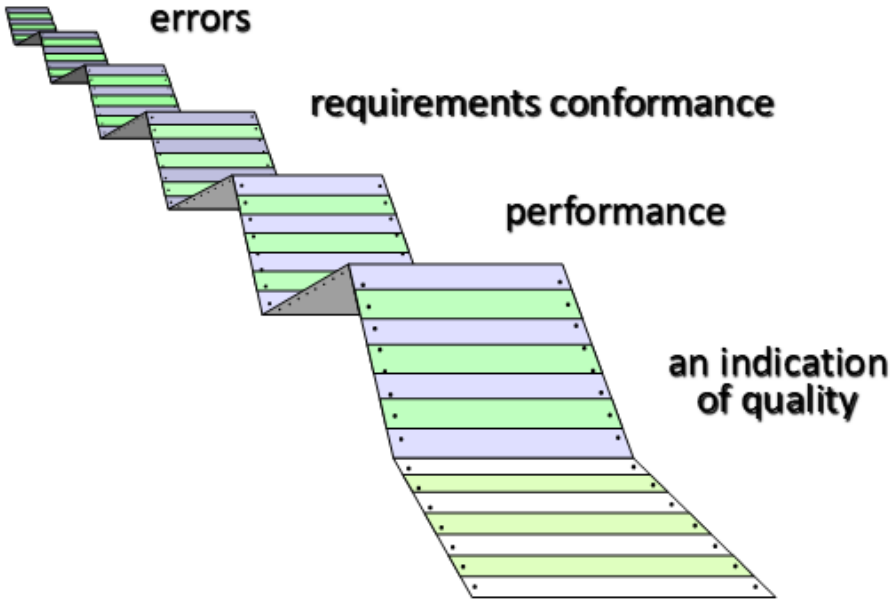
What is Testing?

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.

What Testing Show?



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

Software Testing - Myths

Given below are some of the most common myths about software testing.

Myth 1: Testing is Too Expensive

Reality − There is a saying, pay less for testing during software development or pay more for maintenance or correction later. 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.

Myth 2: Testing is Time-Consuming

Reality − 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.

Myth 3: Only Fully Developed Products are Tested

Reality − No doubt, testing depends on the source code but reviewing requirements and developing test cases is independent from the developed code. However iterative or incremental approach as a development life cycle model may reduce the dependency of testing on the fully developed software.

Myth 4: Complete Testing is Possible

Reality − It becomes an issue when a client or tester thinks that complete testing is possible. It is possible that all paths have been tested by the team but occurrence of

complete testing is never possible. There might be some scenarios that are never executed by the test team or the client during the software development life cycle and may be executed once the project has been deployed.

Myth 5: A Tested Software is Bug-Free

Reality − This is a very common myth that the clients, project managers, and the management team believes in. No one can claim with absolute certainty that a software application is 100% bug-free even if a tester with superb testing skills has tested the application.

Myth 6: Missed Defects are due to Testers

Reality − It is not a correct approach to blame testers for bugs that remain in the application even after testing has been performed. This myth relates to Time, Cost, and Requirements changing Constraints. However the test strategy may also result in bugs being missed by the testing team.

Myth 7: Testers are Responsible for Quality of Product

Reality − It is a very common misinterpretation that only testers or the testing team should be responsible for product quality. Testers' responsibilities include the identification of bugs to the stakeholders and then it is their decision whether they will fix the bug or release the software. Releasing the software at the time puts more pressure on the testers, as they will be blamed for any error.

Myth 8: Test Automation should be used wherever possible to Reduce Time

Reality − Yes, it is true that 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.

Myth 9: Anyone can Test a Software Application

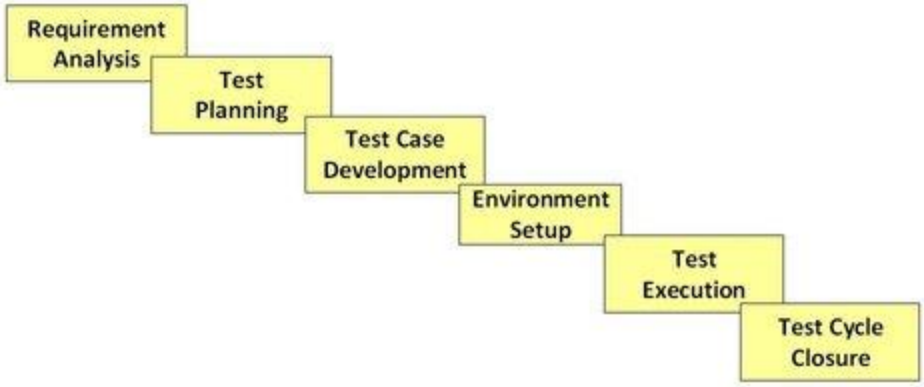
Reality − People outside the IT industry think and even believe that anyone can test a software and testing is not a creative job. However testers know very well that this is a myth. Thinking alternative scenarios, try to crash a software with the intent to explore potential bugs is not possible for the person who developed it.

Myth 10: A Tester's only Task is to Find Bugs

Reality − Finding bugs in a software is the task of the testers, but at the same time, they are domain experts of the particular software. Developers are only responsible for the specific component or area that is assigned to them but testers understand the overall workings of the software, what the dependencies are, and the impacts of one module on another module.

What is Software Testing Life Cycle (STLC)?

SOFTWARE TESTING LIFE CYCLE (STLC) is a sequence of specific activities conducted during the testing process to ensure software quality goals are met.



Requirement Analysis

During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.

Test Planning

Typically, in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan. In this phase, Test Strategy is also determined.

Test Case Development

This phase involves the creation, verification and rework of test cases & test scripts. Test data, is identified/created and is reviewed and then reworked as well.

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of

testing process and ***can be done in parallel with Test Case Development Stage***.

Test Execution

During this phase, the testers will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

Test Cycle Closure

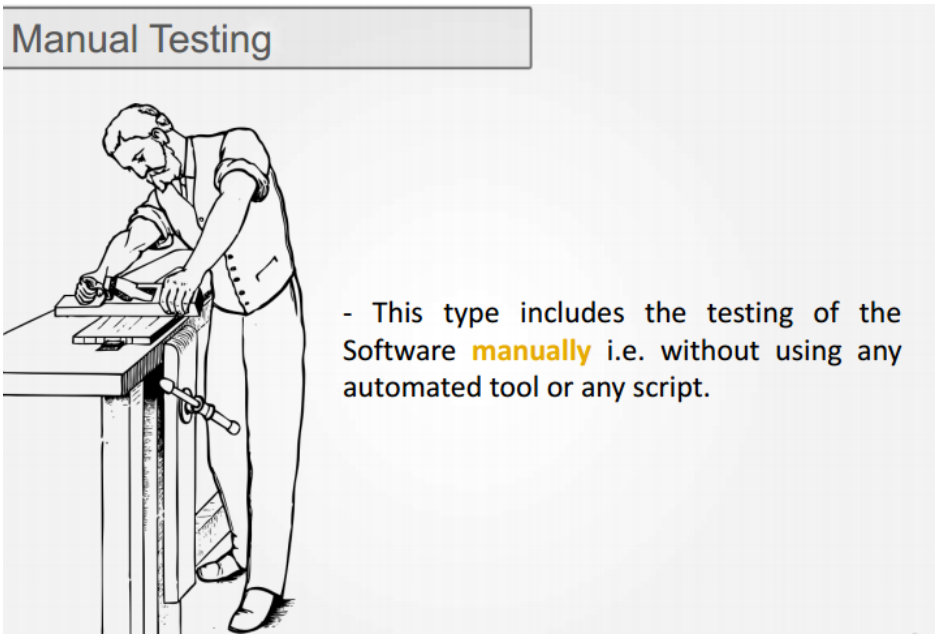
Testing team will meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in the future.

Software Testing - Types of Testing

This section describes the different types of testing that may be used to test a software during SDLC.

Manual Testing

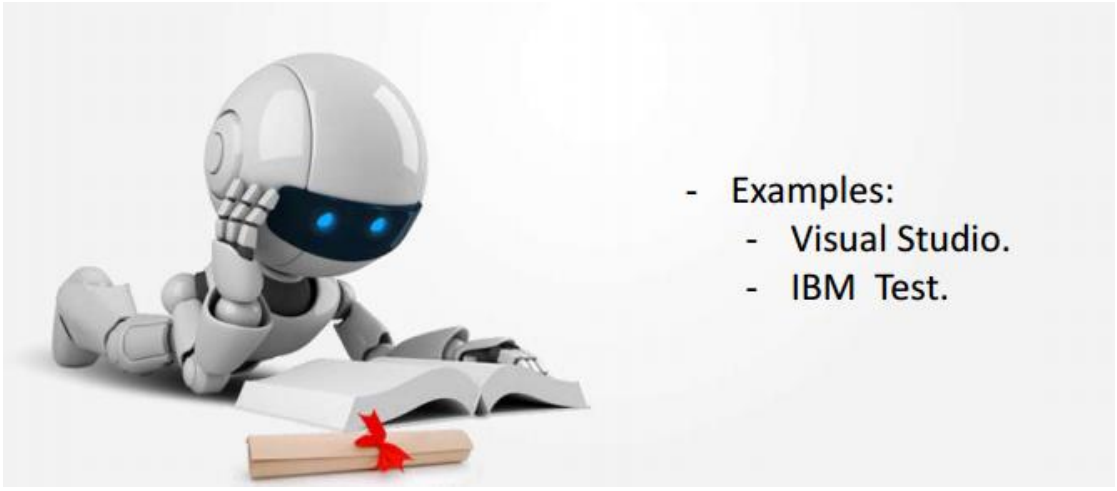
Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.



Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

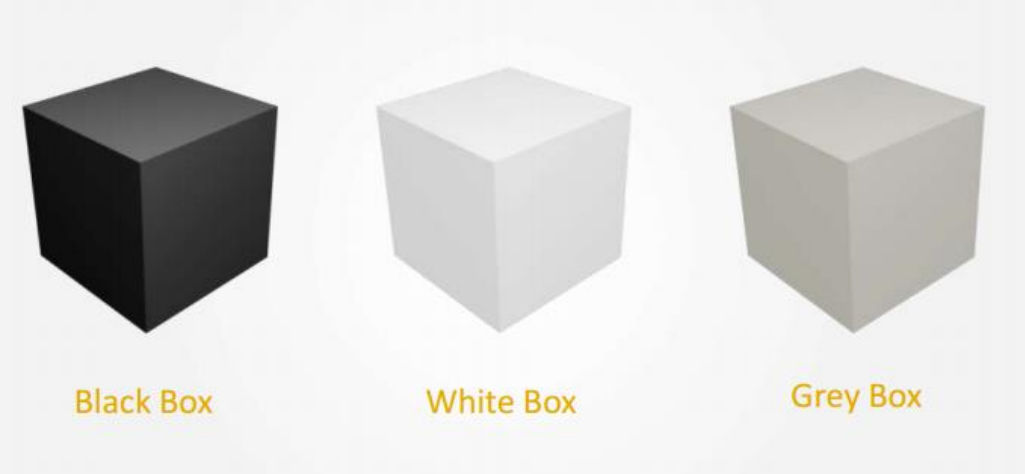
Automation Testing

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.



Software Testing - Methods

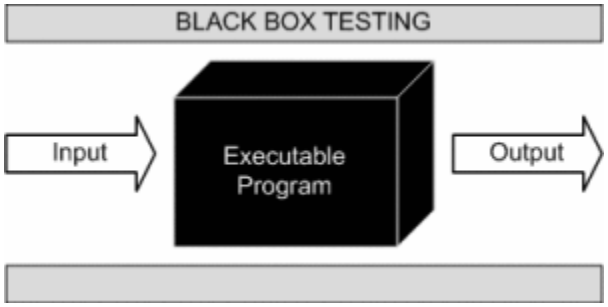
There are different methods that can be used for software testing. This chapter briefly describes the methods available.



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.

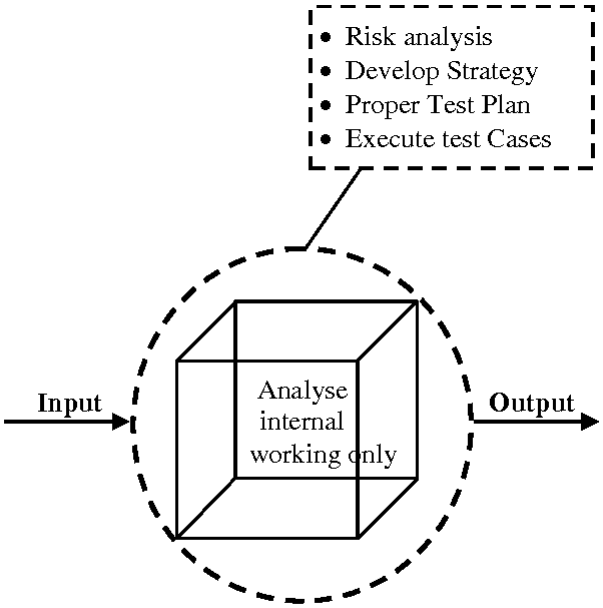


Figure 1. Represent white box testing

White box testing is a test case design method that uses the

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.	

Software Testing - Levels

There are different levels during the process of testing. In this chapter, a brief description is provided about these levels.

Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are −

- Functional Testing
- Non-functional Testing

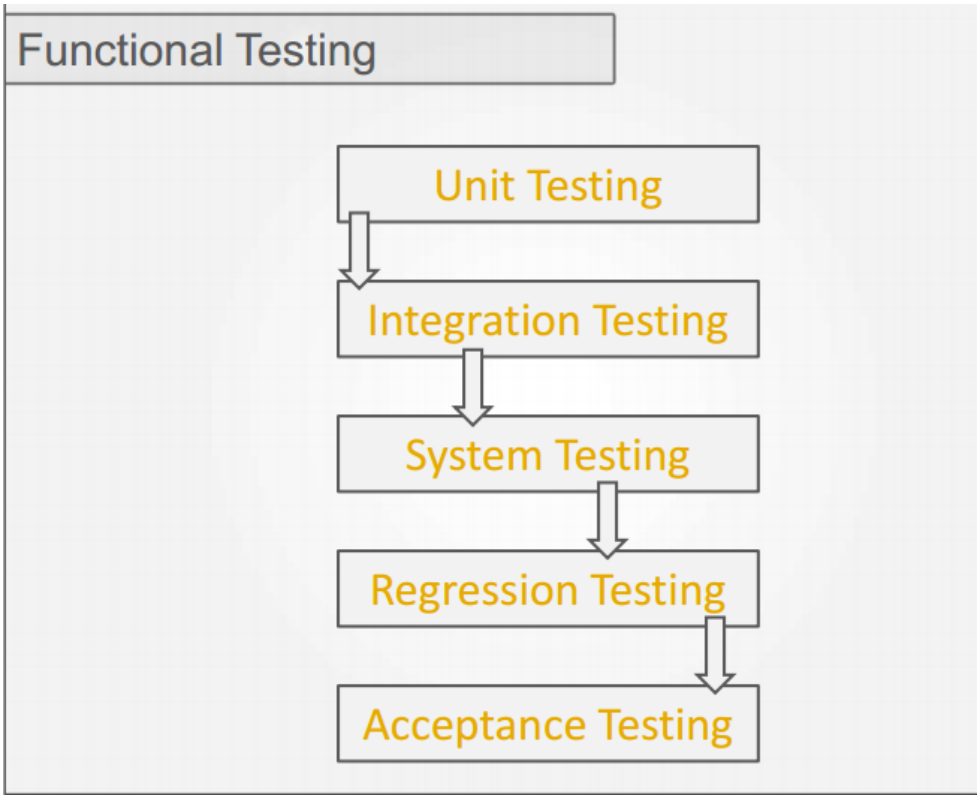
Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved while testing an application for functionality.

Steps	Description
I	The determination of the functionality that the intended application is meant to perform.
II	The creation of test data based on the specifications of the application.
III	The output based on the test data and the specifications of the application.
IV	The writing of test scenarios and the execution of test cases.
V	The comparison of actual and expected results based on the executed test cases.

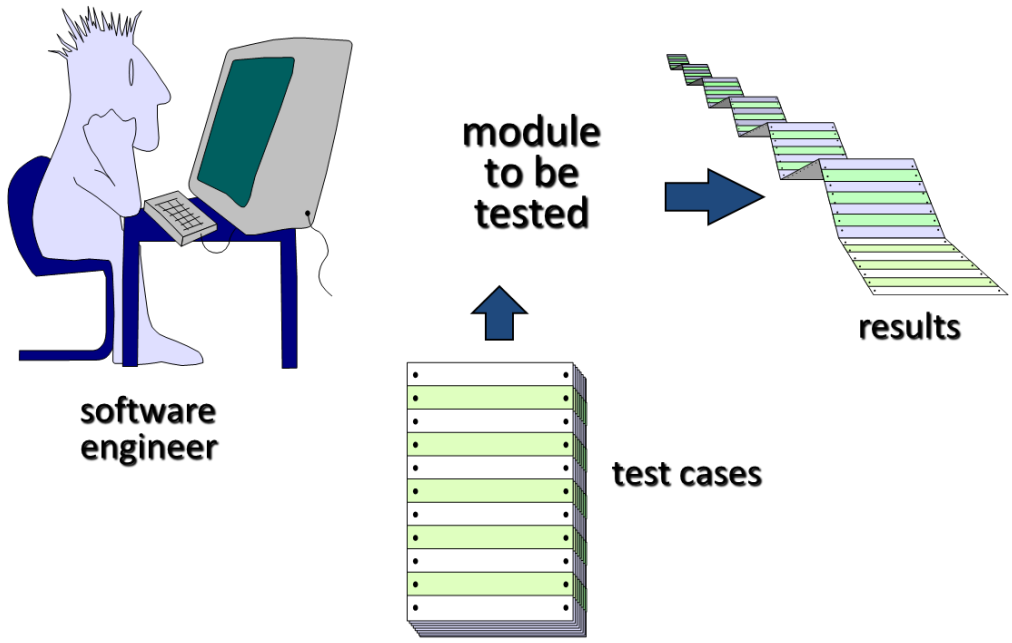
An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.



Unit Testing

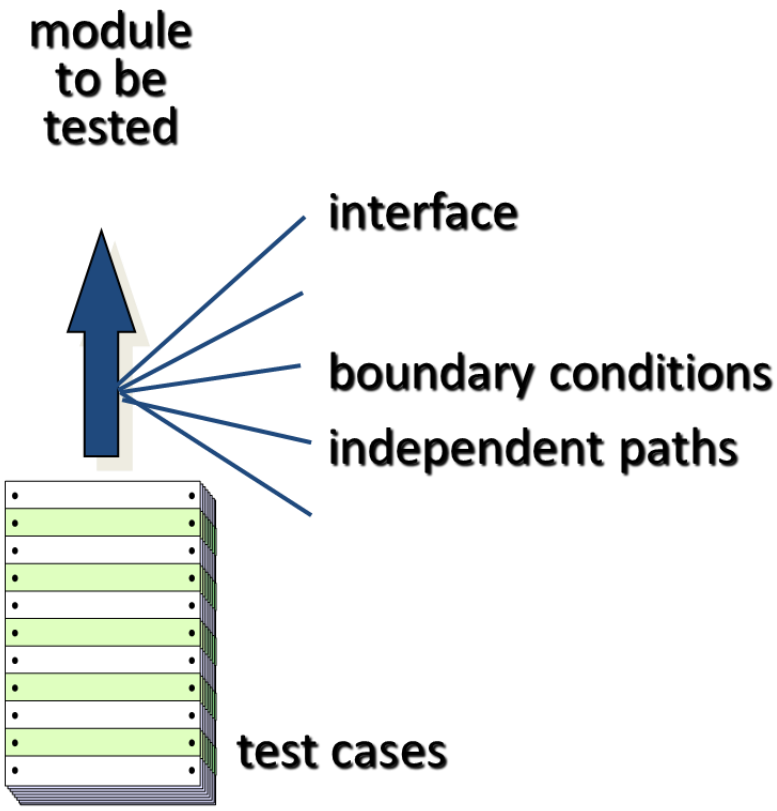
This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.



Targets for Unit Test

- Module interface
  - Ensure that information flows properly into and out of the module
- Local data structures
  - Ensure that data stored temporarily maintains its integrity during all steps in an algorithm execution
- Boundary conditions
  - Ensure that the module operates properly at boundary values established to limit or restrict processing
- Independent paths (basis paths)
  - Paths are exercised to ensure that all statements in a module have been executed at least once
- Error handling paths
  - Ensure that the algorithms respond correctly to specific error conditions



Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Sr.No.	Integration Testing Method
1	**Bottom-up integration**

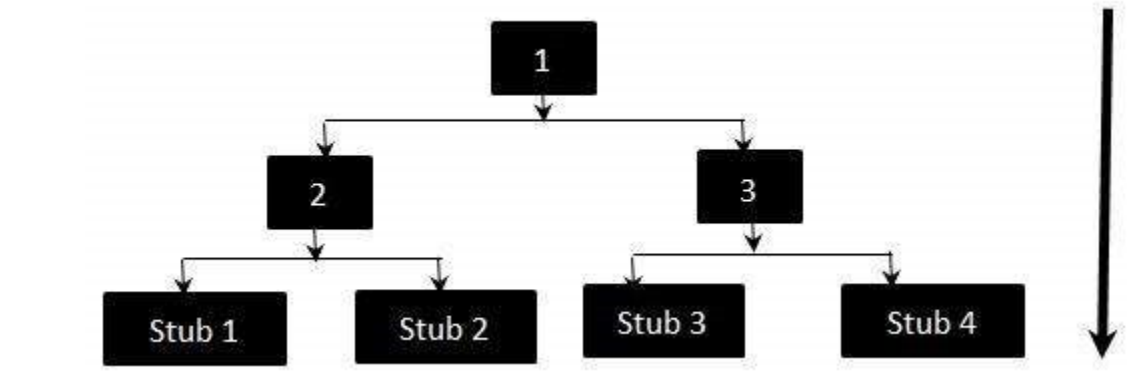
	This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	**Top-down integration** In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

Top Down Integration Testing

Top Down is an approach to Integration Testing where top-level units are tested first and lower level units are tested step by step after that. This approach is taken when top-down development approach is followed. Test Stubs are needed to simulate lower level units which may not be available during the initial phases.

Stub - Flow Diagram:



The above diagrams clearly states that Modules 1, 2 and 3 are available for integration, whereas, below modules are still under development that cannot be integrated at this point of time. Hence, Stubs are used to test the modules. The order of Integration will be:

```
1,2
1,3
2,Stub 1
2,Stub 2
3,Stub 3
3,Stub 4
```

Testing Approach:

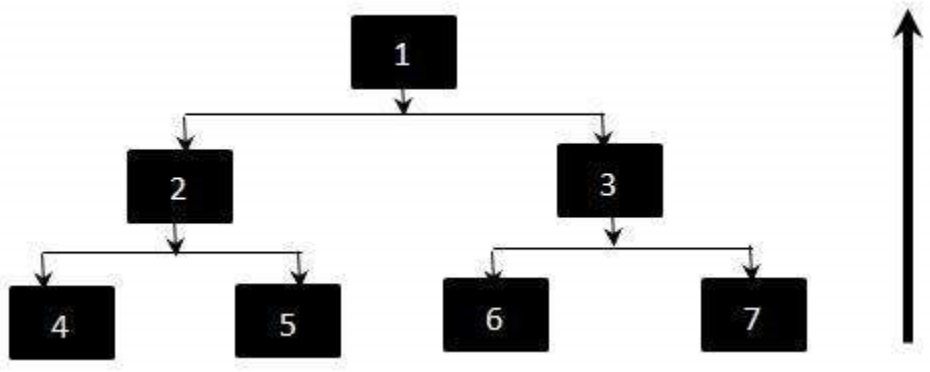
```
+ Firstly, the integration between the modules 1,2 and 3
+ Test the integration between the module 2 and stub 1,stub 2
+ Test the integration between the module 3 and stub 3,stub 4
```

- Advantages
  - This approach verifies major control or decision points early in the test process
- Disadvantages
  - Stubs need to be created to substitute for modules that have not been built or tested yet; this code is later discarded
  - Because stubs are used to replace lower level modules, no significant data flow can occur until much later in the integration/testing process

Bottom Up Integration Testing

Bottom Up is an approach to Integration Testing where bottom level units are tested first and upper-level units step by step after that. This approach is taken when bottom-up development approach is followed. Test Drivers are needed to simulate higher level units which may not be available during the initial phases.

Bottom Up Integration - Flow Diagram



The order of Integration by Bottom-down approach will be:

```
4,2
5,2
6,3
7,3
2,1
3,1
```

Testing Approach:

```
+ Firstly, Test 4,5,6,7 individually using drivers.
+ Test 2 such that it calls 4 and 5 separately. If an error occurs
we know that the problem is in one of the modules.
+ Test 1 such that it calls 3 and If an error occurs we know that
the problem is in 3 or in the interface between 1 and 3
```

Though Top level components are the most important, yet tested last using this strategy. In Bottom-up approach, the Components 2 and 3 are replaced by drivers while testing components 4,5,6,7. They are generally more complex than stubs.

- Advantages
  - This approach verifies low-level data processing early in the testing process
  - Need for stubs is eliminated
- Disadvantages
  - Driver modules need to be built to test the lower-level modules; this code is later discarded or expanded into a full-featured version
  - Drivers inherently do not contain the complete algorithms that will eventually use the services of the lower-level modules; consequently, testing may be incomplete or more testing may be needed later when the upper level modules are available

System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons −

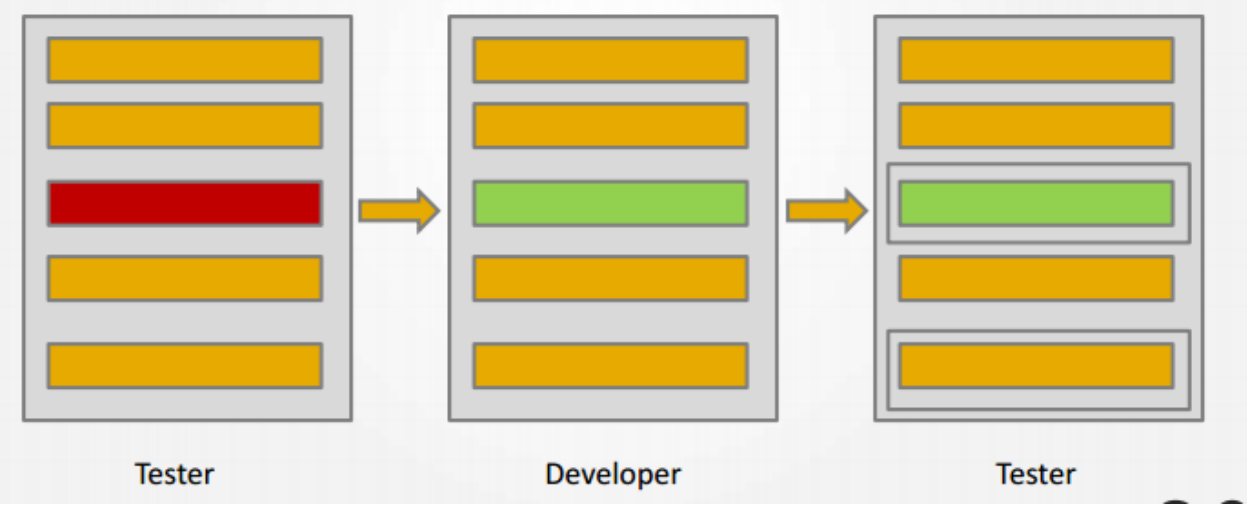
- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons −

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.



Acceptance Testing

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will reduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application −

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following −

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software −

- Network delay
- Client-side processing
- Database transaction processing
- Load balancing between servers
- Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects −

- Speed (i.e. Response Time, data rendering and accessing)
- Capacity

- Stability
- Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that usability is the quality requirement that can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end-user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that a user-friendly system should fulfill the following five goals, i.e., easy to Learn, easy to remember, efficient to use, satisfactory to use, and easy to understand.

In addition to the different definitions of usability, there are some standards and quality models and methods that define usability in the form of attributes and sub-attributes such as ISO-9126, ISO-9241-11, ISO-13407, and IEEE std.610.12, etc.

UI vs Usability Testing

UI testing involves testing the Graphical User Interface of the Software. UI testing ensures that the GUI functions according to the requirements and tested in terms of color, alignment, size, and other properties.

On the other hand, usability testing ensures a good and user-friendly GUI that can be easily handled. UI testing can be considered as a sub-part of usability testing.

Security Testing

Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure −

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

- Software is secure against known and unknown vulnerabilities
- Software data is secure
- Software is according to all security regulations
- Input checking and validation
- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities
- Directory traversal attacks

Portability Testing

Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well. Following are the strategies that can be used for portability testing −

- Transferring an installed software from one computer to another.
- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows −

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

Software Testing - Documentation

Testing documentation involves the documentation of artifacts that should be developed before or during the testing of Software.

Documentation for software testing helps in estimating the testing effort required, test coverage, requirement tracking/tracing, etc. This section describes some of the commonly used documented artifacts related to software testing such as −

- Test Plan
- Test Scenario
- Test Case

Test Plan

A test plan outlines the strategy that will be used to test an application, the resources that will be used, the test environment in which testing will be performed, and the limitations of the testing and the schedule of testing activities. Typically the Quality Assurance Team Lead will be responsible for writing a Test Plan.

A test plan includes the following −

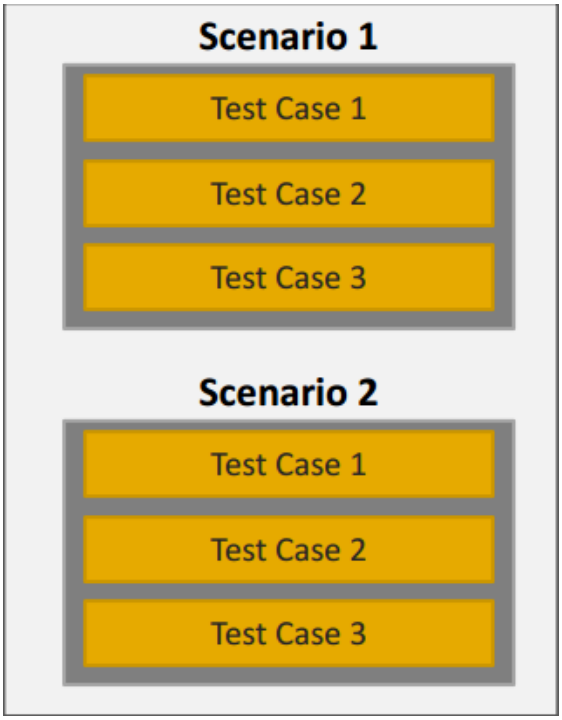
- Introduction to the Test Plan document
- Assumptions while testing the application
- List of test cases included in testing the application
- List of features to be tested
- What sort of approach to use while testing the software
- List of deliverables that need to be tested

- The resources allocated for testing the application
- Any risks involved during the testing process
- A schedule of tasks and milestones to be achieved

Test Scenario

It is a one line statement that notifies what area in the application will be tested. Test scenarios are used to ensure that all process flows are tested from end to end. A particular area of an application can have as little as one test scenario to a few hundred scenarios depending on the magnitude and complexity of the application.

The terms 'test scenario' and 'test cases' are used interchangeably, however a test scenario has several steps, whereas a test case has a single step. Viewed from this perspective, test scenarios are test cases, but they include several test cases and the sequence that they should be executed. Apart from this, each test is dependent on the output from the previous test.



Test Case

Test cases involve a set of steps, conditions, and inputs that can be used while performing testing tasks. The main intent of this activity is to ensure whether a software passes or fails in terms of its functionality and other aspects. There are many types of test cases such as functional, negative, error, logical test cases, physical test cases, UI test cases, etc.

Furthermore, test cases are written to keep track of the testing coverage of a software. Generally, there are no formal templates that can be used during test case writing. However, the following components are always available and included in every test case –

- Test case ID
- Product module
- Product version
- Revision history
- Purpose
- Assumptions
- Pre-conditions
- Steps
- Expected outcome
- Actual outcome
- Post-conditions

Many test cases can be derived from a single test scenario. In addition, sometimes multiple test cases are written for a single software which are collectively known as test suites.