Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

Why to Learn Software Testing?

In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

Applications of Software Testing

- **Cost Effective Development** - Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.
- **Product Improvement** - During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.
- **Test Automation** - Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.
- **Quality Check** - Software testing helps in determining following set of properties of any software such as
    - Functionality
    - Reliability
    - Usability
    - Efficiency
    - Maintainability
    - Portability

What is Testing?

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC −

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process −

- Testing Deadlines
- Completion of test case execution

- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision

TESTING OBJECTIVES:

Software Testing has different goals and objectives.The major objectives of Software testing are as follows:

1. Testing is a process of executing a program with the intent of finding an error.

2. A good test case is one that has a high probability of finding an as yet undiscovered error.

3. A successful test is one that uncovers an as yet undiscovered error.

Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications. The data collected through testing can also provide an indication of the software's reliability and quality. But, testing cannot show the absence of defect -- it can only show that software defects are present.

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it’s performing well and as per the specifications.

While determining the test coverage the test cases should be designed well with maximum possibilities of finding the errors or bugs. The test cases should be very effective. This objective can be measured by the number of defects reported per test cases. Higher the number of the defects reported the more effective are the test cases.

Once the delivery is made to the end users or the customers they should be able to operate it without any complaints. In order to make this happen the tester should know as how the customers are going to use this product and accordingly they should write down the test scenarios and design the test cases. This will help a lot in fulfilling all the customer’s requirements.

Software testing makes sure that the testing is being done properly and hence the system is ready for use. Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues. It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use.

TESTING PRINCIPLES:

Before applying methods to design effective test cases, a software engineer must understand the basic principles that guide software testing.

1. All tests should be traceable to customer requirements. As we have seen, the objective of software testing is to uncover errors. It follows that the most severe defects (from the customer's point of view) are those that cause the program to fail to meet its requirements.
2. Tests should be planned long before testing begins. Test planning can begin as soon as the requirements model is complete.Detailed definition of test cases can begin as soon as the design model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.
3. The Pareto principle applies to software testing. Stated simply, the Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.
4. Testing should begin "in the small" and progress toward testing "in the large." The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system
5. Exhaustive testing is not possible. The number of path permutations for even a moderately sized program is exceptionally large . For this reason, it is impossible to execute every combi-ination of paths during testing. It is possible, however, to dequately cover program logic and to ensure that all conditions in the component-level design have been exercised. To be most effective, testing should be conducted by an independent third party. By most effective, we mean testing that has the highest probability of finding errors (the primary objective of testing).

Testability – What is its role in Software Testing?

Testing is a critical stage of the software development lifecycle. The aim is to release bug-free, performant software that won't cost you a fortune in backend running costs. Clearly, making this process more efficient and effective will save you time and effort, and in the long run, will improve your profitability. This is one of the main drivers behind the switch to test automation. However, one important factor is often overlooked – software testability.

Testability, a property applying to empirical hypothesis, involves two components. The effort and effectiveness of software tests depends on numerous factors including:

- Properties of the software requirements
- Properties of the software itself (such as size, complexity and testability)
- Properties of the test methods used
- Properties of the development- and testing processes
- Qualification and motivation of the persons involved in the test process

Testability of software components

The testability of software components (modules, classes) is determined by factors such as:

- Controllability: The degree to which it is possible to control the state of the component under test (CUT) as required for testing.
- Observability: The degree to which it is possible to observe (intermediate and final) test results.
- Isolateability: The degree to which the component under test (CUT) can be tested in isolation.

- Separation of concerns: The degree to which the component under test has a single, well defined responsibility.
- Understandability: The degree to which the component under test is documented or self-explaining.
- Automatability: The degree to which it is possible to automate testing of the component under test.
- Heterogeneity: The degree to which the use of diverse technologies requires to use diverse test methods and tools in parallel.

The testability of software components can be improved by:

- Test-driven development
- Design for testability (similar to design for test in the hardware domain)

Testability hierarchy

Based on the amount of test cases required to construct a complete test suite in each context (i.e. a test suite such that, if it is applied to the implementation under test, then we collect enough information to precisely determine whether the system is correct or incorrect according to some specification), a testability hierarchy with the following testability classes has been proposed:

- Class I: there exists a finite complete test suite.
- Class II: any partial distinguishing rate (i.e. any incomplete capability to distinguish correct systems from incorrect systems) can be reached with a finite test suite.
- Class III: there exists a countable complete test suite.
- Class IV: there exists a complete test suite.
- Class V: all cases.

It has been proved that each class is strictly included into the next. For instance, testing when we assume that the behavior of the implementation under test can be denoted by a deterministic finite-state machine for some known finite sets of inputs and outputs and with some known number of states belongs to Class I (and all subsequent classes). However, if the number of states is not known, then it only belongs to all classes from Class II on. If the implementation under test must be a deterministic finite-state machine failing the specification for a single trace (and its continuations), and its number of states is unknown, then it only belongs to classes from Class III on. Testing temporal machines where transitions are triggered if inputs are produced within some real-bounded interval only belongs to classes from Class IV on, whereas testing many non-deterministic systems only belongs to Class V (but not all, and some even belong to Class I). The inclusion into Class I does not require the simplicity of the assumed computation model, as some testing cases involving implementations written in any programming language, and testing implementations defined as machines depending on continuous magnitudes, have been proved to be in Class I. Other elaborated cases, such as the testing framework by Matthew Hennessy under must semantics, and temporal machines with rational timeouts, belong to Class II.

Testability of requirements

Requirements need to fulfill the following criteria in order to be testable:

- consistent
- complete
- unambiguous
- quantitative (a requirement like "fast response time" can not be verification/verified)
- verification/verifiable in practice (a test is feasible not only in theory but also in practice with limited resources)

Testability in simple terms can be defined as the extent of ease with which a system can be tested. The ISO defines testability as "attributes of software that bear on the effort needed to validate the software product."

Any project you take or own, testability of the software that is being created or amended is a decisive factor in the time and cost of testing. At times the management decides to drop a few parts or phases of testing which ultimately results in a low quality product and also the product is not able to stand in competition in the market.

Like we all know that testing has no end rather it is a never ending task. It is something that needs to be up-to-date with the user's preferences; it is a part of the life span of any software. And a wholesome truth is that software keeps altering, considering its upgradation to meet the demands of the customers and to stand in the competitive market. The new dimensions of software testing can also serve as a bridge for an effective testability.

In this purview, dedication towards testability of a system gives return over its entire life span. Testability is valuable and is a quality characteristic of a software system, along with all-time classics like functionality, security, and performance.

Software Testability Measurement

Software testability measurement refers to the activities and methods that study, analyze, and measure software testability during a software product life cycle. Once software is implemented, it is essential to make an assessment to finalise which software components are likely to be more difficult and time-consuming in testing due to their poor component testability. If such a measure could be applied at the beginning of a software testing phase, much more effective testing resources allocation and prioritizing could be possible. The objective of software testing is to make sure that the given software product meets the mentioned requirements by validating the function and nonfunctional requirements to uncover as many program defects as possible during a software test process. Unlike software testing, the major objective of software testability measurement is to find out which software components are poor in quality, and where faults can hide from software testing.

Testability is essential because:

- Sooner is better
- Higher testability – Better results, same cost
- Lower testability – Fewer weak results, same cost

There are numerous measures that can be taken to enhance testability as listed below:

- Transparency: Know in an out about the system, what it does and what it is supposed to do. Have a crystal clear understanding.
- Updated system documentation
- Comprehensive well-structured code
- Clarity in software requirements
- Isolation: Try techniques and mention codes by which parts of the system can be tested in isolation. Document each concern separately and make the results visible.
- Go for a test environment simulation: Strike a balance between representativeness and flexibility in favour of the system and give in suggestions wherein you can opt for automated tests.

The aforesaid measures may not be in the hit list of any manager or product owner but certainly they enhance the offerings of both the developers and the testers hence more efficient results. This helps in creating a better quality product and proves cost effective too. Testability establishes the boundary to which the jeopardy of costly or hazardous bugs can be abridged to an acceptable level.

Verification & Validation

These two terms are very confusing for most people, who use them interchangeably. The following table highlights the differences between verification and validation.

Sr.No.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behavior.
3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.	It has dynamic activities, as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify a software.	It is a subjective process and involves subjective decisions on how well a software works.

There are different methods that can be used for software testing. This chapter briefly describes the methods available.

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
Clearly separates user's perspective from the developer's perspective through visibly defined roles.	Blind coverage, since the tester cannot target specific code segments or errorprone areas.
Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	The test cases are difficult to design.

White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
It helps in optimizing the code.	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.

Extra lines of code can be removed which can bring in hidden defects.	It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.
Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	**Disadvantages**
Offers combined benefits of black-box and white-box testing wherever possible.	Since the access to source code is not available, the ability to go over the code and test coverage is limited.
Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.	The tests can be redundant if the software designer has already run a test case.
Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.	Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.
The test is done from the point of view of the user and not the designer.	

A Comparison of Testing Methods

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

What is System Testing?

This is **black box type of testing** where external working of the software is evaluated with the help of requirement documents & it is totally based on Users point of view. For this type of testing do not required knowledge of internal design or structure or code.

System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective.

System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased. It includes both functional and Non-Functional testing.

The process of performing a variety of tests on a system to explore functionality or to identify problems. System testing is usually required before and after a system is put in place. A series of systematic procedures are referred to while testing is being performed. These procedures tell the tester how the system should perform and where common mistakes may be found. Testers usually try to "break the system" by entering data that may cause the system to malfunction or return incorrect information. For example, a tester may put in a city in a search engine designed to only accept states, to see how the system will respond to the incorrect input.

System testing is the type of testing to check the behaviour of a complete and fully integrated software product based on the **software requirements specification (SRS) document**. The main focus of this testing is to evaluate Business / Functional / End-user requirements.

This testing is to be carried out only after System Integration Testing is completed where both Functional & Non-Functional requirements are verified.

In the **integration testing** testers are concentrated on finding bugs/defects on integrated modules. But in the Software System Testing testers are concentrated on finding bugs/defects based on software application behavior, software design and expectation of end user.

Why system testing is important:

a) In Software Development Life Cycle the System Testing is perform as the first level of testing where the System is tested as a whole.

b) In this step of testing check if system meets functional requirement or not.

c) System Testing enables you to test, validate and verify both the Application Architecture and Business requirements.

d) The application/System is tested in an environment that particularly resembles the effective production environment where the application/software will be lastly deployed.

Generally, a separate and dedicated team is responsible for system testing.

Different Hierarchical levels of testing:

As with almost any technical process, software testing has a prescribed order in which things should be done. Different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system. The following is lists of software testing categories arranged in sequentially organize.

Unit testing – Testing is done in the development process while developer completes the unit development. The object of this testing is to verify correctness of the module. The purpose of unit testing is to check that as individual parts are functioning as expected. Basically Unit testing is typically carried out by the developer.

- A unit is the smallest testable part of an application like functions, classes, procedures, interfaces. Unit testing is a method by which individual units of source code are tested to determine if they are fit for use.
- **Unit tests are basically written and executed by software developers** to make sure that code meets its design and requirements and behaves as expected.
- The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly.
- This means that for any function or procedure when a set of inputs are given then it should return the proper values. It should handle the failures gracefully during the course of execution when any invalid input is given.
- A unit test provides a written contract that the piece of code must assure. Hence it has several benefits.
- Unit testing is basically done before integration as shown in the image below.

BENEFITS

- Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
- Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.
- The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or when the software is live.
- Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/ weeks/ months need to be scanned.
- Codes are more reliable. Why? I think there is no need to explain this to a sane person.

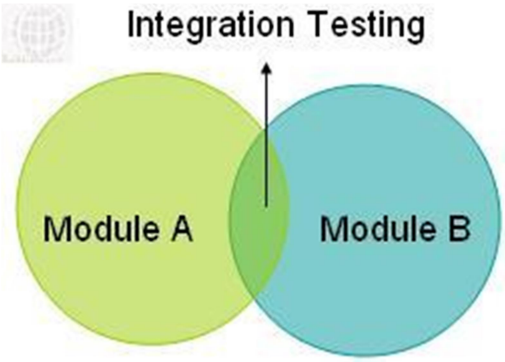
TIPS

- Find a tool/ framework for your language.
- Do not create test cases for everything. Instead, focus on the tests that impact the behavior of the system.
- Isolate the development environment from the test environment.
- Use test data that is close to that of production.

- Before fixing a defect, write a test that exposes the defect. Why? First, you will later be able to catch the defect if you do not fix it properly. Second, your test suite is now more comprehensive. Third, you will most probably be too lazy to write the test after you have already fixed the defect.
- Write test cases that are independent of each other. For example if a class depends on a database, do not write a case that interacts with the database to test the class. Instead, create an abstract interface around that database connection and implement that interface with mock object.
- Aim at covering all paths through the unit. Pay particular attention to loop conditions.
- Make sure you are using a version control system to keep track of your test scripts.
- In addition to writing cases to verify the behavior, write cases to ensure performance of the code.
- Perform unit tests continuously and frequently.

Integration testing – System Integration Testing is started after the individual software modules are integrated as a group. A typical software project consists of multiple modules & these are developed by different developers. So in integration testing is focuses to check that after integrating modules is two modules are communicating with each other or not. It is critical to test every module's effect on the entire program model. Most of the issues are observed in this type of testing.

- Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.
- Also after integrating two different components together we do the integration testing. As displayed in the image below when two different modules 'Module A' and 'Module B' are integrated then the integration testing is done.



- Integration testing is done by a specific integration tester or test team.

In Integration Testing, individual software modules are integrated logically and tested as a group.

A typical software project consists of multiple software modules, coded by different programmers. Integration testing focuses on checking data communication amongst these modules.

Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing'.

Need of 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 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.

Please be patient. The Video will load in some time. If you still face issue viewing video click here

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', 'Mail box' and 'Delete mails' and each of them are integrated logically.

Here do not concentrate much on 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.

Test Case ID

Test Case Objective

Test Case Description

Expected Result

1. 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

2. Check the interface link between the Mailbox and Delete Mails Module

From Mail box select the an email and click delete button

Selected email should appear in the Deleted/Trash folder

Approaches/Methodologies/Strategies of Integration Testing:

The Software Industry uses variety of strategies to execute Integration testing , viz.

Big Bang Approach :

Incremental Approach: which is further divided into following

Top Down Approach

Bottom Up Approach

Sandwich Approach - Combination of Top Down and Bottom Up

Big Bang Approach:

Here all component are integrated together at once, and then tested.

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 links to be tested could be missed easily.

Since the integration testing can commence only after "all" the modules are designed, 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 Approach:

In this approach, testing is done by joining two or more modules that are logically related. Then the other related modules are added and tested for the proper functioning. Process continues until all of the modules are joined and tested successfully.

This process is carried out by using dummy programs called Stubs and Drivers. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

Incremental Approach in turn is carried out by two different Methods:

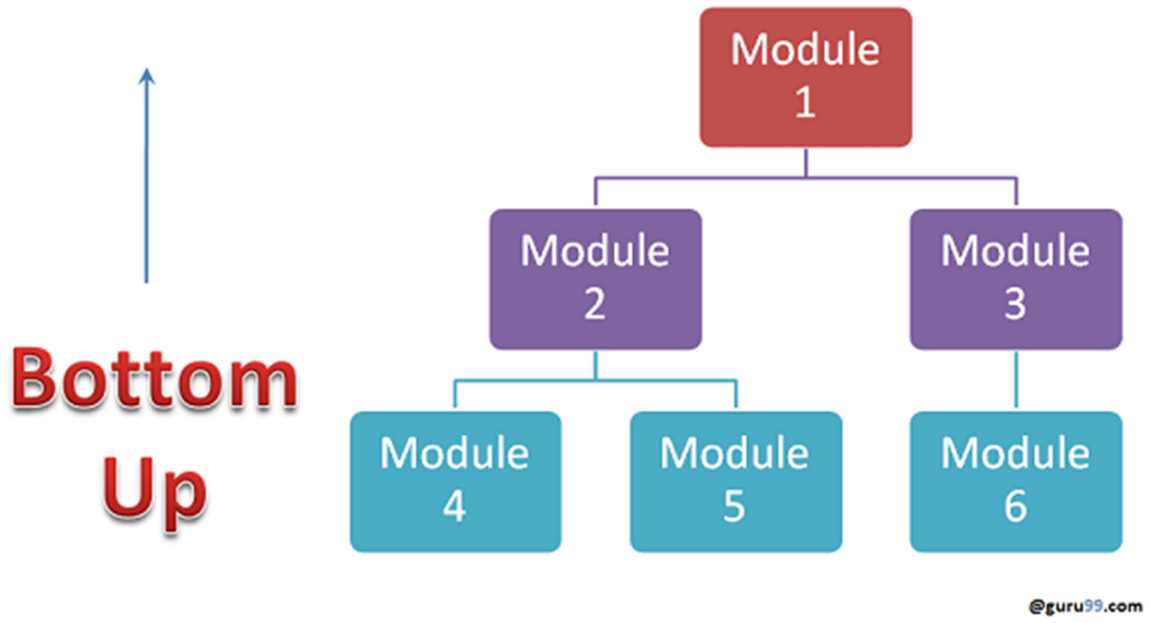
Bottom Up

Top Down

Bottom up Integration

In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

Diagrammatic Representation:



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.

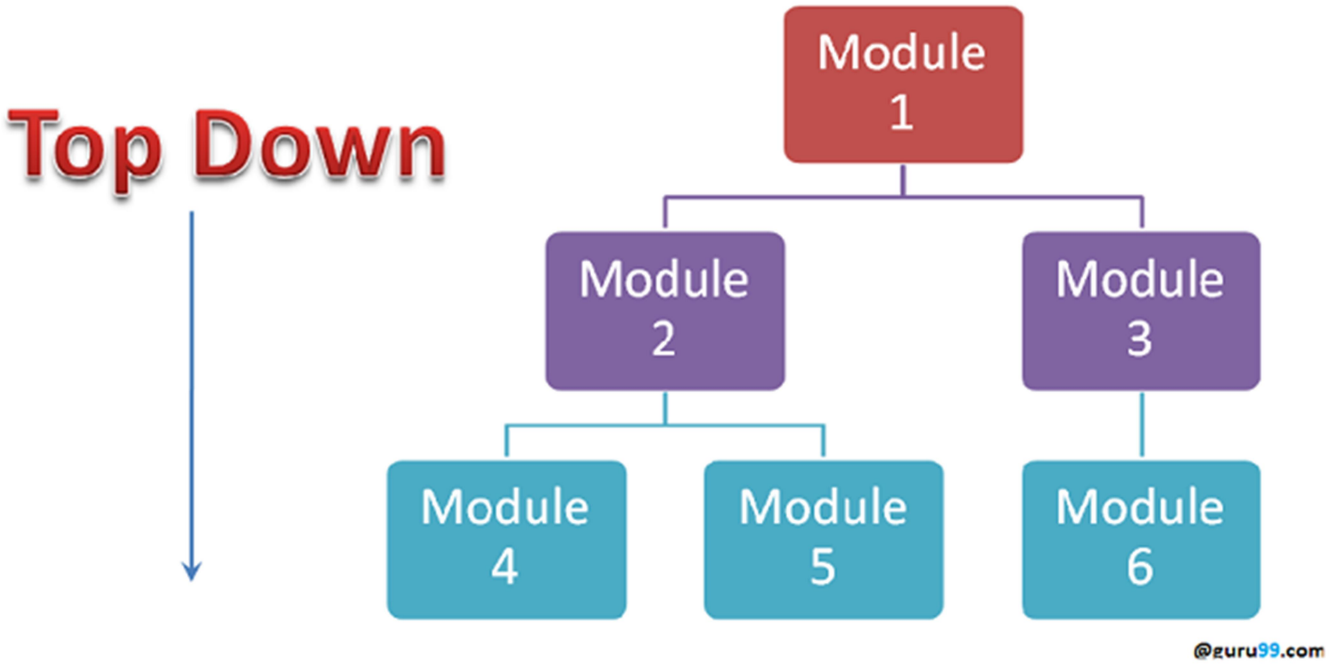
Early prototype is not possible

Top down Integration:

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Takes help of stubs for testing.

Diagrammatic Representation:



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 lower level are tested inadequately.

Integration Testing Procedure

The integration test procedure irrespective of the test strategies (discussed above):

Prepare the Integration Test 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 successfully.

Brief Description of Integration Test Plans:

It includes following attributes:

Methods/Approaches to test (as discussed above).

Scopes and Out of Scopes Items of Integration Testing.

Roles and Responsibilities.

Pre-requisites for Integration testing.

Testing environment.

Risk and Mitigation Plans.

Entry and Exit Criteria.

Entry and Exit Criteria to Integration testing phase in any software development model

Entry Criteria:

Unit Tested Components/Modules

All High prioritized bugs fixed and closed

All Modules to be code completed and integrated successfully.

Integration test Plan, test case, scenarios to be signed off and documented.

Required Test Environment to be set up for Integration testing

Exit Criteria:

Successful Testing of Integrated Application.

Executed Test Cases are documented

All High prioritized bugs fixed and closed

Technical documents to be submitted followed by release Notes.

Best Practices/ Guidelines for Integration Testing

First determine the Integration Test Strategy that could be adopted and later prepare the test cases and test data accordingly.

Study the Architecture design of the Application and identify the Critical Modules. These need to be tested on priority.

Obtain the interface designs from the Architectural team and create test cases to verify all of the interfaces in detail. Interface to database/external hardware/software application must be tested in detail.

After the test cases, it's the test data which plays the critical role.

Always have the mock data prepared, prior to executing. Do not select test data while executing the test cases.

Software Testing - Validation Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Software validation is the process of testing a software to check whether it satisfies the customer needs or not.

Verification and validation testing are two important tests which are carried out on a system before it has been handed over to the customer. The aim of both verification and validation is to ensure that the product is made according to the requirements of the client, and does indeed fulfill the intended purpose. While verification is a quality control process, the quality assurance process carried out before the software is ready for release is known as validation testing. Its goal is to validate and be confident about the product or system, and that it fulfills the requirements given by the customer. The acceptance of the software from the end customer is also its part. Often, testing activities are introduced early in the software development life cycle.

Validation testing provides answers to questions such as:
Does the software fulfill its intended use?
Is the company building the right product?
Can the project be properly implemented?
Are the documents in line with the development process?

Validation Testing Process

The aim of software testing is to measure the quality of a system in terms of number of defects found in it, the number of tests run, and the system covered by the tests. When bugs or defects are found with the help of testing, the bugs are logged and the development team fixes them. Once the bugs are fixed, testing is carried out again to ensure that they are indeed fixed, and no new defects have been introduced in the software. With the entire cycle, the quality of the software increases. The software validation process can be described as follows:

Validation Planning

To plan all the activities that need to be included while testing.

⇓

Define Requirements

To set goals and define the requirements for testing.

⇓

Select Appropriate Team

To select a skilled and knowledgeable development team (third party included).

⇓

Develop Documents

To develop a user specifications document describing the operating conditions.

⇓

Evaluation

To evaluate the software as per the specifications and submit a validation report.

⇓

Incorporating Changes

To change the software so as to remove any errors found during evaluation.

Techniques

There are various validation techniques like fault injection, dependability analysis, etc., that are commonly used for software testing. Here are a few more...

Formal Methods

This technique makes use of mathematical and logical techniques to analyze the input specifications, the product design, the supporting documents, and the behavior of the product under test.

Fault Injection

As the name suggests, faults or bugs are intentionally added to software so as to check its working/functionality under the said conditions. These are of two types - hardware fault injection and software fault injection. In the former, faults in physical hardware are injected, while in the latter, errors are injected in the software or in the system's memory.

Dependability Analysis

This technique is used to increase the dependability of the product. The hazards in the software are identified, and methods are suggested to reduce theses potential hazards.

Hazard Analysis

Every software has its own standard tests to identify hazards. This type of analysis follows the said procedure to identify the hazard and find out its countermeasures.

Risk Analysis:-This takes the hazard analysis further by identifying countermeasures for each type of hazard that is identified.

The Basic Tests:-Two of the most common and widely used terms in the software testing world are black-box testing and white-box testing.

Black-box Testing

This type of testing only focuses on the output that is generated after being subjected to varying inputs. The internal components of the software are completely ignored while carrying out this test. This test is also known as functional testing.

White-box Testing

As opposed to black-box testing, white-box testing takes into account the internal mechanism and components of the software while testing under the desired input conditions.

Types of Validation Testing

If the testers are involved in the product right from the very beginning, then the testing starts right after a component of the system has been developed. Apart from the basic tests, the different types of software validation tests are:

Component/Unit Testing

Component testing is also known as unit testing. The aim of the tests carried out in this testing type is to search for defects in the software component. At the same time, it also verifies the functioning of the different software components, like modules, objects, classes, etc., which can be tested separately.

Integration Testing

This is an important part of the software validation model, where the interaction between the different interfaces of the components is tested. Along with the interaction between the different parts of the system, the interaction of the system with the computer operating system, file system, hardware, and any other software system it might interact with, is also tested.

System Testing

System testing is carried out when the entire software system is ready. The concern of this testing is to check the behavior of the whole system as defined by the scope of the project. The main concern of system testing is to verify the system against the specified requirements. While carrying out the tests, the tester is not concerned with the internals of the system, but checks if the system behaves as per expectations.

Acceptance Testing

Here, the tester has to literally think like the client and test the software with respect to user needs, requirements, and business processes, and determine whether the software can be handed over to the client or not. At this stage, often, a client representative is also a part of the testing team, so that the client has confidence in the system.

Operational Acceptance Testing

This type of testing is done just before the software goes into the production/launching stage. It checks the

readiness of the product, by testing for backups, recovery techniques, shutdown and resumption, component failure, etc. It is also known as operational readiness testing. This method also takes into account the alerts that are raised during component failure or in an error situation.

Alpha Testing

If the testing is done for a large group of users, it is essential that each component of the system be tested for its functionality. This type of testing is done at the developers' site by potential customers/users. Any problems encountered during this testing are rectified by the developers then and there. This is internal acceptance testing.

Beta Testing

Once the software passes the alpha testing stage, beta testing is done at the user's end. Various versions of the software are developed, known as the beta versions. These versions are tested for their functionality by the users, the logs of the problems occurred are recorded and submitted to the developers. This is external acceptance testing.

Regression Testing

This testing is done after the desired changes or modifications are made to the existing code. The code, when put to test, may have certain errors that can be resolved by making necessary changes. The software is again put to test after these changes are made, to check whether the new code fulfills customer requirements or not.

Difference between Verification and Validation

In few days back we have seen article about "V-Model". In the V Model Software Development Life Cycle, based on requirement specification document the development & testing activity is started. The V-model is also called as Verification and Validation model. The testing activity is perform in the each phase of Software Testing Life Cycle. In the first half of the model validations testing activity is integrated in each phase like review user requirements, System Design document & in the next half the Verification testing activity is come in picture.

In interviews most of the interviewers are asking questions on "What is Difference between Verification and Validation?" Lots of people use verification and validation interchangeably but both have different meanings. So in this article I am adding few differences about Verification & Validation.

Verification	Validation
Are we building the system right?	Are we building the right system?
Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	**Validation** is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.
The objective of Verification is to make sure that the product being develop is as per the	The objective of Validation is to make sure that the product actually meet up the user's

requirements and design specifications.	requirements, and check whether the specifications were correct in the first place.
Following activities are involved in **Verification**: Reviews, Meetings and Inspections.	Following activities are involved in **Validation**: Testing like black box testing, white box testing, gray box testing etc.
Verification is carried out by QA team to check whether implementation software is as per specification document or not.	**Validation** is carried out by testing team.
Execution of code is not comes under **Verification**.	Execution of code is comes under **Validation**.
Verification process explains whether the outputs are according to inputs or not.	**Validation** process describes whether the software is accepted by the user or not.
Verification is carried out before the Validation.	**Validation** activity is carried out just after the Verification.
Following items are evaluated during **Verification**: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during **Validation**: Actual product or Software under test.
Cost of errors caught in**Verification** is less than errors found in Validation.	Cost of errors caught in**Validation** is more than errors found in Verification.
It is basically manually checking the of documents and files like requirement specifications etc.	It is basically checking of developed program based on the requirement specifications documents & files.

System testing – This is the first time end to end testing of application on the complete and fully integrated software product before it is launch to the market.

Acceptance testing – User acceptance is a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This is beta testing of the product & evaluated by the actual end users. The main purpose of this testing is to validate the end to end business flow.

Entry Criteria for System Testing:

Unit Testing should be finished.

Integration of modules should be fully integrated.

As per the specification document software development is completed.

Testing environment is available for testing (similar to Staging environment)

How to do System Testing?

In Software System Testing following steps needs to be executed:

Step 1) First & important step is preparation of System Test Plan:

The what all points to be cover in System Test plan may vary from organization to organization as well as based on project plan, test strategy & main test plan.

Nevertheless, here is list of standard point to be considered while creating System Test Plan:

Goals & Objective

Scope

Critical areas Area to focus

Test Deliverable

Testing Strategy for System testing

Testing Schedule

Entry and exit criteria

Suspension & resumption criteria for system testing

Test Environment

Roles and Responsibilities

Glossary

Step 2) Second step is to creation Test Cases:

It is very much similar functional test case writing. In test case writing you should write the test scenarios & use cases.

Here you should consider different type of testing like Functional testing, Regression testing, Smoke testing, Sanity testing, Ad-hoc testing, Exploratory testing, Usability testing, GUI software testing, Compatibility testing, Performance testing, Load testing, Stress testing, Volume testing, Error handling testing, Scalability testing, Security testing, Capacity testing, Installation testing, Recovery testing, Reliability testing, Accessibility testing etc

While writing test case you need to check that test cases are covering all functional, non-functional, technical & UI requirements or not.

Sample Test Case Format:

Test Case ID

Test Suite Name

How to Test? Test Data Expected Result Actual Result

Pass/Fail

Step 3) Creation of test data which used for System testing.

Step 4) Automated test case execution.

Step 5) Execution of normal test case & update test case if using any test management tool (if any).

Step 6) Bug Reporting, Bug verification & Regression testing.

Step 7) Repeat testing life cycle (if required).

In System testing now you have clear about What is System Testing?, Importance of system testing, different levels of testing and How & what criteria needs to be consider in System Testing. If you think any points is missed or to be added in System testing then you

If I missed out addressing some important points on System testing then let me know in comments below.