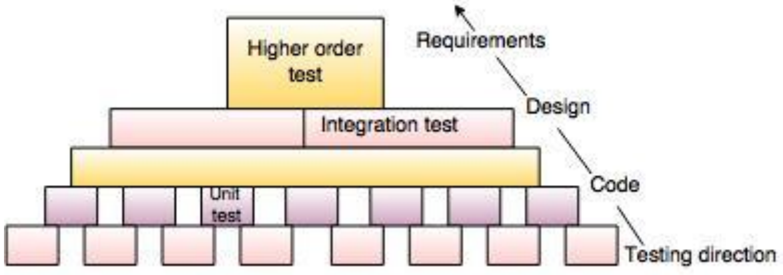
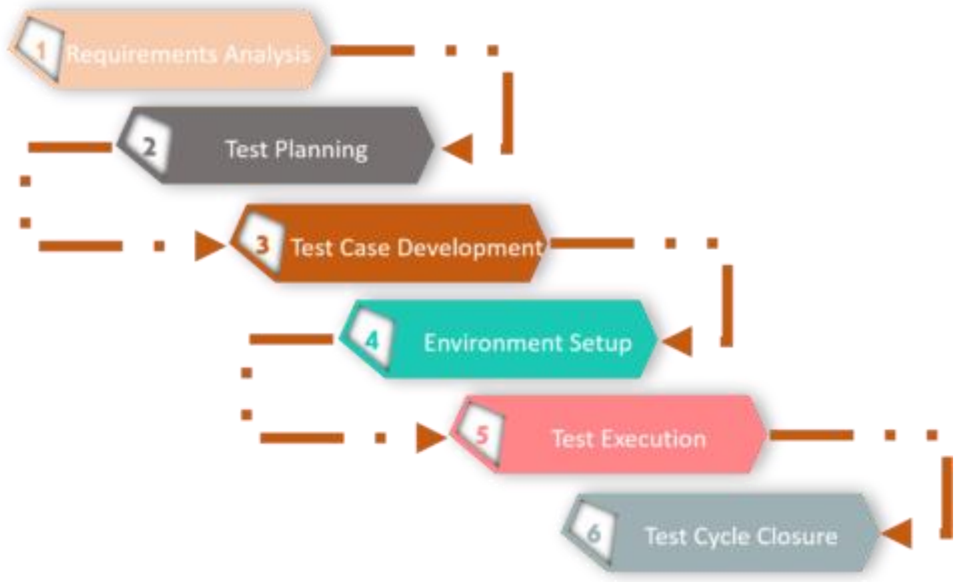


Fig.- Steps of software testing

Software Testing Life Cycle – Different Stages of Testing

What is Software Testing Life Cycle (STLC)?

Software Testing Life Cycle (STLC) defines a series of activities conducted to perform Software Testing. It identifies what test activities to carry out and when to accomplish those test activities. In the STLC process, each activity is carried out in a planned and systematic way and each phase has different goals and deliverables.



What are the different phases of Software Testing Life Cycle?

The different phases of Software testing life cycle are:

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

Now let's move ahead and have a look at the different phases of software testing life cycle in detail.

Requirement Analysis

Requirement Analysis is the first step involved in Software testing life cycle. In this step, Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements. During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.The QA team may interact with various stakeholders such as client, business analyst, technical leads, system architects etc. to understand the requirements in detail.

The different types of Requirements include :

Business Requirements – They are high-level requirements that are taken from the business case from the projects.

Architectural & Design Requirements – These requirements are more detailed than business requirements. It determines the overall design required to implement the business requirement.

System & Integration Requirements – It is detailed description of each and every requirement. It can be in form of user stories which is really describing everyday business language. The requirements are in abundant details so that developers can begin coding.

Entry Criteria	**Deliverable**
The following documents are required: • Requirements Specification. • Application architectural	• List of questions with all answers to be resolved from testable requirements • Automation feasibility report

Activities
• Prepare the list of questions or queries and get resolved from Business Analyst, System Architecture, Client, Technical Manager/Lead etc. • Make out the list for what all Types of Tests performed like Functional, Security, and Performance etc. • Define the testing focus and priorities. • List down the Test environment details where testing activities will be carried out.

- Checkout the Automation feasibility if required & prepare the Automation feasibility report.

Test Planning

Test Planning is the most important phase of Software testing life cycle where all testing strategy is defined. This phase is also called as **Test Strategy** phase. In this phase, the Test Manager is involved to determine the effort and cost estimates for the entire project. It defines the objective & scope of the project.

The commonly used Testing types are :

- Unit Test
- API Testing
- Integration Test
- System Test
- Install/Uninstall Testing
- Agile Testing

Test plan is one of the most important steps in software testing life cycle. The steps involved in writing a test plan include :

1. Analyze the product
2. Design Test Strategy
3. Define Test Objectives
4. Define Test Criteria
5. Resource Planning
6. Plan Test Environment
7. Schedule & Estimation
8. Determine Test Deliverable

Test Case Development

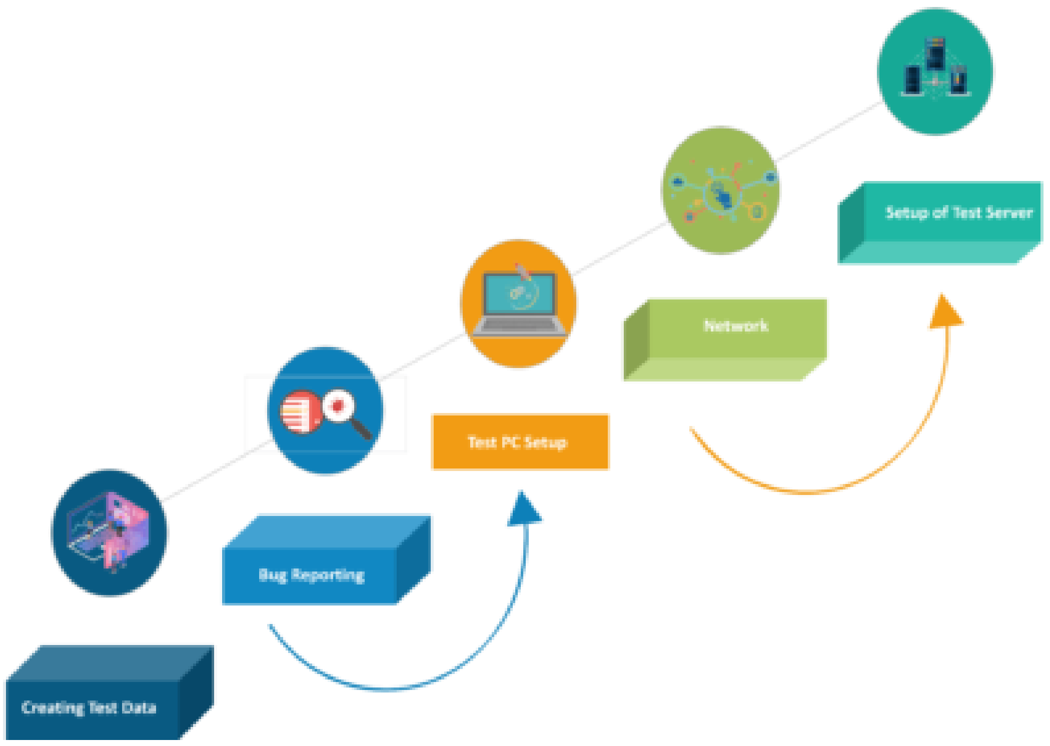
The Test case development begins once the test planning phase is completed. This is the phase of STLC where testing team notes the detailed test cases. Along with test cases, testing team also prepares the test data for testing. Once the test cases are ready then these test cases are reviewed by peer members or QA lead.

A good test case is the one which is effective at finding defects and also covers most of the scenarios on the system under test. Here is the step by step guide on how to develop a good test case :

- Test cases need to be simple and transparent
- Create test case with end user in mind
- Avoid test case repetition
- Do not assume functionality and features of your software application
- Ensure 100% coverage of software requirements
- Name the test case id such that they are identified easily while tracking defects
- Implement testing techniques
- The test case you create must return the Test Environment to the pre-test state
- The test case should generate the same results every time
- Your peers should be able to uncover defects in your test case design

Test Environment Setup

Setting up the test environment is vital part of the Software Testing Life Cycle. A testing environment is a setup of software and hardware for the testing teams to execute test cases. It supports test execution with hardware, software and network configured.



The test environment involves setting up of distinct areas like :

- **Setup of Test Server** – Every test may not be executed on a local machine. It may need establishing a test server, which can support applications.
- **Network** – We need to set up the network as per requirements.
- **Test PC Setup** – We need to set up different browsers for different testers.
- **Bug Reporting** – Bug reporting tools should be provided to testers.
- **Creating Test Data for the Test Environment** – Many companies use a separate test environment to test the software product. The common approach used is to copy production data to test.

Test Execution

The next phase in Software Testing Life Cycle is Test Execution. Test execution is the process of executing the code and comparing the expected and actual results. When test execution begins, the test analysts start executing the test scripts based on test strategy allowed in the project.

Entry Criteria	Deliverable
• Test Plan or Test strategy document. • Test cases. • Test data.	• Test case execution report. • Defect report.

Activities
• Mark status of test cases like Passed, Failed, Blocked, Not Run etc. • Assign Bug Id for all failed and blocked test cases. • Do Retesting once the defects are fixed. • Track the defects to closure.

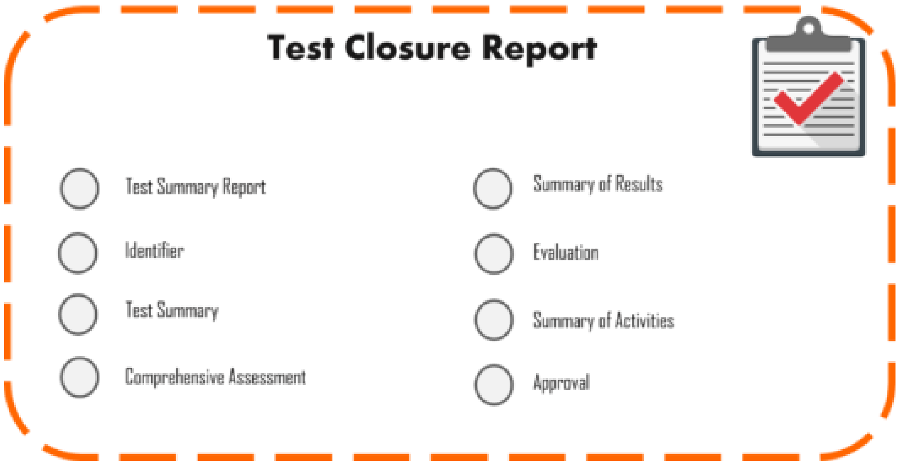
Test Cycle Closure

The final phase of the Software Testing Life Cycle is Test Cycle Closure. It involves calling out the testing team member meeting & evaluating cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software.

A test closure report by the test lead is published after accomplishing the exit criteria and finishing the testing phase. It follows a standard format such as :

- Test Summary Report
- Identifier
- Test Summary
- Variances
- Comprehensiveness Assessment
- Summary of Results

- Evaluation
- Summary of Activities
- Approval



Stages of Test Closure :

The process of test closure is implemented with the assistance of six important stages such as –

1. **Check planned Deliverable** – The planned deliverables that will be given to the stakeholder of the project are checked and analyzed by the team.
2. **Close Incident Reports** – The team checks that the planned deliverable are delivered and validates that all the incidents are resolved before the culmination of the process.
3. **Handover to Maintenance** – After resolving incidents and closing the incident report, the test-wares are then handed over to the maintenance team.
4. **Finalize & Archive Testware/Environment** – It involves finalizing and archiving of the testware and software like test scripts, test environment, test infrastructure, etc.
5. **Document System Acceptance** – It involves system verification and validation according to the strategy outlined.
6. **Analyze Best Practices** – It determines the various changes required for similar projects and their future release.

Let’s now move ahead with this article and understand the difference between SDLC and STLC.

What is SDLC and STLC in Software Testing?

SDLC	STLC
Stands for Software Development Life Cycle	Stands for Software testing Life Cycle
It refers to a sequence of various activities that are performed during the software development process	It refers to a sequence of various activities that are performed during the software testing process
Aims to complete the development of the software including testing and other phases successfully	Aims to evaluate the functionality of a software application to find any software bugs
In SDLC, the code for the software is built based on the design documents	In STLC, the test environment is created and various tests are carried out on the software

Now with this, we come to an end to this “Software Testing Life Cycle” blog. I hope you guys enjoyed this article and understood what is software testing and the different Types of Software testing.

What is Functional Testing?

FUNCTIONAL TESTING is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

Functional testing mainly involves black box testing and it is not concerned about the source code of the application. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other functionality of the Application Under Test. The testing can be done either manually or using automation.

What do you test in Functional Testing?

The prime objective of Functional testing is checking the functionalities of the software system. It mainly concentrates on -

- **Mainline functions**: Testing the main functions of an application
- **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- **Accessibility**: Checks the accessibility of the system for the user
- **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

Software Testing Methodologies

Functional vs. Non-functional Testing

The goal of utilizing numerous testing methodologies in your development process is to make sure your software can successfully operate in multiple environments and across different platforms. These can typically be broken down between functional and non-functional testing.

FUNCTIONAL TESTING is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

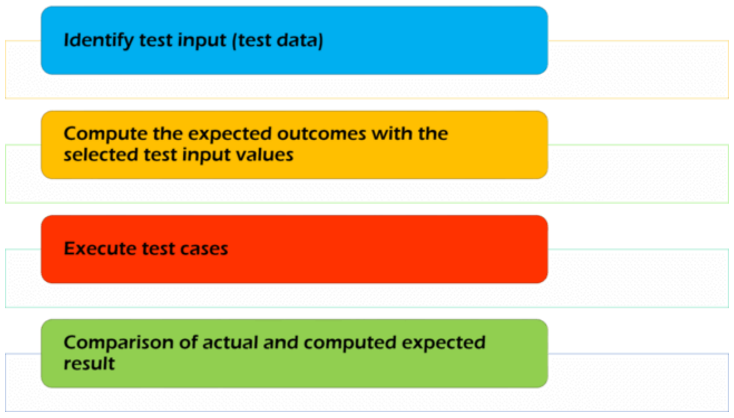
Functional testing involves testing the application against the business requirements. It incorporates all test types designed to guarantee each part of a piece of software behaves as expected by using uses cases provided by the design team or business analyst. These testing methods are usually conducted in order and include:

- Unit testing
- Integration testing
- System testing

- Acceptance testing

How to perform Functional Testing: Complete Process

In order to functionally test an application, the following steps must be observed.


Identify test input (test data)
Compute the expected outcomes with the selected test input values
Execute test cases
Comparison of actual and computed expected result

- Understand the Software Engineering Requirements
- Identify test input (test data)
- Compute the expected outcomes with the selected test input values
- Execute test cases
- Comparison of actual and computed expected result

Non-functional testing methods incorporate all test types focused on the operational aspects of a piece of software. These include:

- Performance testing
- Security testing
- Usability testing
- Compatibility testing

The key to releasing high quality software that can be easily adopted by your end users is to build a robust testing framework that implements both functional and non-functional software testing methodologies.

Unit Testing

Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to. Developers in a test-driven environment will typically write and run the tests prior to the software or feature being passed over to the test team. Unit testing can be conducted manually, but automating the process will speed up delivery cycles and expand test coverage. Unit testing will also make debugging easier because finding issues earlier means they take less time to fix than if they were discovered later in the testing process. TestLeft is a tool that allows advanced testers and developers to shift left with the fastest test automation tool embedded in any IDE.

Integration Testing

After each unit is thoroughly tested, it is integrated with other units to create modules or components that are designed to perform specific tasks or activities. These are then tested as group through integration testing to ensure whole segments of an application behave as expected (i.e, the interactions between units are seamless). These tests are often framed by user scenarios, such as logging into an application or opening files. Integrated tests can be conducted by either developers or independent testers and are usually comprised of a combination of automated functional and manual tests.

System Testing

System testing is a black box testing method used to evaluate the completed and integrated system, as a whole, to ensure it meets specified requirements. The functionality of the software is tested from end-to-end and is typically conducted by a separate testing team than the development team before the product is pushed into production.

Acceptance Testing

Acceptance testing is the last phase of functional testing and is used to assess whether or not the final piece of software is ready for delivery. It involves ensuring that the product is in compliance with all of the original business criteria and that it meets the end user’s needs. This requires the product be tested both internally and externally, meaning you’ll need to get it into the hands of your end users for beta testing along with those of your QA team. Beta testing is key to getting real feedback from potential customers and canaddress any final usability concerns.

Performance Testing

Performance testing is a non-functional testing technique used to determine how an application will behave under various conditions. The goal is to test its responsiveness and stability in real user situations. Performance testing can be broken down into four types:

- **Load testing** is the process of putting increasing amounts of simulated demand on your software, application, or website to verify whether or not it can handle what it’s designed to handle.
  - **Stress testing** takes this a step further and is used to gauge how your software will respond at or beyond its peak load. The goal of stress testing is to overloadthe application on purpose until it breaks by applying both realistic and unrealistic load scenarios. With stress testing, you’ll be able to find the failure point of your piece of software.
  - **Endurance testing,** also known as soak testing, is used to analyze the behavior of an application under a specific amount of simulated load over longer amounts of time. The goal is to understand how your system will behave under sustained use, making it a longer process than load or stress testing (which are designed to end after a few hours). A critical piece of endurance testing is that it helps uncover memory leaks.
  - **Spike testing** is a type of load test used to determine how your software will respond to substantially larger bursts of concurrent user or system activity over varying amounts of time. Ideally, this will help you understand what will happen when the load is suddenly and drastically increased.

Security Testing

With the rise of cloud-based testing platforms and cyber attacks, there is a growing concern and need for the security of data being used and stored in software. Security testing is a non-functional software testing technique used to determine if the information and data in a system is protected. The goal is to purposefully find loopholes and security risks in the system that could result in unauthorized access to or the loss of information by probing the application for weaknesses. There are multiple types of this testing method, each of which aimed at verifying six basic principles of security:

1. Integrity
2. Confidentiality
3. Authentication
4. Authorization
5. Availability
6. Non-repudiation

Usability Testing

Usability testing is a testing method that measures an application's ease-of-use from the end-user perspective and is often performed during the system or acceptance testing stages. The goal is to determine whether or not the visible design and aesthetics of an application meet the intended workflow for various processes, such as logging into an application. Usability testing is a great way for teams to review separate functions, or the system as a whole, is intuitive to use.

Compatibility Testing

Compatibility testing is used to gauge how an application or piece of software will work in different environments. It is used to check that your product is compatible with multiple operating systems, platforms, browsers, or resolution configurations. The goal is toensure that your software's functionality is consistently supported across any environment you expect your end users to be using.

Functional Vs Non-Functional Testing:

Functional Testing	**Non-Functional Testing**
Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements.	Non-Functional testing checks the Performance, reliability, scalability and other non-functional aspects of the software system.
Functional testing is executed first	Non-functional testing should be performed after functional testing
Manual Testing or automation tools can be used for functional testing	Using tools will be effective for this testing
Business requirements are the inputs to functional testing	Performance parameters like speed, scalability are inputs to non-functional testing.
Functional testing describes what the product does	Nonfunctional testing describes how good the product works
Easy to do Manual Testing	Tough to do Manual Testing

Examples of Functional testing are	Examples of Non-functional testing are
• Unit Testing	• Performance Testing
• Smoke Testing	• Load Testing
• Sanity Testing	• Volume Testing
• Integration Testing	• Stress Testing
• White box testing	• Security Testing
• Black Box testing	• Installation Testing
• User Acceptance testing	• Penetration Testing
• Regression Testing	• Compatibility Testing
	• Migration Testing

Defect

A Software **DEFECT / BUG / FAULT** is a condition in a software product which does not meet a software requirement (as stated in the requirement specifications) or end-user expectation (which may not be specified but is reasonable). In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/ unexpected results.

- **defect:** An imperfection or deficiency in a work product where it does not meet its requirements or specifications.

Related Terms

- A program that contains a large number of bugs is said to be ***buggy***.
- Reports detailing defects / bugs in software are known as ***defect reports / bug reports***.
- Applications for tracking defects bugs are known as ***defect tracking tools / bug tracking tools***.

- The process of finding the cause of bugs is known as ***debugging***.
- The process of intentionally injecting bugs in a software program, to estimate test coverage by monitoring the detection of those bugs, is known as ***bebugging***.

Software Testing proves that defects exist but NOT that defects do not exist.

Classification

Software Defects/ Bugs are normally classified as per:

- Severity / Impact
- Probability / Visibility
- Priority / Urgency
- Related Dimension of Quality
- Related Module / Component
- Phase Detected
- Phase Injected

Related Module /Component

Related Module / Component indicates the module or component of the software where the defect was detected. This provides information on which module / component is buggy or risky.

- Module/Component A
- Module/Component B
- Module/Component C
- …

Phase Detected

Phase Detected indicates the phase in the software development lifecycle where the defect was identified.

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Phase Injected

Phase Injected indicates the phase in the software development lifecycle where the bug was introduced. Phase Injected is always earlier in the software development lifecycle than the Phase Detected. Phase Injected can be known only after a proper root-cause analysis of the bug.

- Requirements Development
- High Level Design
- Detailed Design
- Coding
- Build/Deployment

Note that the categorizations above are just guidelines and it is up to the project/ organization to decide on what kind of categorization to use. In most cases, the categorization depends on the defect tracking tool that is being used. It is essential that project members agree beforehand on the categorization (and the meaning of each categorization) so as to avoid arguments, conflicts, and unhealthy bickering later.

Metrics

Some metrics related to Defects are:

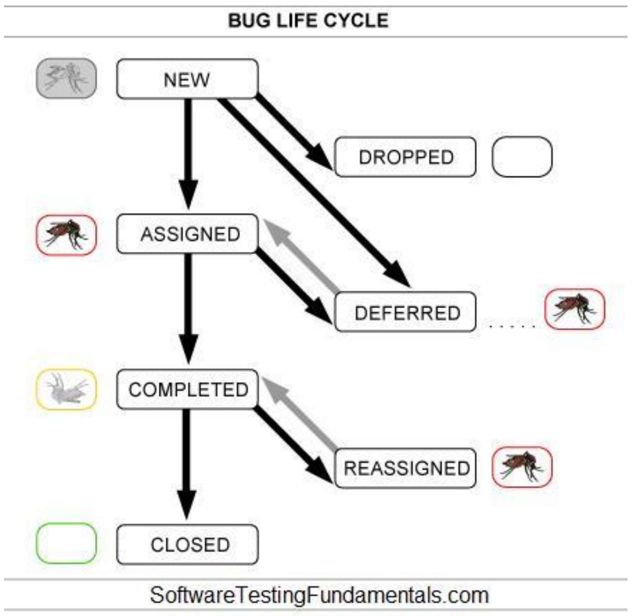
- Defect Age
- Defect Density
- Defect Detection Efficiency

Defect vs Bug

Strictly speaking, a BUG is a deficiency in just the software but a DEFECT could be a deficiency in the software as well as any work product (Requirement Specification, for example). You don't say 'There's a bug in the Test Case'; you say 'There's a defect in the Test Case.'We prefer the term 'Defect' over the term 'Bug' because 'Defect' is more comprehensive.

Defect Life Cycle

DEFECT LIFE CYCLE, also known as Bug Life Cycle, is the journey of a defect from its identification to its closure. The Life Cycle varies from organization to organization and is governed by the software testing process the organization or project follows and/or the Defect tracking tool being used. Nevertheless, the life cycle in general resembles the following:



Status

Status	**Alternative Status**
NEW	
ASSIGNED	**OPEN**
DEFERRED	
DROPPED	**REJECTED**
COMPLETED	**FIXED, RESOLVED, TEST**
REASSIGNED	**REOPENED**
CLOSED	**VERIFIED**

- *NEW*: Tester finds a defect and posts it with the status NEW. This defect is yet to be studied/approved. The fate of a NEW defect is one of ASSIGNED, DROPPED or DEFERRED.
- *ASSIGNED / OPEN*: Test / Development / Project lead studies the NEW defect and if it is found to be valid it is assigned to a member of the Development Team. The assigned Developer's responsibility is now to fix the defect and have it COMPLETED. Sometimes, ASSIGNED and OPEN can be different statuses. In that case, a defect can be open yet unassigned.
- *DEFERRED*: If a valid NEW or ASSIGNED defect is decided to be fixed in upcoming releases instead of the current release it is DEFERRED. This defect is ASSIGNED when the time comes.

- *DROPPED / REJECTED*: Test / Development/ Project lead studies the NEW defect and if it is found to be invalid, it is DROPPED / REJECTED. Note that the specific reason for this action needs to be given.
- *COMPLETED / FIXED / RESOLVED / TEST*: Developer 'fixes' the defect that is ASSIGNED to him or her. Now, the 'fixed' defect needs to be verified by the Test Team and the Development Team 'assigns' the defect back to the Test Team. A COMPLETED defect is either CLOSED, if fine, or REASSIGNED, if still not fine.
- If a Developer cannot fix a defect, some organizations may offer the following statuses:
- *Won't Fix / Can't Fix*: The Developer will not or cannot fix the defect due to some reason.
- *Can't Reproduce*: The Developer is unable to reproduce the defect.
- *Need More Information*: The Developer needs more information on the defect from the Tester.
- *REASSIGNED / REOPENED*: If the Tester finds that the 'fixed' defect is in fact not fixed or only partially fixed, it is reassigned to the Developer who 'fixed' it. A REASSIGNED defect needs to be COMPLETED again.
- *CLOSED / VERIFIED*: If the Tester / Test Lead finds that the defect is indeed fixed and is no more of any concern, it is CLOSED / VERIFIED. This is the happy ending.

Guidelines

- Make sure the entire team understands what each defect status exactly means. Also, make sure the defect life cycle is documented.
- Ensure that each individual clearly understands his/her responsibility as regards each defect.
- Ensure that enough detail is entered in each status change. For example, do not simply DROP a defect but provide a reason for doing so.

- If a defect tracking tool is being used, avoid entertaining any ‘defect related requests’ without an appropriate change in the status of the defect in the tool. Do not let anybody take shortcuts. Or else, you will never be able to get up-to-date and reliable defect metrics for analysis.

Verification and Validation

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. **Barry Boehm** described verification and validation as the following:

Verification: *Are we building the product right?*

Validation: *Are we building the right product?*

Verification:

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.

Verification is **Static Testing**.

Activities involved in verification:

1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

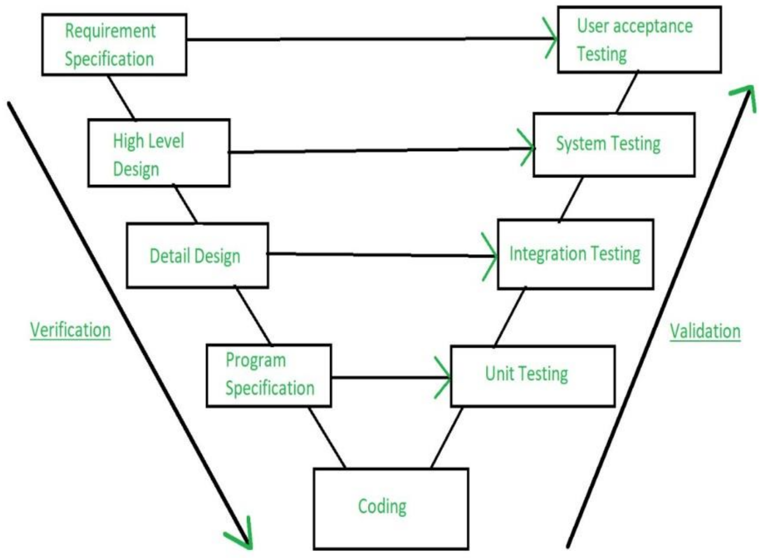
Validation:

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expectedproduct.

Validation is the **Dynamic Testing**.

Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing



Note: Verification is followed by Validation.

Software Testing Functional and Structural

Functional testing is sometimes called black-box testing because no knowledge of the internal logic of the system is used to develop test cases. For example, if a certain function key should produce a specific result when pressed, a functional test validates this expectation by pressing the function key and observing the result. When conducting functional tests, you'll use validation techniquesalmost exclusively.

Conversely, structural testing is sometimes called white-box testing because knowledge of the internal logic of the system is used to develop hypothetical test cases. Structural tests use verification predominantly. If a software development team creates a block of code that will allow a system to process information in a certain way, a test team would verify this structurally by reading the code, and given the system's structure, see if the code would work reasonably. If they felt it could, they would plug the code into the system and run an application to structurally validate the code.

Each method has its pros and cons:

Functional Testing Advantages

1• Simulates actual system usage.

2• Makes no system structure assumptions

Functional Testing Disadvantages

1• Potential of missing logical errors in software

2• Possibility of redundant testing

Structural Testing Advantages

1• You can test the software's structure logic

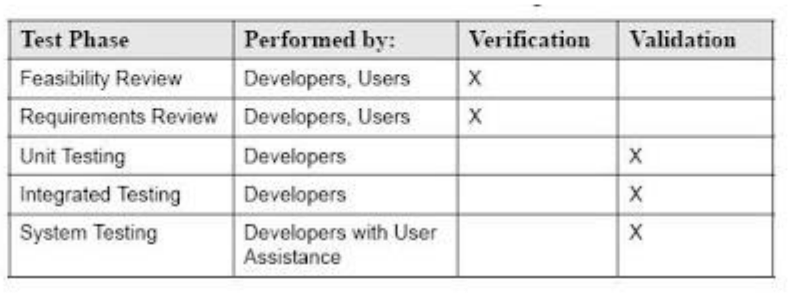
2• You test code that you wouldn't use if you performed only functional testing

Structural Testing Disadvantages

1• Does not ensure that you've met user requirements

2• Its tests may not mimic real-world situations

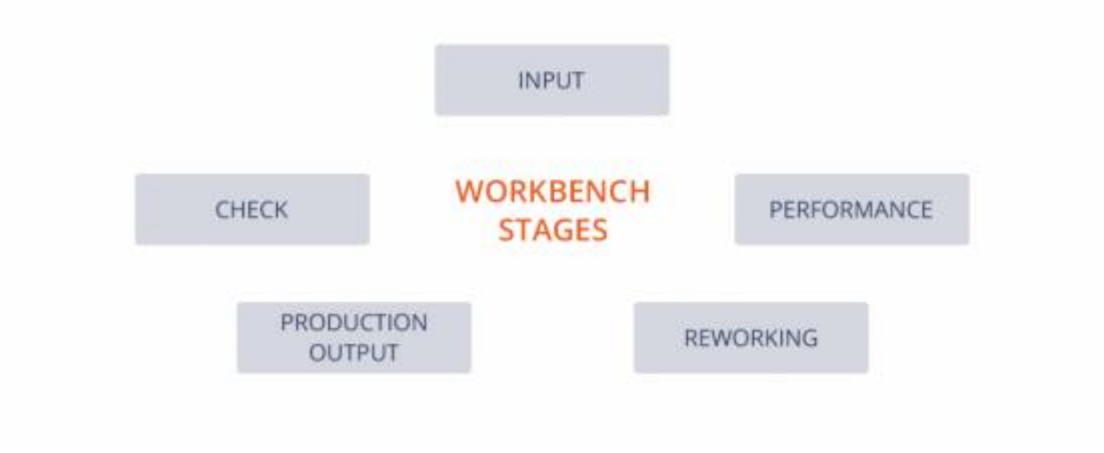
A functional test case might be taken from the documentation description of how to perform a certain function, such as accepting bar code input. A structural test case might be taken from a technical documentation manual. To effectively test systems, you need to use both methods.



Test Phase	Performed by:	Verification	Validation
Feasibility Review	Developers, Users	X	
Requirements Review	Developers, Users	X	
Unit Testing	Developers		X
Integrated Testing	Developers		X
System Testing	Developers with User Assistance		X

Workbench

This is a method which aims to examine and verify the structure of testing performance by detailed documenting. Workbench process has its common stages and steps which serve for different test assignments. The common stages of each workbench include:



Input. It is the initial workbench stage. Each certain assignment should contain its initial and outcome (input and output) requirements to know the available parameters and expected results. Each workbench has its specific inputs depending on the type of product under testing.

Performance. The priority aim of the entire testing is in the transformation of the initial parameters to outcome requirements and reach the prescribed results.

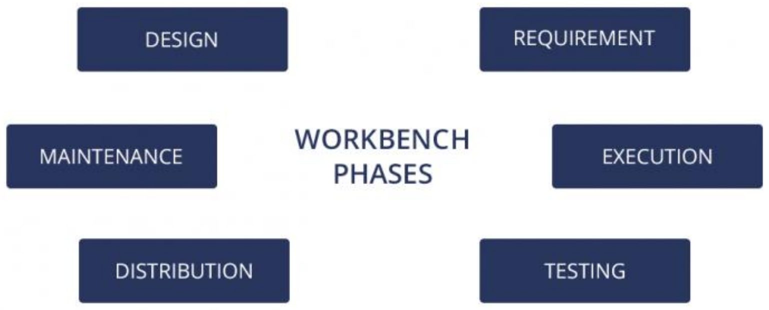
Check. It is an examination of output parameters after the performance phase to verify its accordance with the expected ones.

Production output. It is the final stage of a workbench in case the check confirmed the properly conducted performance.

Reworking. If the outcome parameters are not in compliance with the desired result, it is necessary to return to the performance phase and conduct it from the beginning.

Workbench Phases

Let's look at the initial and outcome data from different angles, considering various phases:



Requirement phase

- the initial data should be collected from the customer to perform a test task;
- the customer's requirements are included in the document to check its accordance with clients' needs;
- the outcome data are received and codified in one document.

Design phase

- the initial data is in the requirement document;
- testers prepare the technical document and check if the design document is technically proper;
- testers check if the information about the requirements is transferred to the requirement document completely.

Execution phase

- it is the performance of the entire testing process;
- the initial data is contained in the technical document, and the performance means adjusting of code according to the documented technical requirements;

- the outcome data is the source code.

Testing phase

The initial parameters are contained in the source code, and the outcomes are formed after the test performance.

Distribution phase

The aim of this phase is to prepare the product which is ready for use. Initial data, in this case, is a code version given by customer with initial requirements and code after testing.

Maintenance phase

- input appears as the outcome of distribution;
- the outcome data forms a new release;
- each change in product requirements is subjected to regression testing to fulfill the customers' requests.

The workbench concept serves to build and monitor the proper structure of testers' work. It helps to divide assignments in each phase of testing and reach the customers' expectations relying on initial data and transforming the product parameters into desirable ones.

The eight considerations listed below provide the framework for developingtesting tactics.

Each is described in the following sections.

•Acquire and study the test strategy

•Determine the type of development project

•Determine the type of software system

•Determine the project scope

•Identify the tactical risks

•Determine when testing should occur

•Build the tactical test plan

•Build the unit test plansAcquire and Study the Test Strategy

A team familiar with the business risks associated with the software normallydevelops the test strategy, and the test team develops the tactics. Thus, the testteam needs to acquire and study the test strategy, focusing on the followingquestions:

•What is the relationship of importance among the test factors?

•Which of the high-level risks are the most significant?

•Who has the best understanding of the impact of the identified businessrisks?

•What damage can be done to the business if the software fails to performcorrectly?

•What damage can be done to the business if the software is notcompleted on time?

Determine the Type of Development Project

The type of project refers to the environment in which the software willbedeveloped, and the methodology used. Changes to the environment also changethe testing risk. For example, the risks associated with a traditional developmenteffort are different from the risks associated with off-the shelf purchasedsoftware.

Testing checklist

During SDLC (Software Development Life Cycle) while software is in the testing phase, it is advised to make a list of all the required documents and tasks to avoid last minute hassle. This way tester will not miss any important step and will keep a check on quality too. If the tester doesn't make any checklist or forgets to include any task in it then it is possible that he may miss some of the important defects.

Testing Checklist is divided into number of categories which are listed as follows:

1) Resource Assignment and Training

- To make sure that testing project has sufficient budget allocated at project level.
- We have sufficient staffing or human resources allocated for testing project.
- Analyze skills and competencies of test team to make sure whether they are competent enough or required more grooming to meet required skill set.
- All required testing tools are installed at workstation with appropriate software licence.
- All resources are well trained on required testing tools and project business.
- Required responsibilities are assigned to team member and respective leads.
- All required sign off are procured from senior management for staffing and training.

2) Software Testing Documentation

- Make sure that all the functional documents and design documents are completed before testing team can start writing test cases.
- Test plan is created covering all the required test cases.
- Test cases are created covering all the required business use cases.
- Review of test cases and test plan following maker and checker policy.
- Setting up of Bug reporting portal to log the defects.
- Creation of tractability matrix with the functional team to make sure functions are mapped to test cases.
- Make sure, project weekly status report format is well defined.
- Sign off or approval from QA manager to execute the test cases.

3) Software Testing Checklist

- Regression suite is executed successfully when testing with new test phase or new project release.
- Make sure each tester is filling the time sheet and logging defect in defect portal on daily basis.
- Keeping a check on total test cases executed on daily basis and hence project work progress.
- Test weekly status reports is circulated on weekly basis with correct format and to required recipients.

- Open bugs are addressed timely by development team, requirement gathering team and senior management.
- Make sure, there are no roadblocks in testing area related to technologies, management and client behavior.
- Make sure before declaring the testing status as complete or providing testing sign off, all major or minor open bugs or defects are either closed or deferred for future release.
- Make sure all system compatibility checks are done, e.g. an application working on IE explorer should also work on chrome, Mozilla, etc.

4) Compliance's

- Review of test plan to make sure project complies with the required design methods.
- Evaluation of project goal statement with the business use cases.
- Project should comply with all required legal compliance's.
- Identify the priorities items in the project which are necessary for organization compliance's and execute those items.
- Examine project plan for strategic compliance with objective of business organization.
- Verify that project deliverable are in compliance with the client requirements.
- Evaluate project capabilities meets the desired outcome to accomplish predefined project goal.
- All necessary project compliance sign offs are procured from senior management.

5) Measurability and Monitoring

- Evaluation of project activities and processes that are well measurable to set the desired level of performance.
- Verification of System process measures for reliability and accuracy.
- Requesting the client to assign accessibility of project system to project team for review.
- Scheduling check point call, test phase completion call and daily scrum call to monitor the progress of test project.
- Make sure test project deliverable has predefined acceptance criteria which is approved, this will help to measure project deliverable.

- Make sure, proper escalation contact details are well communicated to client and project team members.

6) Project Flexibility

- Make sure that project is flexible and has ability to make desired amendments timely.
- Evaluate project has risk mitigation plan after analyzing all the possible project risk factors.
- Make sure project control system is reliable and effective.
- Make sure that project has a contingency plan to address exception and unforeseen events.
- Be confident that project goal completely address problems defined by the business use cases.
- Creation of self-regulatory feedback loop for the project to make sure every problem is reported related to test project work.

These are some of the main terms should be included in the **Testing Checklist**, however, every organization has different software and application *Testing Checklist* may vary. It is always a good practice to make a checklist so that testing can be done in a proper way and no important point should be missed.