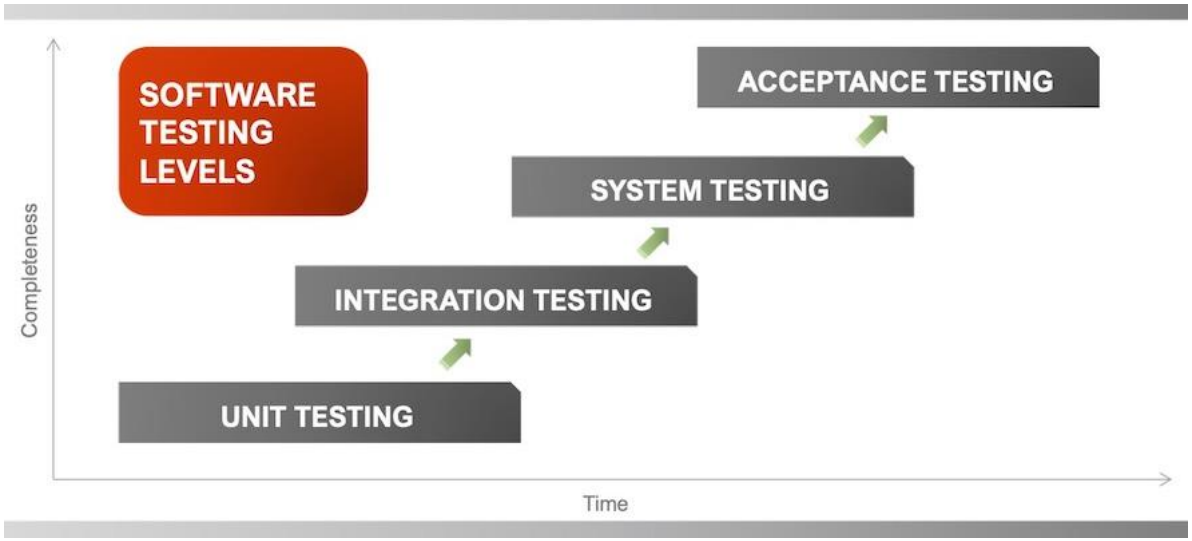
Unit- IV
Software Testing Types

- Dr. Shivani Budhkar

Levels of testing



Levels of testing

- **SOFTWARE TESTING LEVELS** are the different stages of the software development lifecycle where testing is conducted. There are four levels of software testing: Unit >> Integration >> System >> Acceptance.

Level	Summary
Unit Testing	A level of the software testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed.
Integration Testing	A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
System Testing	A level of the software testing process where a complete, integrated system is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
Acceptance Testing	A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

What is Unit Testing?

- **UNIT TESTING** is a type of software testing where individual units or components of a software are tested.
- The purpose is to validate that each unit of the software code performs as expected.
- Unit Testing is done during the development (coding phase) of an application by the developers.
- Unit Tests isolate a section of code and verify its correctness.
- A unit may be an individual function, method, procedure, module, or object.
- In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing.
- Unit testing is a White Box testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

Why Unit Testing?

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

The key reasons to perform unit testing :

- Unit tests help to fix bugs early in the development cycle and save costs.
- It helps the developers to understand the code base and enables them to make changes quickly
- Good unit tests serve as project documentation
- Unit tests help with code re-use. Migrate both your code **and**your tests to your new project. Tweak the code until the tests run again.

When & Who

- **When is it performed?**

Unit Testing is the first level of software testing and is performed prior to Integration Testing. Though unit testing is normally performed after coding, sometimes, specially in test-driven development (TDD), automated unit tests are written prior to coding.

- **Who performs it?**

It is normally performed by software developers themselves or their peers. In rare cases, it may also be performed by independent software testers but they will need to have access to the code and have an understanding of the architecture and design.

- **Test Driven Development (TDD) & Unit Testing**

  - Unit testing in TDD involves an extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to TDD, but they are essential to it. Below we look at some of what TDD brings to the world of unit testing:
  - Tests are written before the code
  - Rely heavily on testing frameworks
  - All classes in the applications are tested
  - Quick and easy integration is made possible

How to do Unit Testing

- In order **to do Unit Testing**, developers write a section of code to test a specific function in software application. Developers can also isolate this function to test more rigorously which reveals unnecessary dependencies between function being tested and other units so the dependencies can be eliminated. Developers generally use UnitTest framework to develop automated test cases for unit testing.
- Unit Testing is of two types
- Manual
- Automated
- Unit testing is commonly automated but may still be performed manually. Software Engineering does not favour one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document.

Unit Testing Techniques

- The **Unit Testing Techniques** are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behaviour of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.
- Code coverage techniques used in Unit Testing are listed below:
- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

Tasks in unit testing

- Unit testing planning
  - Describe unit test approach & risk
  - Identify unit features to be tested
  - Add levels of details to plan
- Designing test cases & test procedures(will be attached to test plan)
- Define Relationship between the tests
- Prepare Auxiliary code – Stub and drivers

Unit Testing Frameworks

- Unit Testing frameworks are mostly used to help write unit tests quickly and easily. Most of the programming languages do not support unit testing with the inbuilt compiler. Third-party open source and commercial tools can be used to make unit testing even more fun.
- **List of popular Unit Testing tools** for different programming languages:
- Java framework – JUnit
- PHP framework – PHPUnit
- C++ frameworks – UnitTest++ and Google C++
- .NET framework – NUnit
- Python framework – py.test

Benefits Of Unit Testing

- **The process becomes agile:** For adding new functions or features to the existing software we need to make changes to the old code. But changing things to the already tested code can be risky as well as costly.
- **Code quality improves:** The quality of code is automatically improved when unit testing is done. The bugs identified during this testing are fixed before it is sent for the integration testing phase. Result in robust design and development as developers write test cases by understanding the specifications first.
- **Detects bugs early:** As developers run unit tests, they detect bugs early in the software development life cycle and resolves them. This includes flaws or missing parts in the specification as well as bugs in the programmer's implementation.
- **Easier changes and simplified integrations:** Doing unit testing makes it easy for the developer to restructure the code, make changes, and maintain the code. It also makes testing the code after integration much easier. Fixing an issue in Unit Testing can fix many other issues occurring in later development and testing stages
- **Documentation availability:** Developers who are looking into the functionality at a later stage can refer to the unit testing documentation and can easily find the unit test interface and correct or work fast and easily.

Benefits Of Unit Testing

- **Easy debugging process:** It helps in simplifying the debugging process. If the test fails at any stage the code needs to be debugged or else the process can be continued without any obstacles.
- **Lower cost:** When bugs are detected and resolved during unit testing, cost and development time is reduced. Without this testing, if the same bugs are detected at a later stage after the code integration, it becomes more difficult to trace and resolve, making it more costly and increasing development time.
- **Code completeness can be demonstrated using unit tests:** This is more useful in the agile process. Testers don't get the functional builds to test until integration is completed. Code completion cannot be justified by showing that you have written and checked in the code. But running Unit tests can demonstrate code completeness.
- **Saves development time:** Code completion may take more time but due to fewer bugs in the System and Acceptance testing, overall development time can be saved.
- Code coverage can be measured

Integration testing

- **INTEGRATION TESTING** is a level of software testing where individual units / components are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.
- Integration testing is done to test the modules/components when integrated to verify that they work as expected i.e. to test the modules which are working fine individually does not have issues when integrated.
- A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated
- Integration Testing focuses on checking data communication amongst these modules.
- The main function or goal of this testing is to test the interfaces between the units/modules.
- We normally do Integration testing after “Unit testing”. Once all the individual units are created and tested, we start combining those “Unit Tested” modules and start doing the integrated testing.
- The individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behaviour, and validate whether the requirements are implemented correctly or not.
- Integration testing means testing of interfaces

Why do Integration Testing?

- Although each software module is unit tested, defects still exist for various reasons like -
  - A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary to verify the software modules work in unity
  - At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
  - Interfaces of the software modules with the database could be erroneous
  - External Hardware interfaces, if any, could be erroneous
  - Inadequate exception handling could cause issues.

Approaches, Strategies, Methodologies of Integration Testing

- Software Engineering defines variety of strategies to execute Integration testing, viz.
- Big Bang Approach :
- Incremental Approach: which is further divided into the following
  - Top Down Approach
  - Bottom Up Approach
  - Sandwich Approach - Combination of Top Down and Bottom Up

Big Bang Testing

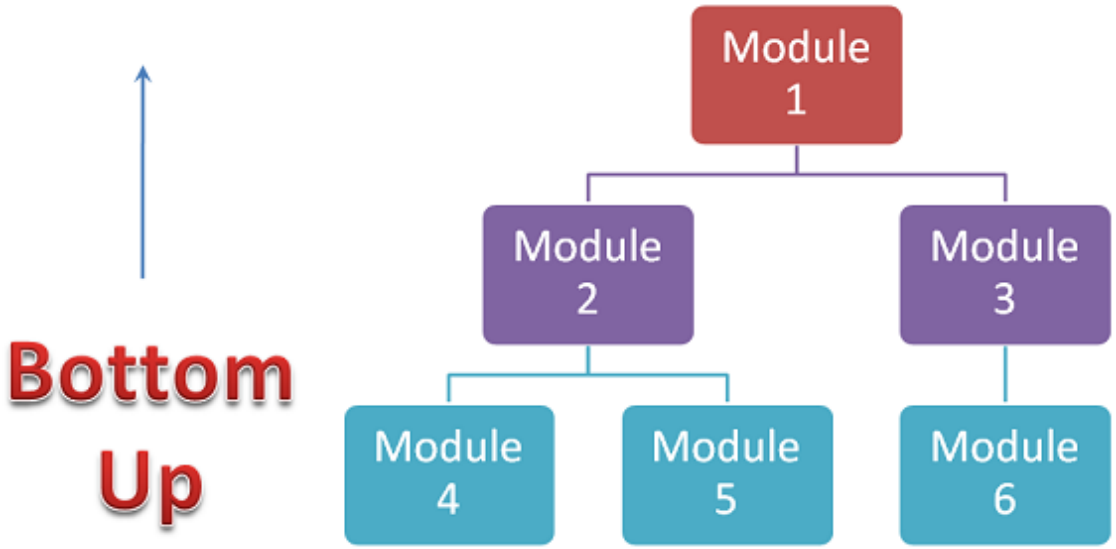
- **Big Bang Testing** is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit. This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.
- **Advantages:**
  - Convenient for small systems.
- **Disadvantages:**
  - Fault Localization is difficult.
  - Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
  - Since the Integration testing can commence only after "all" the modules are designed, the testing team will have less time for execution in the testing phase.
  - Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Testing

- In the **Incremental Testing** approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully.
- Incremental Approach, in turn, is carried out by two different Methods:
  - Bottom Up
  - Top Down
- **Stubs and Drivers**
- **Stubs and Drivers** are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as a substitutes for the missing models in the testing. They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.
- **Stub**: Is called by the Module under Test.
- **Driver**: Calls the Module to be tested.

Bottom-up Integration Testing

- **Bottom-up Integration Testing** is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

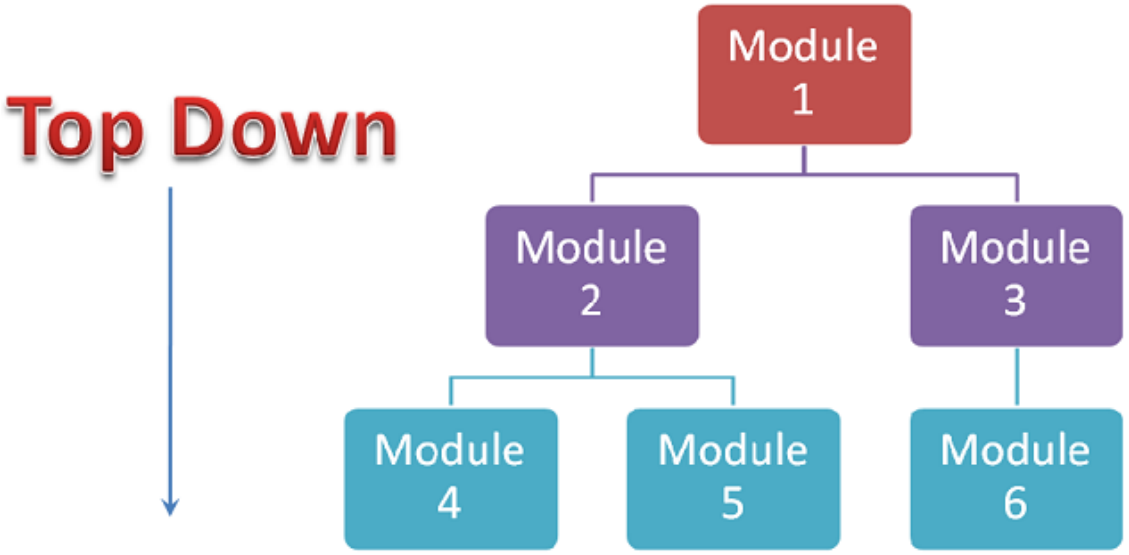


Bottom-up Integration Testing

- **Advantages:**
  - Fault localization is easier.
  - No time is wasted waiting for all modules to be developed unlike Big-bang approach
- **Disadvantages:**
  - Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
  - An early prototype is not possible

Top-down Integration Testing

- **Top Down Integration Testing** is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.

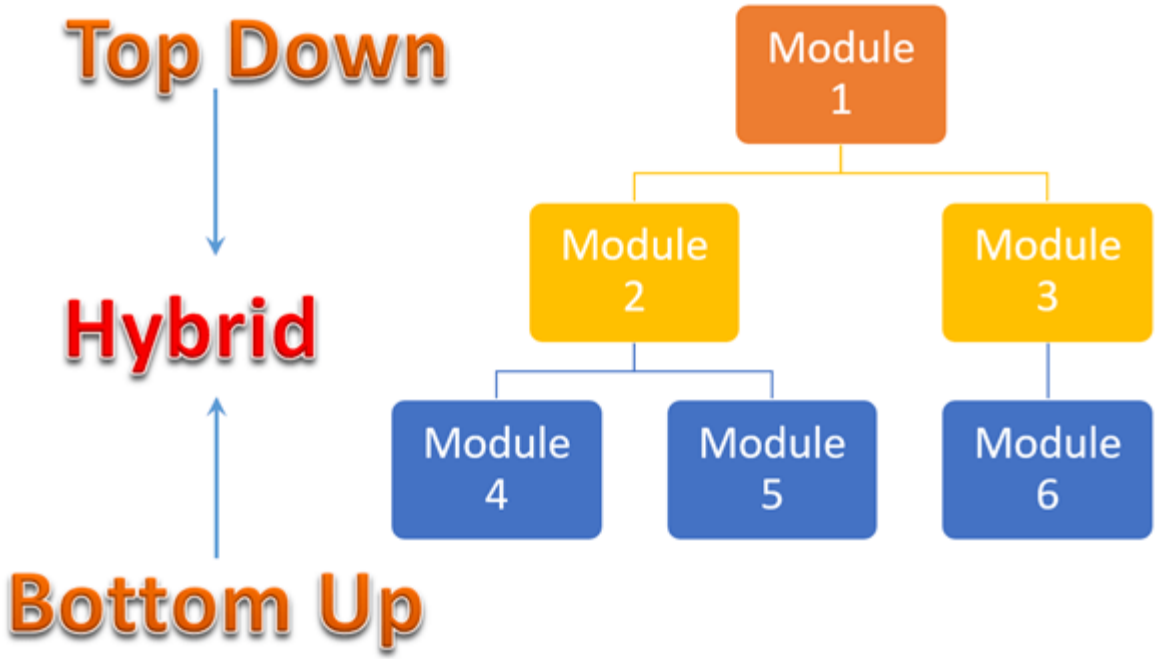


Top-down Integration Testing

- **Advantages:**
  - Fault Localization is easier.
  - Possibility to obtain an early prototype.
  - Critical Modules are tested on priority; major design flaws could be found and fixed first.
- **Disadvantages:**
  - Needs many Stubs.
  - Modules at a lower level are tested inadequately.

Sandwich Testing

- **Sandwich Testing** is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called **Hybrid Integration Testing**. It makes use of both stubs as well as drivers.



How to do Integration Testing?

- The Integration test procedure irrespective of the Software testing strategies
  - Prepare the Integration Tests Plan
  - Design the Test Scenarios, Cases, and Scripts.
  - Executing the test Cases followed by reporting the defects.
  - Tracking & re-testing the defects.
  - Steps 3 and 4 are repeated until the completion of Integration is successful.

Example of Integration Test Case

- Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.
- Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.
- Here the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.
- Similarly Mail Box: Check its integration to the Delete Mails Module.

Example of Integration Test Case

Test cases

Test Case ID	Test case Objective	Test case Description	Expected Result
ITC_001	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
ITC_002	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear in the Deleted/Trash folder

System Testing

- **SYSTEM TESTING** is a level of testing that validates the complete and fully integrated software product.
- The purpose of a system test is to evaluate the end-to-end system specifications.
- Usually, the software is only one element of a larger computer-based system.
- Ultimately, the software is interfaced with other software/hardware systems.
- System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.
- System test falls under the **black box testing** category of software testing.
- System test involves the external workings of the software from the user's perspective.
- System testing done by a professional testing agent on the completed software product before it is introduced to the market.
- System testing is only phase of testing which tests both functional and non functional aspects of products

What do you verify in System Testing?

System Testing involves testing the software code for following-

- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.
- You need to build detailed test cases and test suites that test each aspect of the application as seen from the outside without looking at the actual source code.

Non functional side of system testing

- **Some of the non functional testing types -**
- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behaviour of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.
- **Reliability testing:** To evaluate the ability of system to perform its required functions repeatedly for specific period of time is called reliability testing
- **Interoperability testing:** This testing is done to ensure that two or more products can exchange information, use information and work closely
- **Recoverability Testing:** To make sure how well the system recovers from various input errors and other failure situations.
- **Documentation Testing:** To make sure that the system's user guide and other help topics documents are correct and usable.
- **Security Testing:** To make sure that the system does not allow unauthorized access to data and resources.
- **Usability Testing:** To make sure that the system is easy to use, learn and operate.

Why System Testing?

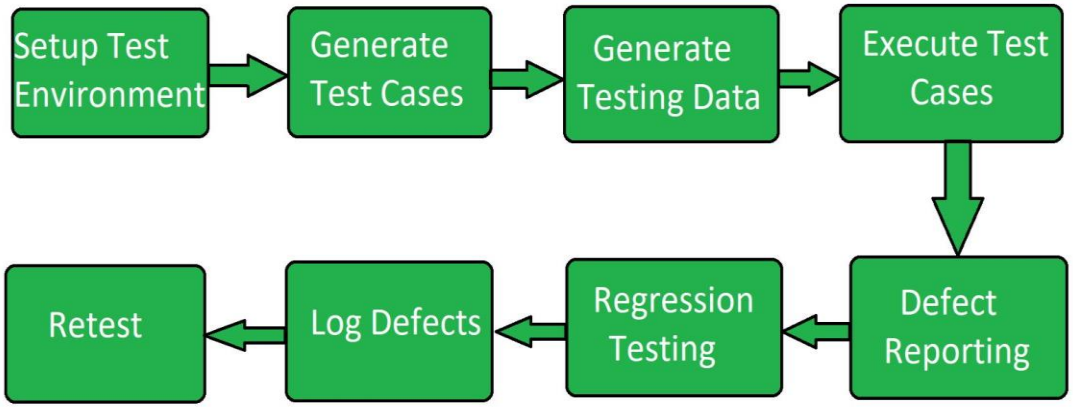
- It is very important to complete a full test cycle and ST is the stage where it is done.
- ST is performed in an environment that is similar to the production environment and hence stakeholders can get a good idea of the user's reaction.
- It helps to minimize after-deployment troubleshooting and support calls.
- In this STLC stage Application Architecture and Business requirements, both are tested.
- This testing is very important and it plays a significant role in delivering a quality product to the customer.
- The behaviour of complete product is verified during system testing
- System testing is highly complementary to other phases of testing

System Testing Process

System Testing is performed after the integration testing and before the acceptance testing.

- System Testing is performed in the following steps:
- **Test Environment Setup:**
  Create testing environment for the better quality testing.
- **Create Test Case:**
  Generate test case for the testing process.
- **Create Test Data:**
  Generate the data that is to be tested.
- **Execute Test Case:**
  After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:**
  Defects in the system are detected.
- **Regression Testing:**
  It is carried out to test the side effects of the testing process.
- **Log Defects:**
  Defects are fixed in this step.
- **Retest:**
  If the test is not successful then again test is performed.

System Testing Process



Entry/Exit Criteria

- **Entry Criteria:**
  - The system should have passed the exit criteria of Integration testing i.e. all the test cases should have been executed and there should be no critical or Priority P1, a P2 bug in an open state.
  - Test Plan for this testing should be approved & signed off.
  - Test cases/scenarios should be ready to be executed.
  - Test scripts should be ready to be executed.
  - All the non–functional requirements should be available and test cases for the same should have been created.
  - The testing environment should be ready.
- **Exit Criteria:**
  - All the test cases should be executed.
  - No critical or Priority or security-related bugs should be in an open state.
  - If any medium or low priority bugs are in an open state, then it should be implemented with the acceptance of the customer.
  - Exit Report should be submitted.

Functional Vs. 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.

Functional Vs. Non-Functional Testing

Parameters	Functional testing	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 behaviour 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	Examples of Non-functional Testing Types Performance Testing Volume Testing Scalability Usability Testing Load Testing Stress Testing Compliance Testing Portability Testing Disaster Recover Testing

Acceptance testing

- Once the System Testing process is completed by the testing team and is signed-off, the entire Product/application is handed over to the customer/few users of customers/both, to test for its acceptability i.e., Product/application should be flawless in meeting both the critical and major Business requirements. Also, end-to-end business flows are verified similar as in real-time scenario.
- The production-like environment will be the testing environment for Accepting Testing
- This is a black-box testing technique where only the functionality is verified to ensure that the product meets the specified acceptance criteria

Why Acceptance Tests?

- Though System testing has been completed successfully, the Acceptance test is demanded by the customer. Tests conducted here are repetitive, as they would have been covered in System testing.
- ***Then, why is this testing is conducted by customers?***
- **This is because:**
  - To gain confidence in the product that is getting released to the market.
  - To ensure that the product is working in the way it has to.
  - To ensure that the product matches current market standards and is competitive enough with the other similar products in the market.

Types of Acceptance testing

- There are several types of this testing.
- **Few of them are :**

1. User Acceptance Testing (UAT)

- UAT is to assess whether the Product is working for the user, correctly for the usage. Specific requirements which are quite often used by the end-users are primarily picked for the testing purpose. This is also termed as End-User Testing.
- The term “User” here signifies the end-users to whom the Product/application is intended and hence, testing is performed from the end-users perspective and from their point of view.

2. Business Acceptance Testing (BAT)

- This is to assess whether the Product meets the business goals and purposes or not.
- BAT mainly focuses on business benefits (finances) which are quite challenging due to the changing market conditions/advancing technologies so that the current implementation may have to undergo changes which result in extra budgets.
- Even the Product passing the technical requirements may fail BAT due to these reasons.

Types of Acceptance testing

3. Contract Acceptance Testing (CAT)

- This is a contract which specifies that once the Product goes live, within a predetermined period, the acceptance test must be performed and it should pass all the acceptance use cases.
- Contract signed here is termed as Service Level Agreement (SLA), which includes the terms where the payment will be made only if the Product services are in-line with all the requirements, which means the contract is fulfilled.
- Sometimes, this contract may happen before the Product goes live. Either the ways, a contract should be well defined in terms of the period of testing, areas of testing, conditions on issues encountered at later stages, payments, etc.

4. Regulations/Compliance Acceptance Testing (RAT)

- This is to assess whether the Product violates the rules and regulations that are defined by the government of the country where it is being released. This may be unintentional but will impact negatively on the business.
- Usually, the developed Product/application that is intended to be released all over the world, has to undergo RAT, as different countries/regions have different rules and regulations defined by its governing bodies.
- If any of the rules and regulations are violated for any country, then that country or the specific region in that country will not be allowed to use the Product and is considered as a Failure. Vendors of the Product will be directly responsible if the Product is released even though there is a violation.

Types of Acceptance testing

5. Operational Acceptance Testing (OAT)

- This is to assess the operational readiness of the Product and is a non-functional testing. It mainly includes testing of recovery, compatibility, maintainability, technical support availability, reliability, fail-over, localization etc.
- OAT mainly assures the stability of the Product before releasing it to the production.

6. Alpha Testing

- This is to assess the Product in the development/testing environment by a specialized testers team usually called alpha testers. Here, the testers feedback, suggestions help to improve the Product usage and also to fix certain bugs.
- Here, testing happens in a controlled manner.

7. Beta Testing/Field Testing

- This is to assess the Product by exposing it to the real end-users, usually called beta testers/beta users, in their environment. Continuous feedback from the users is collected and the issues are fixed. Also, this helps in enhancing/improving the Product to give a rich user experience.
- Testing happens in an uncontrolled manner, which means a user has no restrictions on the way in which the Product is being used.

All these types have a common goal:

- *Ensure to gain/enrich Confidence in the Product.*
- *Ensure that the Product is ready to be used by the real users.*

Who does Acceptance Testing?

- For Alpha type, only the members of the organization (who developed the Product) perform the testing. These members are not directly a part of the project (Project managers/leads, developers, testers). Management, Sales, Support teams usually perform the testing and provide feedback accordingly.

- Apart from the Alpha type, all other acceptance types are generally performed by different stakeholders. Like customers, customer's customers, specialized testers from the organization (not always).

- It is also good to involve Business Analysts and Subject Matter Expertise while performing this testing based on its type.

Acceptance Tests

- Similar to Product test cases, we do have acceptance tests. Acceptance tests are derived from User stories' acceptance criteria. These are usually the scenarios that are written at the high-level detailing on what the Product has to do under different conditions.

- It does not give a clear picture on how to perform tests, as in test cases. Acceptance tests are written by Testers who have a complete grip on the Product, usually Subject Matter Expertise. All the tests written are reviewed by a customer and/or business analysts.

- These tests executed during acceptance test. Along with acceptance tests, a detailed document on any set-ups to be done has to be prepared. It should include every minute details with proper screenshots, set-up values, conditions, etc.