

# UNIT 5: Software Testing

## Goals of Software Testing

The main goal of [software testing](#) is to find bugs as early as possible and fix bugs and make sure that the software is bug-free.

Important Goals of Software Testing:

- Detecting bugs as soon as feasible in any situation.
- Avoiding errors in a project's and product's final versions.
- Inspect to see whether the customer requirements criterion has been satisfied.
- Last but not least, the primary purpose of testing is to gauge the project and product level of quality.

The goals of software testing may be classified into three major categories as follows:

1. **Immediate Goals**
2. **Long-term Goals**
3. **Post-Implementation Goals**

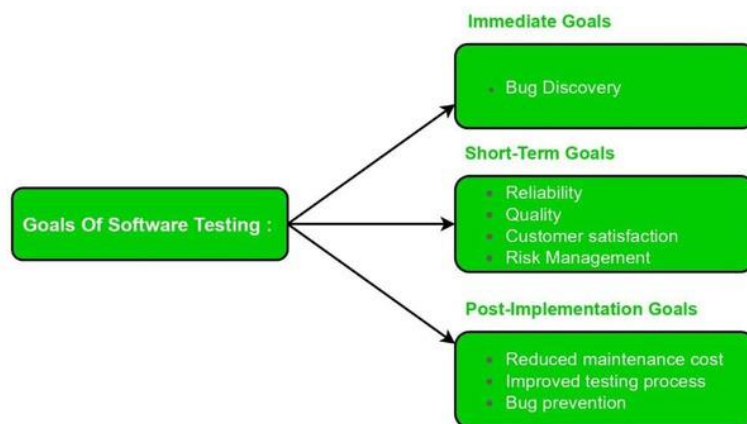


Fig : Software Testing Goals

**1. Immediate Goals:** These objectives are the direct outcomes of testing. These objectives may be set at any time during the SDLC process. Some of these are covered in detail below:

- **Bug Discovery:** This is the immediate goal of software testing to find errors at any stage of software development. The number of bugs is discovered in the early stage of testing. The primary purpose of software testing is to detect flaws at any step of the development process. The higher the number of issues detected at an early stage, the higher the software testing success rate.
- **Bug Prevention:** This is the immediate action of bug discovery, that occurs as a result of bug discovery. Everyone in the software development team learns how to code from the behavior and analysis of issues detected, ensuring that bugs are not duplicated in subsequent phases or future projects.

**2. Long-Term Goals:** These objectives have an impact on product quality in the long run after one cycle of the SDLC is completed. Some of these are covered in detail below:

- **Quality:** This goal enhances the quality of the software product. Because software is also a product, the user's priority is its quality. Superior quality is ensured by thorough testing. Correctness, integrity, efficiency, and reliability are all aspects that influence quality. To attain quality, you must achieve all of the above-mentioned quality characteristics.
- **Customer Satisfaction:** This goal verifies the customer's satisfaction with a developed software product. The primary purpose of software testing, from the user's standpoint, is customer satisfaction. Testing should be extensive and thorough if we want the client and customer to be happy with the software product.
- **Reliability:** It is a matter of confidence that the software will not fail. In short, reliability means gaining the confidence of the customers by providing them with a quality product.
- **Risk Management:** Risk is the probability of occurrence of uncertain events in the organization and the potential loss that could result in negative consequences. Risk management must be done to reduce the failure of the product and to manage risk in different situations.

**3. Post-Implemented Goals:** After the product is released, these objectives become critical. Some of these are covered in detail below:

- **Reduce Maintenance Cost:** Post-released errors are costlier to fix and difficult to identify. Because effective software does not wear out, the maintenance cost of any software product is not the same as the physical cost. The failure of a software product due to faults is the only expense of maintenance. Because they are difficult to discover, post-release mistakes always cost more to rectify. As a result, if testing

is done thoroughly and effectively, the risk of failure is lowered, and maintenance costs are reduced as a result.

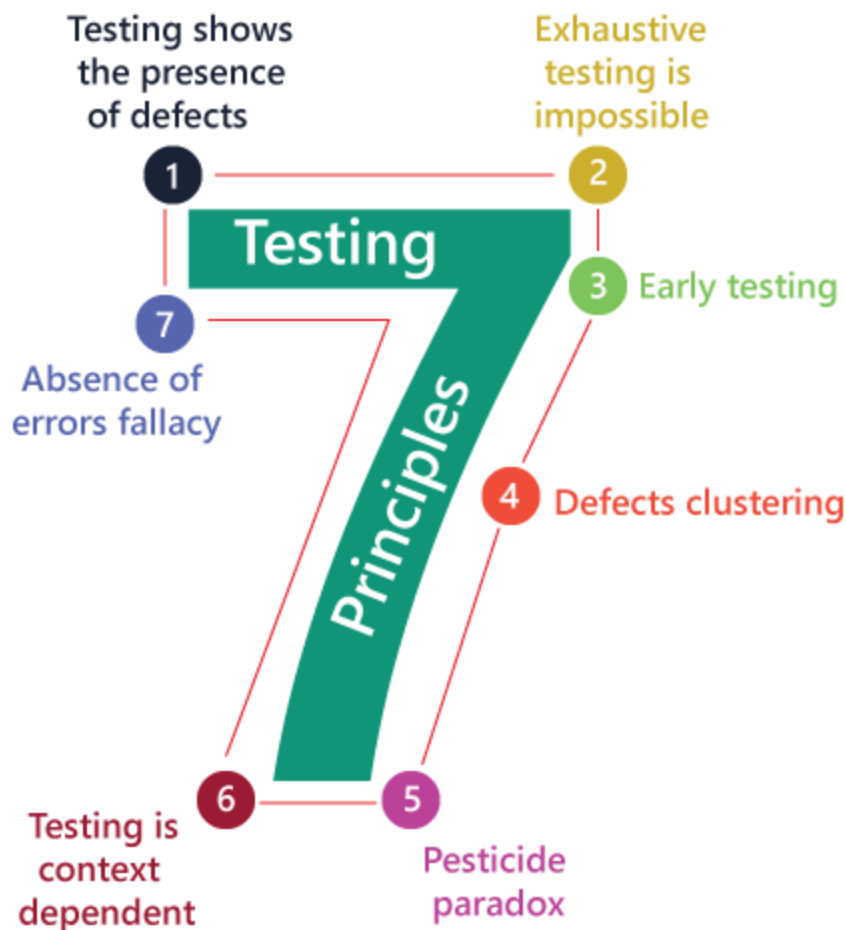
- **Improved Software Testing Process:** These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals. A project's testing procedure may not be completely successful, and there may be room for improvement. As a result, the bug history and post-implementation results can be evaluated to identify stumbling blocks in the current testing process that can be avoided in future projects.

## Software Testing Principles

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

Let us see the seven different testing principles, one by one:

- Testing shows the presence of defects
- Exhaustive Testing is not possible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context-dependent
- Absence of errors fallacy



## Testing shows the presence of defects

The test engineer will test the application to make sure that the application is bug or defects free. While doing testing, we can only identify that the application or software has any errors. The primary purpose of doing testing is to identify the numbers of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement, which means that to find any defects that might cause the product failure to meet the client's needs.

By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug-free while performing multiple types of testing on it. But at the time of deployment in the production server, if the end-user encounters those bugs which are not found in the testing process.

## Exhaustive Testing is not possible

Sometimes it seems to be very hard to test all the modules and their features with effective and non-effective combinations of the inputs data throughout the actual testing process.

Hence, instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful. So we can complete this type of variations according to the importance of the modules because the product timelines will not permit us to perform such type of testing scenarios.

## Early Testing

Here early testing means that all the testing activities should start in the early stages of the software development life cycle's **requirement analysis stage** to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

To perform testing, we will require the requirement specification documents; therefore, if the requirements are defined incorrectly, then it can be fixed directly rather than fixing them in another stage, which could be the development phase.

## Defect clustering

The defect clustering defined that throughout the testing process, we can detect the numbers of bugs which are correlated to a small number of modules. We have various reasons for this, such as the modules could be complicated; the coding part may be complex, and so on.

These types of software or the application will follow the **Pareto Principle**, which states that we can identify that approx. Eighty percent of the complication is present in 20 percent of the modules. With the help of this, we can find the uncertain modules, but this method has its difficulties if the same tests are performing regularly, hence the same test will not be able to identify the new defects.

## Pesticide paradox

This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently. And the new and different tests are

necessary to be written for the implementation of multiple parts of the application or the software, which helps us to find more bugs.

## **Testing is context-dependent**

Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

## **Absence of errors fallacy**

Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period would not help as testing is done on the wrong specification, which does not apply to the client's requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

## **Software Testability**

Software testability is measured with respect to the efficiency and effectiveness of testing. Efficient software architecture is very important for software testability. Software testing is a time-consuming, necessary activity in the software development lifecycle, and making this activity easier is one of the important tasks for software companies as it helps to reduce costs and increase the probability of finding bugs. There are certain metrics that could be used to measure testability in most of its aspects. Sometimes, testability is used to mean how adequately a particular set of tests will cover the product.

- Testability helps to determine the efforts required to execute test activities.
- Less the testability larger will be efforts required for testing and vice versa.

## **Factors of Software Testability**

Below are some of the metrics to measure software testability:

**1. Operability:** “The better it works, the more efficiently it can be tested.”

- The system has a few bugs (bugs add analysis and reporting overhead to the test process).
- No bugs block the execution of tests.
- The product evolves in functional stages (allows simultaneous development testing).

**2. Observability:** “What you see is what you test.”

- Distinct output is generated for each input.  
System states and variables are visible or queryable during execution. Past system states and variables are visible or queryable. For example, transaction logs.
- All factors affecting the output are visible.
- Incorrect output is easily identified.
- Internal errors are automatically detected through self-testing mechanisms. o Internal errors are automatically reported. co Source code is accessible.

**3. Controllability:** “The better we can control the software, the more the testing can be automated and optimized.”

- All possible outputs can be generated through some combination of inputs. All code is executable through some combination of input.
- Software and hardware states and variables can be controlled directly by the test engineer.
- Input and output formats are consistent and structured.

**4. Decomposability:** “By controlling the scope of testing, we can more problems and perform smarter retesting.” quickly isolate

- The software system is built from independent modules.
- Software modules can be tested independently.

**5. Simplicity:** “The less there is to test, the more quickly we can test it.”

- Functional simplicity (e.g., the feature set is the minimum necessary to meet requirements).
- Structural simplicity (e.g., architecture is modularized to limit the propagation of faults).
- Code simplicity (e.g., a coding standard is adopted for ease of inspection and maintenance).

**6. Stability:** “The fewer the changes, the fewer the disruptions to testing.” Changes to the software are infrequent.

- Changes to the software are controlled.
- Changes to the software do not invalidate existing tests. The software recovers well from failures.

**7. Understandability:** “The more information we have, the smarter we will test.”

- The design is well understood.
- Dependencies between internal, external, and shared components are well

- understood.
- Changes to the design are communicated. Technical documentation is instantly accessible.
- Technical documentation is well organized. Technical documentation is specific and detailed.
- Technical documentation is accurate.

**8. Availability:** “The more accessible objects are the easier is to design test cases”. It is all about the accessibility of objects or entities for performing the testing, including bugs, source code, etc.

**9. Testing Tools:** Testing tools that are easy to use will reduce the staff size and less training will be required.

**10. Documentation:** Specifications and requirements documents should be according to the client’s needs and fully featured.

## How to Measure Software Testability?

Software testability evaluates how easy it is to test the software and how likely software testing will find the defects in the application. Software testability assessment can be accomplished through software metrics assessment:

- Depth of Inheritance Tree.
- Fan Out (FOUT).
- Lack Of Cohesion Of Methods (LCOM).
- Lines Of Code per Class (LOCC).
- Response For Class (RFC).
- Weighted Methods per Class (WMC).

During software launch, it is crucial to determine which components may be more challenging to test. Software testability assessment is crucial during the start of the testing phase as it affects the efficiency of the planning process.

## Requirements of Software Testability

The attributes suggested by Bach can be used by a software engineer to develop a software configuration (i.e., programs, data, and documents) that is amenable to testing. Below are some of the capabilities that are associated with software testability requirements:

- **Module capabilities:** Software is developed in modules and each module will be tested separately. Test cases will be designed for each module and then the interaction between the modules will be tested.



- **Testing support capabilities:** The entry point to test drivers and root must be saved for each person, test interface as during the increment level testing, the trouble and accuracy level of testing root and driver should be given high priority and importance.
- **Defects disclosure capabilities:** The system errors should be less so that they do not block the software testing. The requirement document should also pass the following parameters to be testable:
  - The requirement must be accurate, correct, concise, and complete.
  - The requirement should be unambiguous i.e it should have one meaning for all staff members.
  - A requirement should not contradict other requirements.
  - Priority-based ranking of requirements should be implemented.
  - A requirement must be domain-based so that the changing requirements won't be a challenge to implement.
- **Observation capabilities:** Observing the software to monitor the inputs, their outcomes, and the factors influencing them.

## Types of Software Testability

Below are the different types of software testability:

**1. Object-oriented programs testability:** Software testing object-oriented software is done at three levels, Unit, Integration, and System testing. Unit testing is the most accessible level to get better software testability as one can apply testability examination earlier in the improvement life-cycle.

**2. Domain-based testability:** Software products that are developed with the concept of domain-driven development are easy to test and changes can be also done easily. The domain testable software is modifiable to make it observable and controllable.

**3. Testability based on modules:** The module-based approach for software testability consists of three stages:

- **Normalize program:** In this stage, the program needs to be normalized using some semantic and systematic tools to make it more reasonable for testability measures.
- **Recognize testable components:** In this stage, the testable components are recognized based on the demonstrated normalized data flow.
- **Measure program testability:** Program testability is measured based on the information stream testing criteria.

## Improving Software Testability

Below are some of the parameters that can be implemented in practice to improve software testability:

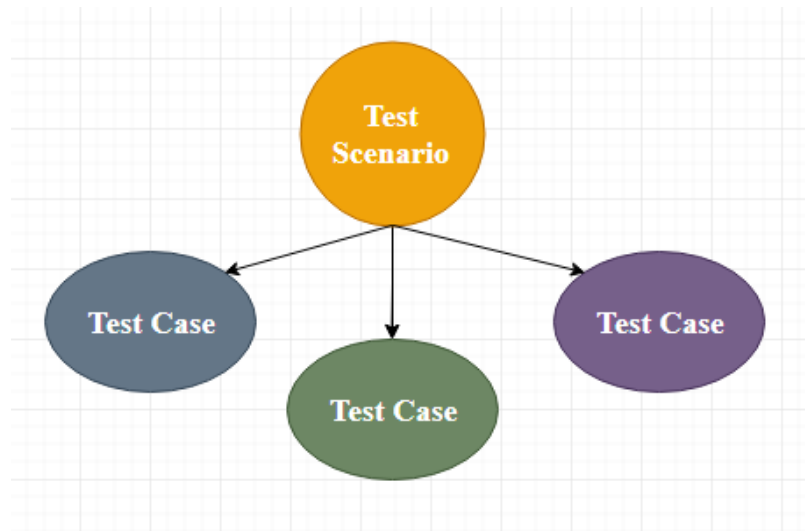
- **Appropriate test environment:** If the test environment corresponds to the production environment then testing will be more accurate and easier.
- **Adding tools for testers:** Building special instruments for manual testing helps to make the process easier and simpler.
- **Consistent element naming:** If the developers can ensure that they are naming the elements correctly, consistently, logically, and uniquely then it makes testing more convenient. Although this approach is difficult in larger projects with multiple developers and engineers.
- **Improve observability:** Improving observability provides unique outputs for unique inputs for the Software Under Test.
- **Adding assertions:** Adding assertions to the units in the software code helps to make the software more testable and find more defects.
- **Manipulating coupling:** Manipulating coupling to make it Domain dependent relative to the increased testability of code.
- **Internal logging:** If software accurately logs the internal state then manual testing can be streamlined and it enables to check of what is happening during any test.
- **Consistent UI design:** Consistent UI design also helps to improve software testability as the testers can easily comprehend how the user interface principles work.

## Benefits of software testability

- **Minimizes testers' efforts:** Testability calculates and minimizes the testers' efforts to perform testing as improved software testability facilitates estimating the difficulty in finding the software flaws.
- **Determines the volume of automated testing:** Software testability determines the volume of automated testing based on the software product's controllability.
- **Early detection of bugs:** Software testability helps in the early and effortless detection of bugs and thus saves time, cost, and effort required in the software development process.

# Test Case

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.



It is an in-details document that contains all possible inputs (positive as well as negative) and the navigation steps, which are used for the test execution process. Writing of test cases is a one-time attempt that can be used in the future at the time of regression testing.

Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.

To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

Generally, we will write the test case whenever the developer is busy in writing the code.

## When do we write a test case?

We will write the test case when we get the following:

- When the customer gives the business needs then, the developer starts developing and says that they need 3.5 months to build this product.
- And In the meantime, the testing team will **start writing the test cases**.
- Once it is done, it will send it to the Test Lead for the review process.
- And when the developers finish developing the product, it is handed over to the testing team.
- The test engineers never look at the requirement while testing the product document because testing is constant and does not depends on the mood of the person rather than the quality of the test engineer.

***Note: When writing the test case, the actual result should never be written as the product is still being in the development process. That's why the actual result should be written only after the execution of the test cases.***

## Why we write the test cases?

We will write the test for the following reasons:

- **To require consistency in the test case execution**
- **To make sure a better test coverage**
- **It depends on the process rather than on a person**
- **To avoid training for every new test engineer on the product**

**To require consistency in the test case execution:** we will see the test case and start testing the application.

**To make sure a better test coverage:** for this, we should cover all possible scenarios and document it, so that we need not remember all the scenarios again and again.

**It depends on the process rather than on a person:** A test engineer has tested an application during the first release, second release, and left the company at the time of third release. As the test engineer understood a module and tested the application thoroughly by deriving many values. If the person is not there for the third release, it

becomes difficult for the new person. Hence all the derived values are documented so that it can be used in the future.

**To avoid giving training for every new test engineer on the product:** When the test engineer leaves, he/she leaves with a lot of knowledge and scenarios. Those scenarios should be documented so that the new test engineer can test with the given scenarios and also can write the new scenarios.

**Note:** when the developers are developing the first product for the First release, the test engineer writes the test cases. And in the second release, when the new features are added, the test engineer writes the test cases for that also, and in the next release, when the elements are modified, the test engineer will change the test cases or writes the new test cases as well.

## Test case template

The primary purpose of writing a test case is to achieve the efficiency of the application.

### Header

Test Case Name/ID :-Release - Version - Application Name - Module

Test Case Type:-

F.T.C   I.T.C   S.T.C

Requirement Number:-

Module:-

Severity:- Critical/Major/Minor

Status:-

Release:-

Version:-

Pre-condition:-

Test Data:-

Summary:-

### Body

Step No.	Desci- pation	Inputs	Expected Result	Actual Reasult	Status	Comments
...	...	...	...	...	...	...
...	...	...	...	...	...	...

### Footer

Author:-

Reviewd By:-

Date:-

Approved By:-

As we know, the **actual result** is written after the test case execution, and most of the time, it would be same as the **expected result**. But if the test step will fail, it will be

different. So, the actual result field can be skipped, and in **the Comments** section, we can write about the bugs.

And also, the **Input field** can be removed, and this information can be added to the **Description field**.

The above template we discuss above is not the standard one because it can be different for each company and also with each application, which is based on the test engineer and the test lead. But, for testing one application, all the test engineers should follow a usual template, which is formulated.

The test case should be written in simple language so that a new test engineer can also understand and execute the same.

In the above sample template, the header contains the following:

### **Step number**

It is also essential because if step number 20 is failing, we can document the bug report and hence prioritize working and also decide if it's a critical bug.

### **Test case type**

It can be functional, integration or system test cases or positive or negative or positive and negative test cases.

### **Release**

One release can contain many versions of the release.

### **Pre-condition**

These are the necessary conditions that need to be satisfied by every test engineer before starting the test execution process. Or it is the data configuration or the data setup that needs to be created for the testing.

**For example:** In an application, we are writing test cases to add users, edit users, and delete users. The pre-condition will be seen if user A is added before editing it and removing it.

## Test data

These are the values or the input we need to create as per the per-condition.

**For example**, Username, Password, and account number of the users.

The test lead may be given the test data like username or password to test the application, or the test engineer may themselves generate the username and password.

## Severity

The severity can be **major, minor, and critical**, the severity in the test case talks about the importance of that particular test cases. All the test execution process always depends on the severity of the test cases.

We can choose the severity based on the module. There are many features include in a module, even if one element is critical, we claim that test case to be critical. It depends on the functions for which we are writing the test case.

**For example**, we will take the Gmail application and let us see the severity based on the modules:

Modules	Severity
Login	Critical
Help	Minor
Compose mail	Critical
Setting	Minor
Inbox	Critical
Sent items	Major

Logout	Critical
--------	----------

And for the banking application, the severity could be as follows:

Modules	Severity
Amount transfer	critical
Feedback	minor

### Brief description

The test engineer has written a test case for a particular feature. If he/she comes and reads the test cases for the moment, he/she will not know for what feature has written it. So, the brief description will help them in which feature test case is written.

## Example of a test case template

Here, we are writing a test case for the **ICICI application's Login** module:



New Microsoft Excel Worksheet - Microsoft Excel						
File Home Insert Page Layout Formulas Data Review View						
Clipboard Font Alignment Number Styles						
D5						
	A	B	C	D	E	F
1	<b>Test case template</b>					
2	test case name	Delta-3.0-ICICI-Login				
3	test case type	Functional test case				
4	requirement no	1				
5	module	login				
6	status	XXX				
7	severity	critical				
8	release	Delta				
9	version	3				
10	pre-condition	required one login				
11	test data	username-abc, password-123				
12	summary	to check the functionality of login				
13	Steps no	Description	Inputs	Expected result	Actual results	Status
14	1	open "Browser" and enter the "Url"	<a href="https://QA/Main/">https://QA/Main/</a>	Login page must be display	As Expected	pass
15	2	enter the following values for "Username" :				
16		Valid(abc)	abc	Accept	Login page must be display	pass
17		Invalid	555	Error message "invalid login"	not as expected	fail
18		Blank	Null	Error message username cannot t	not as expected	fail
19		Symbols	2 alphabet	Error message invalid login	not as expected	fail
20	3	enter the following values for "Password":				
21		valid	123	Accept	Login page must be display	pass
22		invalid	xy3	Error message invalid login	not as expected	fail
23		Blank		Error message password cannot t	not as expected	fail
24		enter the valid username and password and				
25	4	click on "OK" button	abc,123	"home Pag" must be displayed	home page is displayin	pass
26		enter the valid username and password and				
27	5	click on "Cancel" button	abc,123	all field must be cleard	the entered data is clea	pass
28	author	test engineer name				
29	date	1/4/2020				
30	reviewed by	ryan				
31	approved by	jessica				
32						

## Types of test cases

We have a different kind of test cases, which are as follows:

- **Function test cases**
- **Integration test cases**
- **System test cases**

## The functional test cases

Firstly, we check for which field we will write test cases and then describe accordingly.

In functional testing or if the application is data-driven, we require the input column else; it is a bit time-consuming.

### Rules to write functional test cases:

- In the expected results column, try to use **should be** or **must be**.
- Highlight the Object names.
- We have to describe only those steps which we required the most; otherwise, we do not need to define all the steps.
- To reduce the excess execution time, we will write steps correctly.
- Write a generic test case; do not try to hard code it.

Let say it is the amount transfer module, so we are writing the functional test cases for it and then also specifies that it is not a login feature.

The image shows a green-themed 'AMOUNT TRANSFER' form. On the left, there is a vertical white sidebar with five sets of four dots (\*\*\*\*). The main form area contains the following elements:

- AMOUNT TRANSFER** (Title)
- From Account Number** (Label) with a white input field and '(FAN)' text below it.
- To Account Number** (Label) with a white input field and '(TAN)' text below it.
- Amount** (Label) with a white input field.
- TRANSFER** (Button)
- CANCEL** (Button)

The functional test case for amount transfer module is in the below Excel file:

<b>Functional Test case template</b>						
Test case name	beta-1.0-ICIQ-amount transfer					
Test case type	Functional test case					
Requirement no	6					
Module	amount transfer					
Status	XXX					
Severity	Critical					
Release	Beta					
Version	1					
Pre-condition	sender login two account number Balance--> exist					
Test data	Username:xyz, Password:1234 1231, 4321 3000-9000					
Summary	to check the functionality of amount transfer					
<b>Steps no</b>	<b>Description</b>	<b>Inputs</b>	<b>Expected result</b>	<b>Actual results</b>	<b>Status</b>	<b>Comments</b>
1	Open "Browser" and enter the "Url"	<a href="https://QA/Main/#!/">https://QA/Main/#!/</a>	login page must be display	As Expected	pass	XXX
2	Enter the following values for "Username" and "Password" and click on the "OK" button	xyz, 1234	"Home page" must be displayed	As Expected	pass	XXX
3	Click on the "Amount Transfer"	Null			pass	XXX
4	Enter the following for From Account number (FAN):					
	valid	1234	Accept	As Expected	pass	
	invalid	1124	Error message invalid account		fail	
	blank		Error message FAN value cannot be blank		fail	
			test maximum coverage			
	Enter the following values for "TO account number" (TAN)					
5	valid	4321	Accept	As expected	pass	XXX
	invalid	6655	Error message invalid account		fail	
	Blank		Error message TAN value cannot be blank			
			test maximum coverage			
6	enter the value for "Amount"					
	valid	5000, 5001, 9000, 84	Accept	As expected	pass	XXX
	invalid	4999, 9001	error message amount should be between (5000-9000)		fail	
7	Enter the value for "FAN, TAN, Amount" click on the "Transfer" button					
	FAN	1234				
			"Confirmation Message" amount transfer successfully must be displayed	As expected	pass	XXX
	TAN	4321				
	Amount	6000				
8	Enter the value for "FAN, TAN, Amount" click on the "Cancel" button					
	FAN	1234				
	TAN	4321	All field must be cleared	As expected	pass	XXX
	Amount	6000				
Author	Sern					
Date	4/1/2020					
Reviewed by	jessica					
Approved by	ryan					

if we are saying (5000-9000) balance, but if want to test for 9001, so obviously it will give the error message (unufficient message)

## Integration test case

In this, we should not write something which we already covered in the functional test cases, and something we have written in the integration test case should not be written in the system test case again.

### Rules to write integration test cases

- Firstly, understand the product

- Identify the possible scenarios
- Write the test case based on the priority

When the test engineer writing the test cases, they may need to consider the following aspects:

If the test cases are in details:

- They will try to achieve maximum test coverage.
- All test case values or scenarios are correctly described.
- They will try to think about the execution point of view.
- The template which is used to write the test case must be unique.

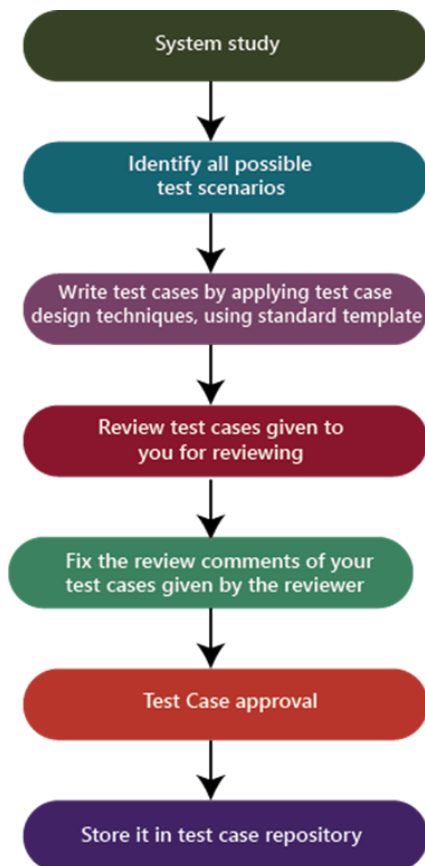
***Note: when we involve fewer numbers of steps or coverage is more, it should be the best test case, and when these test cases are given to anyone, they will understand easily.***

## System test cases

We will write the system test cases for the end-to-end business flows. And we have the entire modules ready to write the system test cases.

## The process to write test cases

The method of writing a test case can be completed into the following steps, which are as below:



## System study

In this, we will understand the application by looking at the requirements or the SRS, which is given by the customer.

## Identify all scenarios:

- When the product is launched, what are the possible ways the end-user may use the software to identify all the possible ways.
- I have documented all possible scenarios in a document, which is called test design/high-level design.
- The test design is a record having all the possible scenarios.

## Write test cases

Convert all the identified scenarios to test claims and group the scenarios related to their features, prioritize the module, and write test cases by applying test case design

techniques and use the standard test case template, which means that the one which is decided for the project.

### **Review the test cases**

Review the test case by giving it to the head of the team and, after that, fix the review feedback given by the reviewer.

### **Test case approval**

After fixing the test case based on the feedback, send it again for the approval.

### **Store in the test case repository**

After the approval of the particular test case, store in the familiar place that is known as the test case repository.

## **Difference between White Box testing and Black Box testing**

In this article, we will discuss white box testing and black box testing, along with the comparison between them. In Black box testing (or "behavioral testing"), the tester understands what the program is supposed to do, rather than its internal working. Whereas, in White box testing, there is a testing of internal coding and infrastructure of software.

Before jumping directly to the comparison, let's first see a brief description of white-box and black-box testing.

### **White Box testing**

The term 'white box' is used because of the internal perspective of the system. The **clear box or white box, or transparent box** name denotes the ability to see through the software's outer shell into its inner workings.



It is performed by Developers, and then the software will be sent to the testing team, where they perform black-box testing. The main objective of white-box testing is to test the application's infrastructure. It is done at lower levels, as it includes unit testing and integration testing. It requires programming knowledge, as it majorly focuses on code structure, paths, conditions, and branches of a program or software. The primary goal of white-box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

It is also known as structural testing, clear box testing, code-based testing, and transparent testing. It is well suitable and recommended for algorithm testing.

To read more about white box testing, you can refer to the following link – [White box testing](#).

### **Black Box testing**

The primary source of black-box testing is a specification of requirements that are stated by the customer. It is another type of manual testing. It is a software testing technique that examines the functionality of the software without knowing its internal structure or coding. It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function. In this testing, the test engineer analyzes the software against requirements, identifies the defects or bugs, and sends it back to the development team.



In this method, the tester selects a function and gives input value to examine its functionality, and checks whether the function is giving the expected output or not. If the function produces the correct output, then it is passed in testing, otherwise failed.

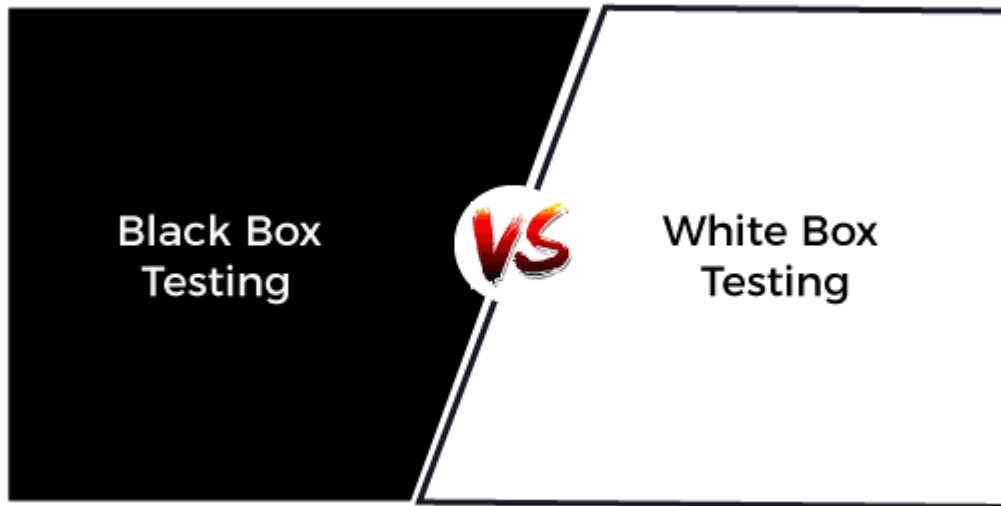
Black box testing is less exhaustive than White Box and Grey Box testing methods. It is the least time-consuming process among all the testing processes. The main objective of implementing black box testing is to specify the business needs or the customer's requirements.

In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer's requirement. Mainly, there are three types of black-box testing: **functional testing**, **Non-Functional testing**, and **Regression testing**. Its main objective is to specify the business needs or the customer's requirements.

To read more about black box testing, you can refer to the following link – [Black Box testing](#).



## White box testing v/s Black box testing



Now, let's see the comparison chart between white-box testing and black-box testing. We are comparing both terms on the basis of some characteristics.

S.no.	On the basis of	Black Box testing	White Box testing
1.	<b>Basic</b>	It is a software testing technique that examines the functionality of software without knowing its internal structure or coding.	In white-box testing, the internal structure of the software is known to the tester.
2.	<b>Also known as</b>	Black Box Testing is also known as functional testing, data-driven testing, and closed-box testing.	It is also known as structural testing, clear box testing, code-based testing, and transparent testing.
3.	<b>Programming knowledge</b>	In black-box testing, there is less programming knowledge is required.	In white-box testing, there is a requirement of programming knowledge.

4.	<b>Algorithm testing</b>	It is not well suitable for algorithm testing.	It is well suitable and recommended for algorithm testing.
5.	<b>Usage</b>	It is done at higher levels of testing that are system testing and acceptance testing.	It is done at lower levels of testing that are unit testing and integration testing.
6.	<b>Automation</b>	It is hard to automate black-box testing due to the dependency of testers and programmers on each other.	It is easy to automate the white box testing.
7.	<b>Tested by</b>	It is mainly performed by the software testers.	It is mainly performed by developers.
8.	<b>Time-consuming</b>	It is less time-consuming. In Black box testing, time consumption depends upon the availability of the functional specifications.	It is more time-consuming. It takes a long time to design test cases due to lengthy code.
9.	<b>Base of testing</b>	The base of this testing is external expectations.	The base of this testing is coding which is responsible for internal working.
10.	<b>Exhaustive</b>	It is less exhaustive than White Box testing.	It is more exhaustive than Black Box testing.
11.	<b>Implementation knowledge</b>	In black-box testing, there is no implementation knowledge is required.	In white-box testing, there is a requirement of implementation knowledge.

12.	<b>Aim</b>	The main objective of implementing black box testing is to specify the business needs or the customer's requirements.	Its main objective is to check the code quality.
13.	<b>Defect detection</b>	In black-box testing, defects are identified once the code is ready.	Whereas, in white box testing, there is a possibility of early detection of defects.
14.	<b>Testing method</b>	It can be performed by trial and error technique.	It can test data domain and data boundaries in a better way.
15.	<b>Types</b>	Mainly, there are three types of black-box testing: <b>functional testing, Non-Functional testing, and Regression testing.</b>	The types of white box testing are – <b>Path testing, Loop testing, and Condition testing.</b>
16.	<b>Errors</b>	It does not find the errors related to the code.	In white-box testing, there is the detection of hidden errors. It also helps to optimize the code.

## Conclusion

So, both white box testing and black box testing are required for the successful delivery of software. But 100% testing is not possible with both cases. Tester is majorly responsible for finding the maximum defects to improve the application's efficiency. Both black box testing and white box testing are done to certify that an application is working as expected.

Hence, it is necessary to understand both testing techniques. It will also be helpful to learn the difference between both terms to effectively pick up the right option.

# Unit Testing

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with its performance.

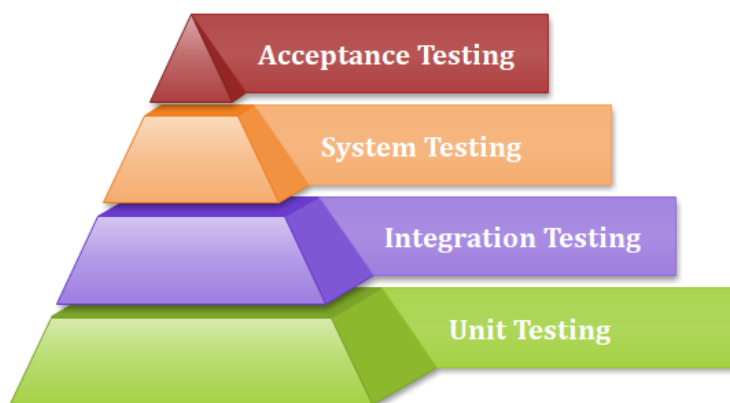
A unit is a single testable part of a software system and tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as **Unit testing** or **components testing**.

## Why Unit Testing?

In a testing level hierarchy, unit testing is the first level of testing done before integration and other remaining levels of the testing. It uses modules for the testing process which reduces the dependency of waiting for Unit testing frameworks, stubs, drivers and mock objects are used for assistance in unit testing.



Generally, **the** software goes under four level of testing: Unit Testing, Integration Testing, System Testing, and Acceptance Testing but sometimes due to time consumption software testers does minimal unit testing but skipping of unit testing may lead to higher

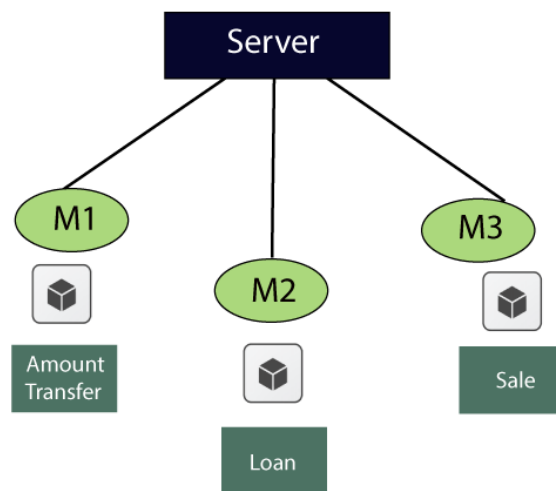
defects during Integration Testing, System Testing, and Acceptance Testing or even during Beta Testing which takes place after the completion of software application.

**Some crucial reasons are listed below:**

- Unit testing helps tester and developers to understand the base of code that makes them able to change defect causing code quickly.
- Unit testing helps in the documentation.
- Unit testing fixes defects very early in the development phase that's why there is a possibility to occur a smaller number of defects in upcoming testing levels.
- It helps with code reusability by migrating code and test cases.

## Example of Unit testing

Let us see one sample example for a better understanding of the concept of unit testing:



For the **amount transfer**, requirements are as follows:

1.	Amount transfer
1.1	From account number (FAN)→ Text Box
1.1.1	FAN→ accept only 4 digit

1.2	To account no (TAN)→ Text Box
1.2.1	TAN→ Accept only 4 digit
1.3	Amount→ Text Box
1.3.1	Amount → Accept maximum 4 digit
1.4	Transfer→ Button
1.4.1	Transfer → Enabled
1.5	Cancel→ Button
1.5.1	Cancel→ Enabled

Below are the application access details, which is given by the customer

- URL→ login Page
- Username/password/OK → home page
- To reach Amount transfer module follow the below

### **Loans → sales → Amount transfer**

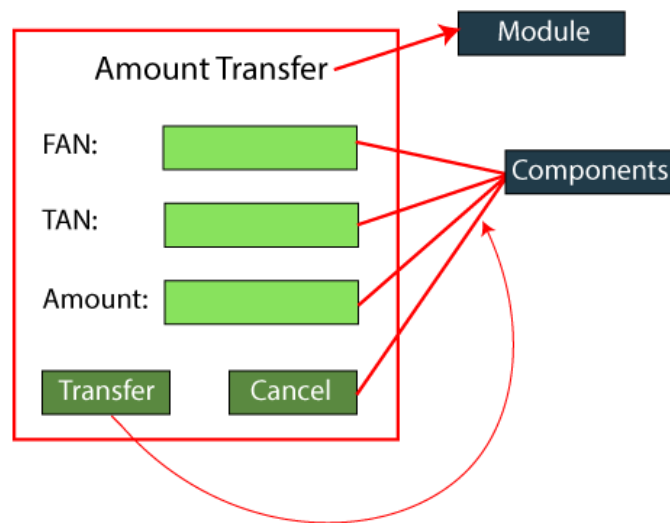
While performing unit testing, we should follow some rules, which are as follows:

- To start unit testing, at least we should have one module.
- Test for positive values
- Test for negative values
- No over testing
- No assumption required

When we feel that the **maximum test coverage** is achieved, we will stop the testing.

Now, we will start performing the unit testing on the different components such as

- **From account number(FAN)**
- **To account number(TAN)**
- **Amount**
- **Transfer**
- **Cancel**



### For the FAN components

Values	Description
1234	accept
4311	Error message→ account valid or not
blank	Error message→ enter some values
5 digit/ 3 digit	Error message→ accept only 4 digit
Alphanumeric	Error message → accept only digit

Blocked account no	Error message
Copy and paste the value	Error message→ type the value
Same as FAN and TAN	Error message

### For the TAN component

- Provide the values just like we did in **From account number** (FAN) components

### For Amount component

- Provide the values just like we did in FAN and TAN components.

### For Transfer component

- Enter valid FAN value
- Enter valid TAN value
- Enter the correct value of Amount
- Click on the Transfer button→ amount transfer successfully( confirmation message)

### For Cancel Component

- Enter the values of FAN, TAN, and amount.
- Click on the Cancel button → all data should be cleared.

## Unit Testing Tools

We have various types of unit testing tools available in the market, which are as follows:

- NUnit
- JUnit
- PHPUnit
- Parasoft Jtest
- EMMA



For more information about Unit testing tools, refers to the below link:

## Unit Testing Techniques:

Unit testing uses all white box testing techniques as it uses the code of software application:

- Data flow Testing
- Control Flow Testing
- Branch Coverage Testing
- Statement Coverage Testing
- Decision Coverage Testing

## How to achieve the best result via Unit testing?

Unit testing can give best results without getting confused and increase complexity by following the steps listed below:

- Test cases must be independent because if there is any change or enhancement in requirement, the test cases will not be affected.
- Naming conventions for unit test cases must be clear and consistent.
- During unit testing, the identified bugs must be fixed before jump on next phase of the SDLC.
- Only one code should be tested at one time.
- Adopt test cases with the writing of the code, if not doing so, the number of execution paths will be increased.
- If there are changes in the code of any module, ensure the corresponding unit test is available or not for that module.

## Advantages and disadvantages of unit testing

The pros and cons of unit testing are as follows:

## Advantages

- Unit testing uses module approach due to that any part can be tested without waiting for completion of another parts testing.
- The developing team focuses on the provided functionality of the unit and how functionality should look in unit test suits to understand the unit API.
- Unit testing allows the developer to refactor code after a number of days and ensure the module still working without any defect.

## Disadvantages

- It cannot identify integration or broad level error as it works on units of the code.
- In the unit testing, evaluation of all execution paths is not possible, so unit testing is not able to catch each and every error in a program.
- It is best suitable for conjunction with other testing activities.

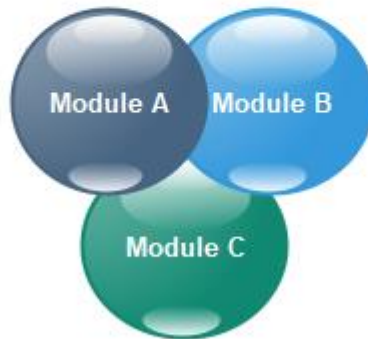
## Integration testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.



Tested in Unit Testing

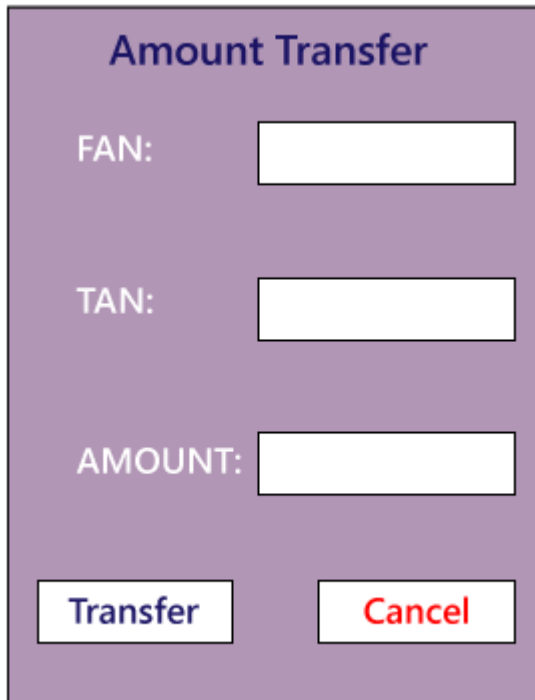


Under Integration Testing

Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as **integration testing**.

Let us see one sample example of a banking application, as we can see in the below image of amount transfer.

Advertisement

A purple rectangular form titled "Amount Transfer" in bold blue text. It contains three input fields: "FAN:" with a white rectangular box, "TAN:" with a white rectangular box, and "AMOUNT:" with a white rectangular box. At the bottom, there are two buttons: "Transfer" in blue text on a white background, and "Cancel" in red text on a white background.

**Amount Transfer**

FAN:

TAN:

AMOUNT:

**Transfer** **Cancel**

- First, we will login as a user **P** to amount transfer and send Rs200 amount, the confirmation message should be displayed on the screen as **amount transfer successfully**. Now logout as P and login as user **Q** and go to amount balance page and check for a balance in that account = Present balance + Received Balance. Therefore, the integration test is successful.
- Also, we check if the amount of balance has reduced by Rs200 in P user account.
- Click on the transaction, in P and Q, the message should be displayed regarding the data and time of the amount transfer.

## Guidelines for Integration Testing

- We go for the integration testing only after the functional testing is completed on each module of the application.
- We always do integration testing by picking module by module so that a proper sequence is followed, and also we don't miss out on any integration scenarios.
- First, determine the test case strategy through which executable test cases can be prepared according to test data.
- Examine the structure and architecture of the application and identify the crucial modules to test them first and also identify all possible scenarios.

- Design test cases to verify each interface in detail.
- Choose input data for test case execution. Input data plays a significant role in testing.
- If we find any bugs then communicate the bug reports to developers and fix defects and retest.
- Perform **positive and negative integration testing**.

Here **positive** testing implies that if the total balance is Rs15, 000 and we are transferring Rs1500 and checking if the amount transfer works fine. If it does, then the test would be a pass.

And **negative testing** means, if the total balance is Rs15, 000 and we are transferring Rs20, 000 and check if amount transfer occurs or not, if it does not occur, the test is a pass. If it happens, then there is a bug in the code, and we will send it to the development team for fixing that bug.

**Note:** Any application in this world will do functional testing compulsory, whereas integration testing will be done only if the modules are dependent on each other. Each integration scenarios should compulsorily have source→ data→destination. Any scenarios can be called as integration scenario only if the data gets saved in the destination.

**For example:** In the Gmail application, the **Source** could be **Compose**, **Data** could be **Email** and the **Destination** could be the **Inbox**.

## Example of integration testing

Let us assume that we have a **Gmail** application where we perform the integration testing.

First, we will do **functional testing** on the **login page**, which includes the various components such as **username, password, submit, and cancel** button. Then only we can perform integration testing.

The different integration scenarios are as follows:

**Compose Mail**

**Inbox**  
**Compose mail**  
**Sent Items**  
**Trash**  
**Spam**  
**Contact**  
**Folders**  
**Logout**

**To**

**From**

**Subject**

**TEXT FIELD**

☐ **Save To Draft**  
☐ **Add To Contact**

**SEND** **CANCEL**

### Scenarios1:

- First, we login as **P** users and click on the **Compose** mail and performing the functional testing for the specific components.
- Now we click on the **Send** and also check for **Save Drafts**.
- After that, we send a **mail** to **Q** and verify in the **Sent Items** folder of P to check if the send mail is there.
- Now, we will **log out** as P and login as Q and move to the **Inbox** and verify that if the mail has reached.

**Secanrios2:** We also perform the integration testing on **Spam** folders. If the particular contact has been marked as spam, then any mail sent by that user should go to the spam folder and not in the inbox.

**Note:** We will perform functional testing for all features, such as to send items, inbox, and so on.

As we can see in the below image, we will perform the **functional testing** for all the **text fields and every feature**. Then we will perform **integration testing** for the related functions. We first test the **add user, list of users, delete user, edit user**, and then **search user**.

The image shows a web application interface for adding users. It has a light green background. On the left, there is a white sidebar menu with the following items: 'Add User', 'Delete User', 'List User', 'Edit User', 'Product Sales', 'Product Purchases', 'Search Users', and 'Help'. The main area is titled 'Add Users' in bold black text. Below the title, there are several input fields with red labels: 'User Name', 'Password', 'Designation' (which is a dropdown menu with 'Team Lead' and 'Manager' as visible options), 'Email', 'Telephone', and 'Address'. At the bottom of the form, there are two buttons: 'Submit' and 'Cancel'.

**Note:**

- There are some features, we might be performing only the **functional testing**, and there are some features where we are performing both **functional** and **integration testing** based on the feature's requirements.
- **Prioritizing is essential**, and we should perform it at all the phases, which means we will open the application and select which feature needs to be tested first. Then go to that feature and choose which component must be tested first. Go to those components and determine what values to be entered first. And don't apply the same rule everywhere because testing logic varies from feature to feature.

- While performing testing, we should test one feature entirely and then only proceed to another function.
- Among the two features, we must be performing **only positive integrating testing** or both **positive and negative integration** testing, and this also depends on the features need.

## Reason Behind Integration Testing

Although all modules of software application already tested in unit testing, errors still exist due to the following reasons:

1. Each module is designed by individual software developer whose programming logic may differ from developers of other modules so; integration testing becomes essential to determine the working of software modules.
2. To check the interaction of software modules with the database whether it is an erroneous or not.
3. Requirements can be changed or enhanced at the time of module development. These new requirements may not be tested at the level of unit testing hence integration testing becomes mandatory.
4. Incompatibility between modules of software could create errors.
5. To test hardware's compatibility with software.
6. If exception handling is inadequate between modules, it can create bugs.

## Integration Testing Techniques

Any testing technique (Blackbox, Whitebox, and Greybox) can be used for Integration Testing; some are listed below:

### Black Box Testing

- State Transition technique
- Decision Table Technique
- Boundary Value Analysis
- All-pairs Testing
- Cause and Effect Graph



- Equivalence Partitioning
- Error Guessing

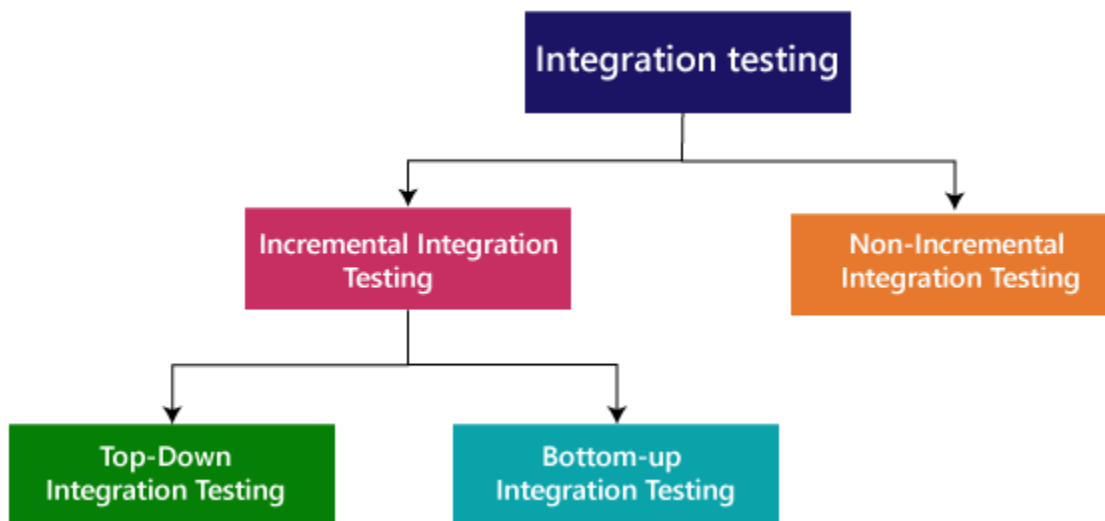
## White Box Testing

- Data flow testing
- Control Flow Testing
- Branch Coverage Testing
- Decision Coverage Testing

## Types of Integration Testing

Integration testing can be classified into two parts:

- **Incremental integration testing**
- **Non-incremental integration testing**



## Incremental Approach

In the Incremental Approach, modules are added in ascending order one by one or according to need. The selected modules must be logically related. Generally, two or more than two modules are added and tested to determine the correctness of functions. The process continues until the successful testing of all the modules.

**OR**

In this type of testing, there is a strong relationship between the dependent modules. Suppose we take two or more modules and verify that the data flow between them is working fine. If it is, then add more modules and test again.



**For example:** Suppose we have a Flipkart application, we will perform incremental integration testing, and the flow of the application would like this:

Flipkart→ Login→ Home → Search→ Add cart→Payment → Logout

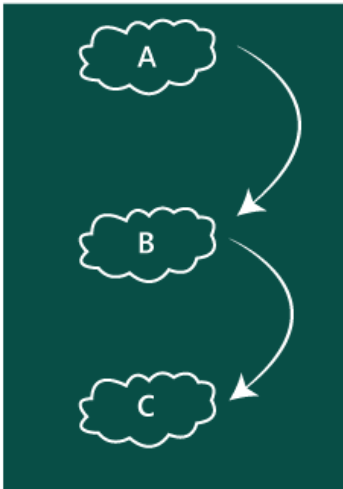
Incremental integration testing is carried out by further methods:

- Top-Down approach
- Bottom-Up approach

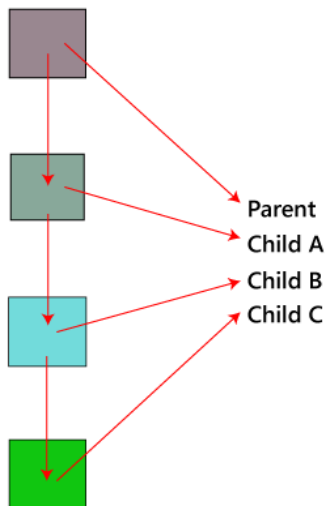
### Top-Down Approach

The top-down testing strategy deals with the process in which higher level modules are tested with lower level modules until the successful completion of testing of all the modules. Major design flaws can be detected and fixed early because critical modules tested first. In this type of method, we will add the modules incrementally or one by one and check the data flow in the same order.

### Top-Down Approach



In the top-down approach, we will be ensuring that the module we are adding is the **child of the previous one** like **Child C is a child of Child B** and so on as we can see in the below image:



### Advantages:

- Identification of defect is difficult.
- An early prototype is possible.

### Disadvantages:

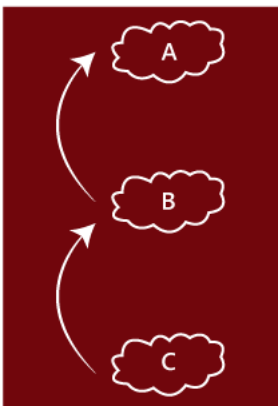
- Due to the high number of stubs, it gets quite complicated.

- Lower level modules are tested inadequately.
- Critical Modules are tested first so that fewer chances of defects.

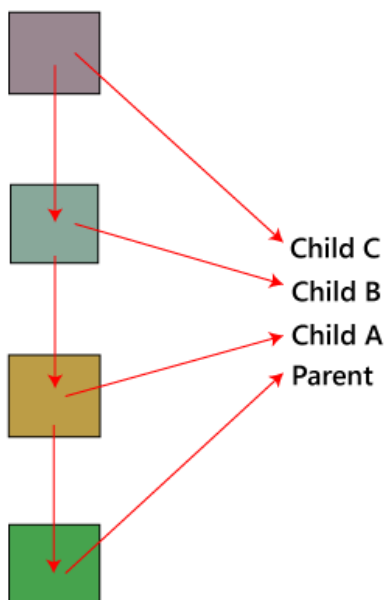
## Bottom-Up Method

The bottom to up testing strategy deals with the process in which lower level modules are tested with higher level modules until the successful completion of testing of all the modules. Top level critical modules are tested at last, so it may cause a defect. Or we can say that we will be adding the modules from **bottom to the top** and check the data flow in the same order.

Bottom-up Approach



In the bottom-up method, we will ensure that the modules we are adding **are the parent of the previous one** as we can see in the below image:



## Advantages

- Identification of defect is easy.
- Do not need to wait for the development of all the modules as it saves time.

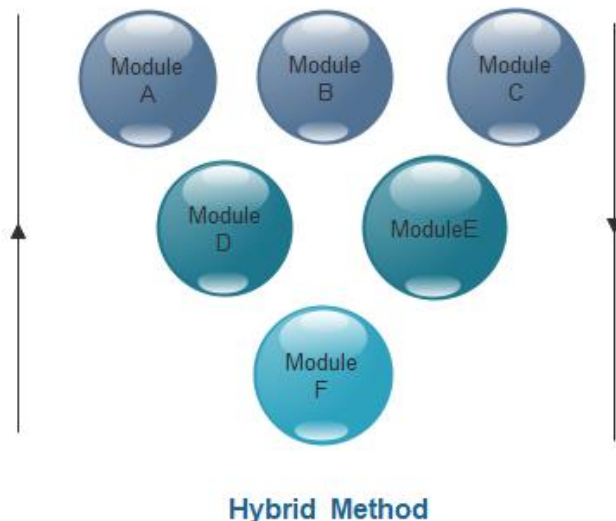
## Disadvantages

- Critical modules are tested last due to which the defects can occur.
- There is no possibility of an early prototype.

In this, we have one addition approach which is known as **hybrid testing**.

## Hybrid Testing Method

In this approach, both **Top-Down** and **Bottom-Up** approaches are combined for testing. In this process, top-level modules are tested with lower level modules and lower level modules tested with high-level modules simultaneously. There is less possibility of occurrence of defect because each module interface is tested.



## Advantages

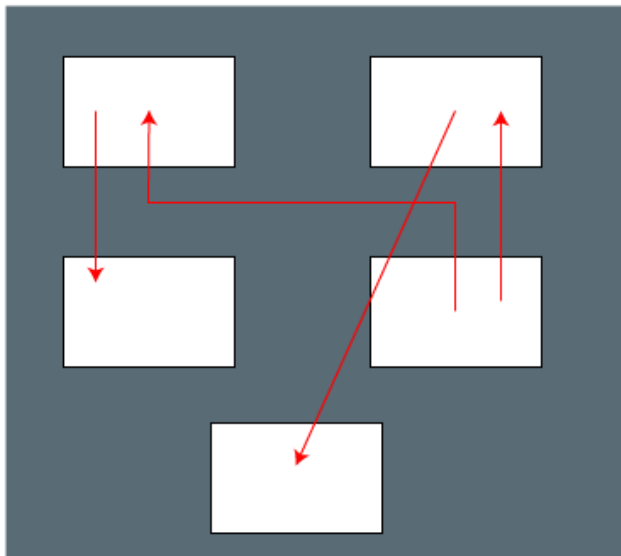
- The hybrid method provides features of both Bottom Up and Top Down methods.
- It is most time reducing method.
- It provides complete testing of all modules.

## Disadvantages

- This method needs a higher level of concentration as the process carried out in both directions simultaneously.
- Complicated method.

## Non- incremental integration testing

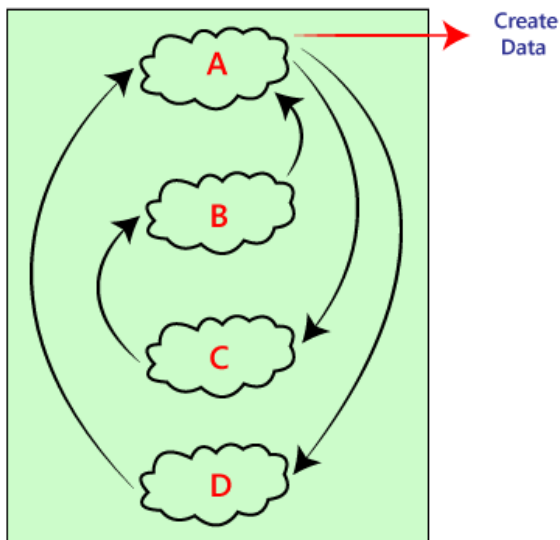
We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. Hence, it is also known as the **Big bang method**.



## Big Bang Method

In this approach, testing is done via integration of all modules at once. It is convenient for small software systems, if used for large software systems identification of defects is difficult.

Since this testing can be done after completion of all modules due to that testing team has less time for execution of this process so that internally linked interfaces and high-risk critical modules can be missed easily.



### Advantages:

- It is convenient for small size software systems.

### Disadvantages:

- Identification of defects is difficult because finding the error where it came from is a problem, and we don't know the source of the bug.
- Small modules missed easily.
- Time provided for testing is very less.
- We may miss to test some of the interfaces.

## Software Testing Strategies

Software testing is the process of evaluating a software application to identify if it meets specified requirements and to identify any defects. The following are common testing strategies:

1. **Black box testing** – Tests the functionality of the software without looking at the internal code structure.
2. **White box testing** – Tests the internal code structure and logic of the software.
3. **Unit testing** – Tests individual units or components of the software to ensure they are functioning as intended.
4. **Integration testing** – Tests the integration of different components of the software to ensure they work together as a system.

5. **Functional testing** – Tests the functional requirements of the software to ensure they are met.
6. **System testing** – Tests the complete software system to ensure it meets the specified requirements.
7. **Acceptance testing** – Tests the software to ensure it meets the customer's or end-user's expectations.
8. **Regression testing** – Tests the software after changes or modifications have been made to ensure the changes have not introduced new defects.
9. **Performance testing** – Tests the software to determine its performance characteristics such as speed, scalability, and stability.
10. **Security testing** – Tests the software to identify vulnerabilities and ensure it meets security requirements.

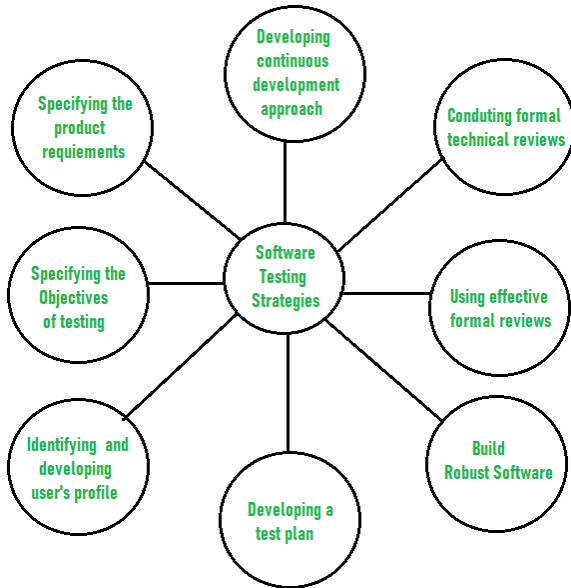
[Software Testing](#) is a type of investigation to find out if there is any default or error present in the software so that the errors can be reduced or removed to increase the quality of the software and to check whether it fulfills the specified requirements or not.

According to Glen Myers, software testing has the following objectives:

- The process of investigating and checking a program to find whether there is an error or not and does it fulfill the requirements or not is called testing.
- When the number of errors found during the testing is high, it indicates that the testing was good and is a sign of good test case.
- Finding an unknown error that wasn't discovered yet is a sign of a successful and a good test case.

The main objective of software testing is to design the tests in such a way that it systematically finds different types of errors without taking much time and effort so that less time is required for the development of the software. The overall strategy for testing software includes:





1. **Before testing starts, it's necessary to identify and specify the requirements of the product in a quantifiable manner.** Different characteristics quality of the software is there such as maintainability that means the ability to update and modify, the probability that means to find and estimate any risk, and usability that means how it can easily be used by the customers or end-users. All these characteristic qualities should be specified in a particular order to obtain clear test results without any error.
2. **Specifying the objectives of testing in a clear and detailed manner.** Several objectives of testing are there such as effectiveness that means how effectively the software can achieve the target, any failure that means inability to fulfill the requirements and perform functions, and the cost of defects or errors that mean the cost required to fix the error. All these objectives should be clearly mentioned in the test plan.
3. **For the software, identifying the user's category and developing a profile for each user.** Use cases describe the interactions and communication among different classes of users and the system to achieve the target. So as to identify the actual requirement of the users and then testing the actual use of the product.
4. **Developing a test plan to give value and focus on rapid-cycle testing.** Rapid Cycle Testing is a type of test that improves quality by identifying and measuring the any changes that need to be required for improving the process of software. Therefore, a test plan is an important and effective document that helps the tester to perform rapid cycle testing.
5. **Robust software is developed that is designed to test itself.** The software should be capable of detecting or identifying different classes of errors. Moreover,

software design should allow automated and regression testing which tests the software to find out if there is any adverse or side effect on the features of software due to any change in code or program.

6. **Before testing, using effective formal reviews as a filter.** Formal technical reviews is technique to identify the errors that are not discovered yet. The effective technical reviews conducted before testing reduces a significant amount of testing efforts and time duration required for testing software so that the overall development time of software is reduced.
7. **Conduct formal technical reviews to evaluate the nature, quality or ability of the test strategy and test cases.** The formal technical review helps in detecting any unfilled gap in the testing approach. Hence, it is necessary to evaluate the ability and quality of the test strategy and test cases by technical reviewers to improve the quality of software.
8. **For the testing process, developing a approach for the continuous development.** As a part of a statistical process control approach, a test strategy that is already measured should be used for software testing to measure and control the quality during the development of software.

## Advantages or Disadvantages:

### Advantages of software testing:

1. Improves software quality and reliability – Testing helps to identify and fix defects early in the development process, reducing the risk of failure or unexpected behavior in the final product.
2. Enhances user experience – Testing helps to identify usability issues and improve the overall user experience.
3. Increases confidence – By testing the software, developers and stakeholders can have confidence that the software meets the requirements and works as intended.
4. Facilitates maintenance – By identifying and fixing defects early, testing makes it easier to maintain and update the software.
5. Reduces costs – Finding and fixing defects early in the development process is less expensive than fixing them later in the life cycle.

### Disadvantages of software testing:

1. Time-consuming – Testing can take a significant amount of time, particularly if thorough testing is performed.

2. Resource-intensive – Testing requires specialized skills and resources, which can be expensive.
3. Limited coverage – Testing can only reveal defects that are present in the test cases, and it is possible for defects to be missed.
4. Unpredictable results – The outcome of testing is not always predictable, and defects can be hard to replicate and fix.
5. Delays in delivery – Testing can delay the delivery of the software if testing takes longer than expected or if significant defects are identified.

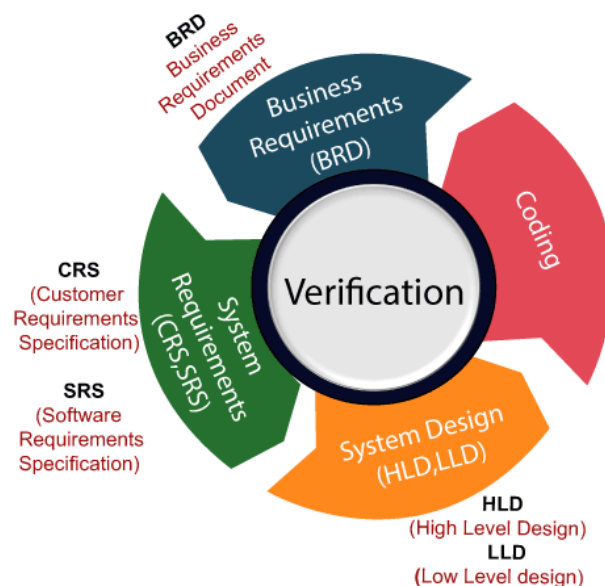
## Verification and Validation Testing

In this section, we will learn about verification and validation testing and their major differences.

### Verification testing

Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.

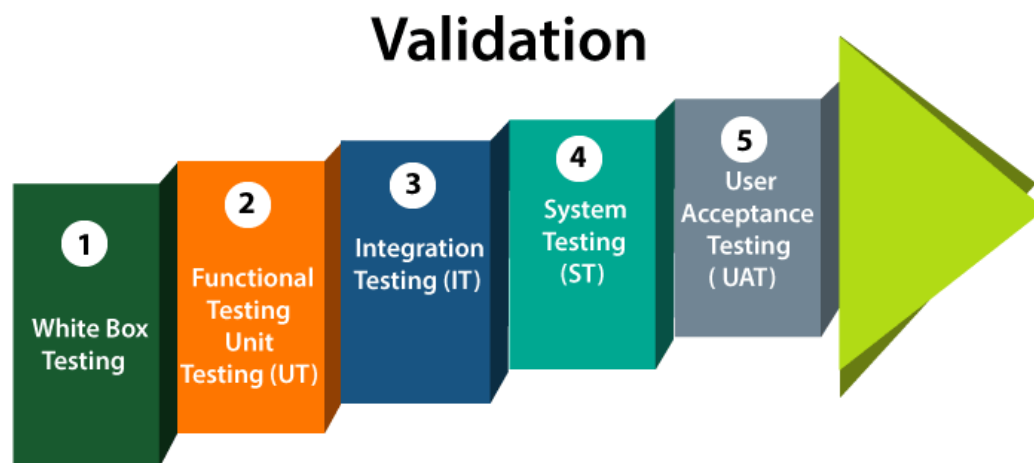
It is also known as static testing, where we are ensuring that "**we are developing the right product or not**". And it also checks that the developed application fulfilling all the requirements given by the client.



## Validation testing

Validation testing is testing where tester performed functional and non-functional testing. Here **functional testing** includes **Unit Testing** (UT), **Integration Testing** (IT) and **System Testing** (ST), and **non-functional** testing includes **User acceptance testing** (UAT).

Validation testing is also known as dynamic testing, where we are ensuring that "**we have developed the product right.**" And it also checks that the software meets the business needs of the client.



*Note: Verification and Validation process are done under the V model of the software development life cycle.*

## Difference between verification and validation testing

Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as <b>static testing</b> .	Validation is also known as <b>dynamic testing</b> .
Verification includes different methods like Inspections, Reviews, and Walkthroughs.	Validation includes testing like <b>functional testing</b> , system testing, <b>integration</b> , and User acceptance testing.

It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.
<b>Quality assurance</b> comes under verification testing.	<b>Quality control</b> comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.
In this type of testing, we can verify that the inputs follow the outputs or not.	In this type of testing, we can validate that the user accepts the product or not.

## System Testing

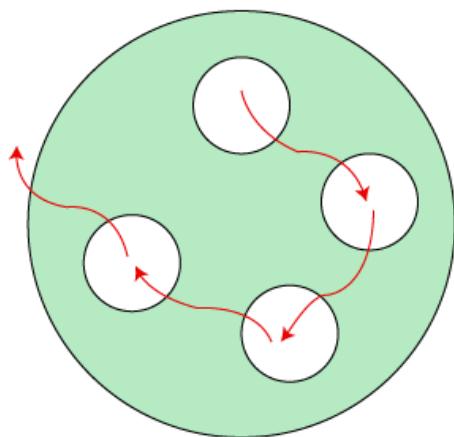
System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible

hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.



To check the end-to-end flow of an application or the software as a user is known as **System testing**. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

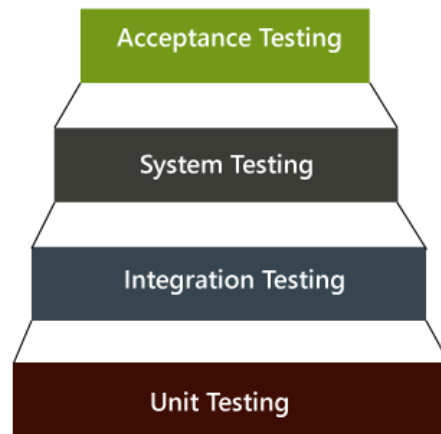
It is **end-to-end testing** where the testing environment is similar to the production environment.



There are four levels of software testing: unit testing, integration testing, system testing and acceptance testing, all are used for the testing purpose. Unit Testing used to test a

single software; Integration Testing used to test a group of units of software, System Testing used to test a whole system and Acceptance Testing used to test the acceptability of business requirements. Here we are discussing system testing which is the third level of testing levels.

## Hierarchy of Testing Levels



There are mainly two widely used methods for software testing, one is **White box testing** which uses internal coding to design test cases and another is black box testing which uses GUI or user perspective to develop test cases.

- White box testing
- Black box testing

**System testing falls under Black box testing** as it includes testing of the external working of the software. Testing follows user's perspective to identify minor defects.

System Testing includes the following steps.

- Verification of input functions of the application to test whether it is producing the expected output or not.
- Testing of integrated software by including external peripherals to check the interaction of various components with each other.
- Testing of the whole system for End to End testing.
- Behavior testing of the application via a user's experience

## Example of System testing

Suppose we open an application, let say **www.rediff.com**, and there we can see that an advertisement is displayed on the top of the homepage, and it remains there for a few seconds before it disappears. These types of Ads are done by the Advertisement Management System (AMS). Now, we will perform system testing for this type of field.

The below application works in the following manner:

- Let's say that Amazon wants to display a promotion ad on January 26 at precisely 10:00 AM on the Rediff's home page for the country India.
- Then, the sales manager logs into the website and creates a request for an advertisement dated for the above day.
- He/she attaches a file that likely an image files or the video file of the AD and applies.
- The next day, the AMS manager of Rediffmail login into the application and verifies the awaiting Ad request.
- The AMS manager will check those Amazons ad requests are pending, and then he/she will check if the space is available for the particular date and time.
- If space is there, then he/she evaluate the cost of putting up the Ad at 15\$ per second, and the overall Ad cost for 10 seconds is approximate 150\$.
- The AMS manager clicks on the payment request and sends the estimated value along with the request for payment to the Amazon manager.
- Then the amazon manager login into the Ad status and confirms the payment request, and he/she makes the payment as per all the details and clicks on the **Submit** and **Pay**
- As soon as Rediff's AMs manager gets the amount, he/she will set up the Advertisement for the specific date and time on the Rediffmail's home page.

The various system test scenarios are as follows:

**Scenario1:** The first test is the general scenario, as we discussed above. The test engineer will do the system testing for the underlying situation where the Amazon manager creates a request for the Ad and that Ad is used at a particular date and time.

**Scenario2:** Suppose the Amazon manager feels that the AD space is too expensive and cancels the request. At the same time, the Flipkart requests the Ad space on January 26



at 10:00 AM. Then the request of Amazon has been canceled. Therefore, Flipkart's promotion ad must be arranged on January 26 at 10 AM.

After all, the request and payment have been made. Now, if Amazon changes their mind and they feel that they are ready to make payment for January 26 at 10 AM, which should be given because Flipkart has already used that space. Hence, another calendar must open up for Amazon to make their booking.

**Scenario3:** in this, first, we login as AMS manger, then click on Set Price page and set the price for AD space on logout page to 10\$ per second.

Then login as Amazon manager and select the date and time to put up and Ad on the logout page. And the payment should be 100\$ for 10 seconds of an Ad on Rediffmail logout page.

Name	Status	Total Amount
Amazon	Available	150\$
Flipkart	Available	100\$
.....		
.....		

Credit Card No.

.....

.....

.....

REQUEST FOR ADVERTISEMENT

Product

Country 

China  
Russia  
U.S.A

Page 

Sent Items  
Compose Mail  
Logout

Duration

Date

Attach Advertisement  
[Text,Audio,Image,Video]

PENDING ADVERTISEMENT

NAME	STATUS	SELECT
.....	.....	<input type="checkbox"/>
.....	.....	<input type="checkbox"/>
.....	.....	<input type="checkbox"/>
.....	.....	<input type="checkbox"/>
.....	.....	<input type="checkbox"/>

Pending Ads

Set Rates

Amazon Pending

Flipkart Pending

SET PRICE

Country

Page

Duration

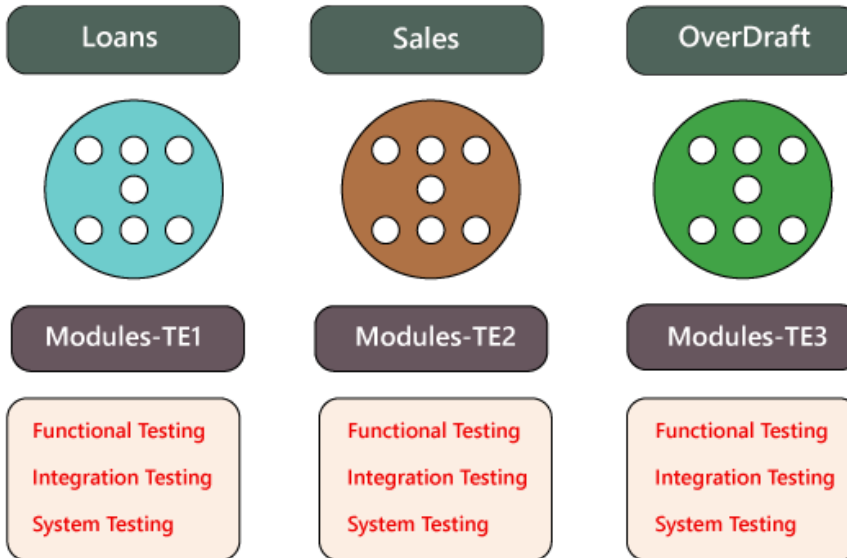
Amount

Amazon Advertisement Available at 10:00AM on January 26th

**Note:** Generally, every test engineer does the functional, integration, and system testing on their assigned module only.

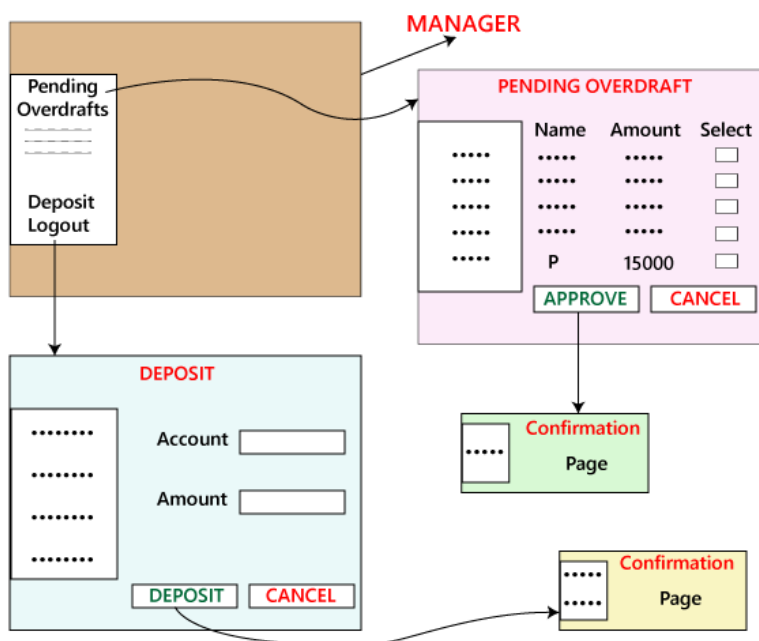
As we can see in the below image, we have three different modules like **Loans, Sales, and Overdraft**. And these modules are going to be tested by their assigned test engineers only because if data flow between these modules or scenarios, then we need to clear that in which module it is going and that test engineer should check that thing.

Let us assume that here we are performing system testing on the interest estimation, where the customer takes the Overdraft for the first time as well as for the second time.



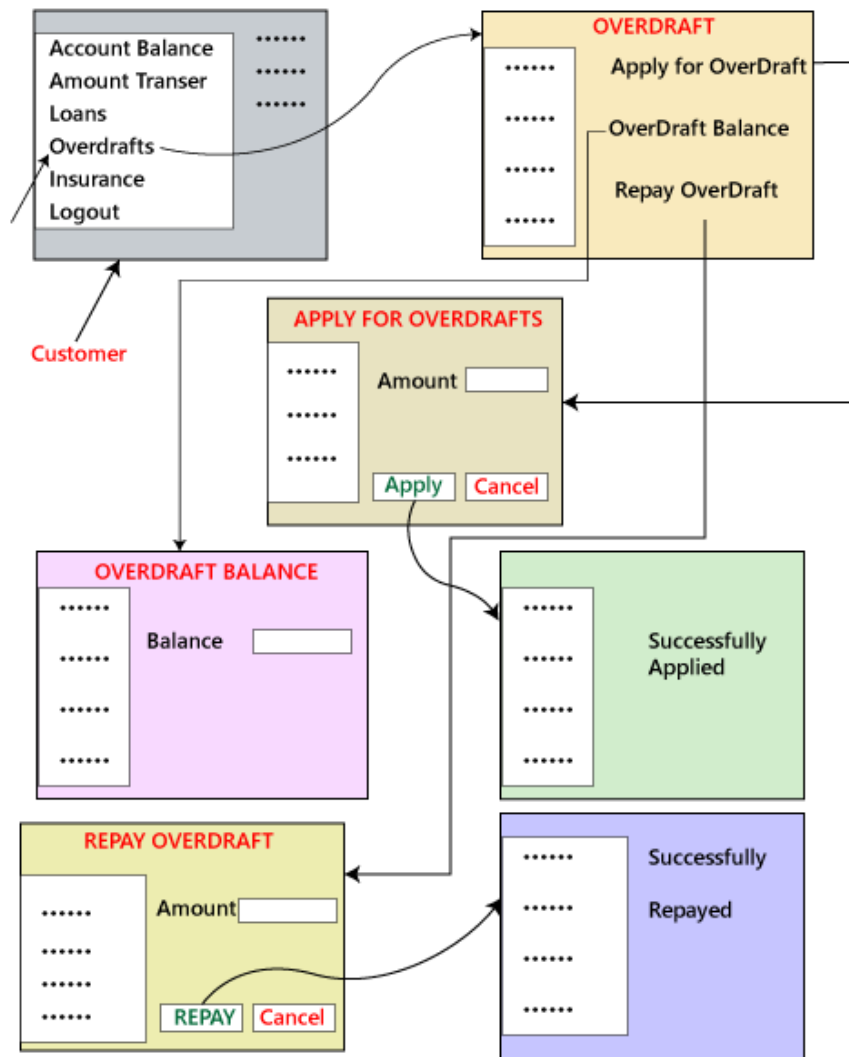
In this particular example, we have the following scenarios:

### Scenarios 1



- First, we will log in as a User; let see P, and apply for Overdraft Rs15000, click on apply, and logout.
- After that, we will log in as a Manager and approve the Overdraft of P, and logout.

- Again we will log in as a P and check the Overdraft balance; Rs15000 should be deposited and logout.
- Modify the server date to the next 30 days.
- Login as P, check the Overdraft balance is  $15000 + 300 + 200 = 15500$ , than logout
- Login as a Manager, click on the Deposit, and Deposit Rs500, logout.
- Login as P, Repay the Overdraft amount, and check the Overdraft balance, which is Rs zero.
- Apply for Overdraft in Advance as a two-month salary.
- Approve by the Manager, amount credit and the interest will be there to the processing fee for 1st time.
- Login user → Homepage [Loan, Sales, Overdraft] → Overdraft page [Amount Overdraft, Apply Overdraft, Repay Overdraft] → Application
- Login manager → Homepage [Loan, Sales, Overdraft] → Overdraft page [Amount Overdraft, Apply Overdraft, Repay Overdraft, Approve Overdraft] → Approve Page → Approve application.
- Login as user P → Homepage [Loan, Sales, Overdraft] → Overdraft page [Amount Overdraft, Apply Overdraft, Repay Overdraft] → Approved Overdraft → Amount Overdraft
- Login as user P → Homepage [Loan, Sales, Overdraft] → Overdraft page [Amount Overdraft, Apply Overdraft, Repay Overdraft] → Repay Overdraft → with process fee + interest amount.



## Scenario 2

Now, we test the alternative scenario where the bank provides an offer, which says that a customer who takes Rs45000 as Overdraft for the first time will not charge for the Process fee. The processing fee will not be refunded when the customer chooses another overdraft for the third time.

We have to test for the third scenario, where the customer takes the Overdraft of Rs45000 for the first time, and also verify that the Overdraft repays balance after applying for another overdraft for the third time.

## Scenario 3

In this, we will reflect that the application is being used generally by all the clients, all of a sudden the bank decided to reduce the processing fee to Rs100 for new customer, and we have test Overdraft for new clients and check whether it is accepting only for Rs100.

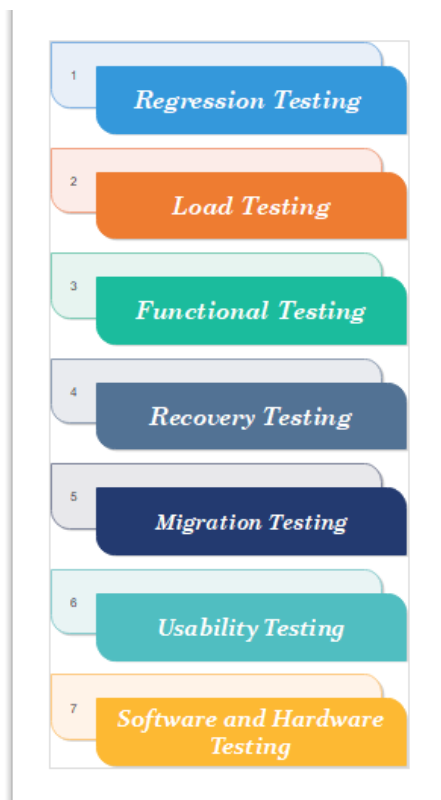
But then we get conflicts in the requirement, assume the client has applied for Rs15000 as Overdraft with the current process fee for Rs200. Before the Manager is yet to approve it, the bank decreases the process fee to Rs100.

Now, we have to test what process fee is charged for the Overdraft for the pending customer. And the testing team cannot assume anything; they need to communicate with the Business Analyst or the Client and find out what they want in those cases.

Therefore, if the customers provide the first set of requirements, we must come up with the maximum possible scenarios.

## Types of System Testing

System testing is divided into more than 50 types, but software testing companies typically uses some of them. These are listed below:



## Regression Testing

Regression testing is performed under system testing to confirm and identify that if there's any defect in the system due to modification in any other part of the system. It makes sure, any changes done during the development process have not introduced a new defect and also gives assurance; old defects will not exist on the addition of new software over the time.

## Load Testing

Load testing is performed under system testing to clarify whether the system can work under real-time loads or not.

## Functional Testing

Functional testing of a system is performed to find if there's any missing function in the system. Tester makes a list of vital functions that should be in the system and can be added during functional testing and should improve quality of the system.

## Recovery Testing

Recovery testing of a system is performed under system testing to confirm reliability, trustworthiness, accountability of the system and all are lying on recouping skills of the system. It should be able to recover from all the possible system crashes successfully.

In this testing, we will test the application to check how well it recovers from the crashes or disasters.

Recovery testing contains the following steps:

- Whenever the software crashes, it should not vanish but should write the **crash log message or the error log message** where the reason for crash should be mentioned. **For example: C://Program Files/QTP/Cresh.log**
- It should kill its own procedure before it vanishes. Like, in Windows, we have the Task Manager to show which process is running.
- We will introduce the bug and crash the application, which means that someone will lead us to how and when will the application crash. Or **By experiences**, after few months of involvement on working the product, we can get to know how and when the application will crash.

- Re-open the application; the application must be reopened with earlier settings.

**For example:** Suppose, we are using the Google Chrome browser, if the power goes off, then we switch on the system and re-open the Google chrome, we get a message asking whether we want to **start a new session** or **restore the previous session**. For any developed product, the developer writes a recovery program that describes, why the software or the application is crashing, whether the crash log messages are written or not, etc.

## Migration Testing

Migration testing is performed to ensure that if the system needs to be modified in new infrastructure so it should be modified without any issue.

## Usability Testing

The purpose of this testing to make sure that the system is well familiar with the user and it meets its objective for what it supposed to do.

## Software and Hardware Testing

This testing of the system intends to check [hardware](#) and [software](#) compatibility. The hardware configuration must be compatible with the software to run it without any issue. Compatibility provides flexibility by providing interactions between hardware and software.

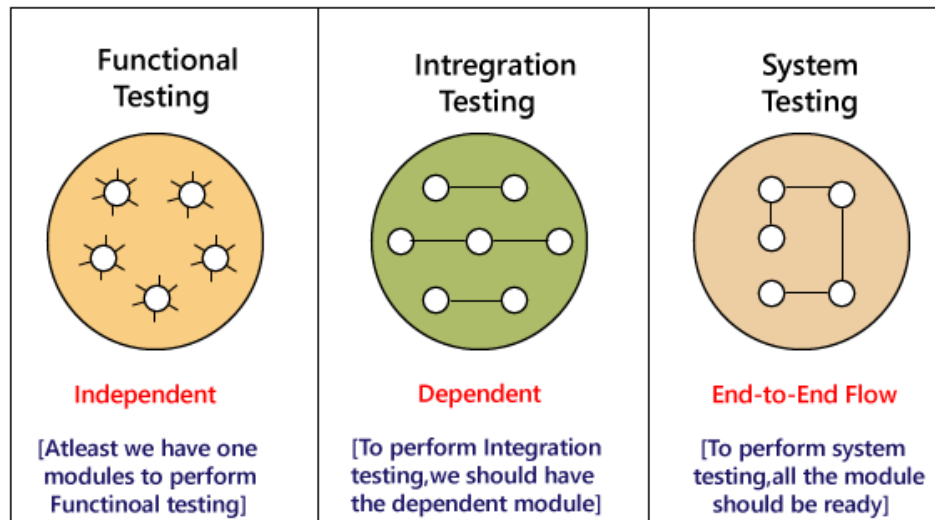
## Why is System Testing Important?

- System Testing gives hundred percent assurance of system performance as it covers end to end function of the system.
- It includes testing of System software architecture and business requirements.
- It helps in mitigating live issues and bugs even after production.
- System testing uses both existing system and a new system to feed same data in both and then compare the differences in functionalities of added and existing functions so, the user can understand benefits of new added functions of the system.

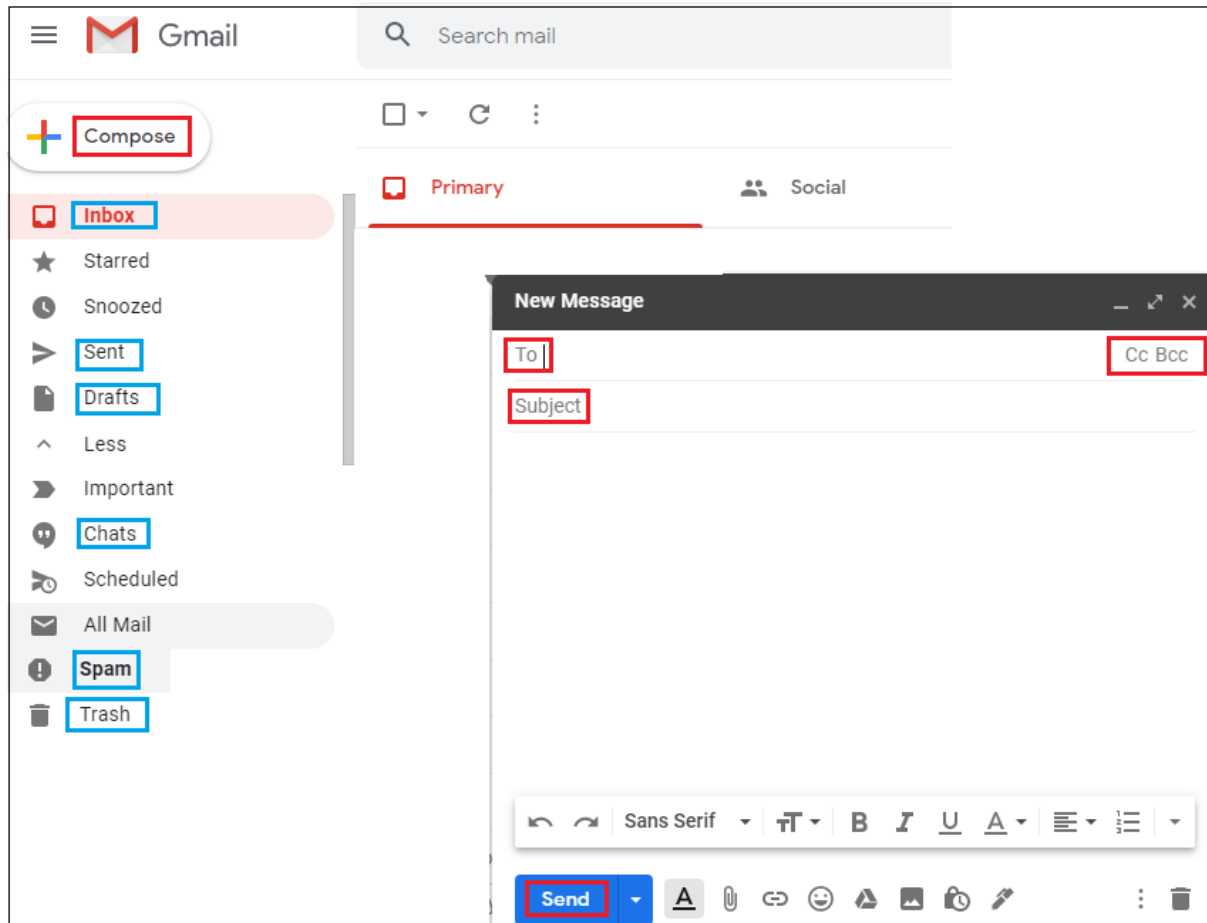


# Testing Any Application

Here, we are going to test the **Gmail** application to understand how **functional, integration, and System testing** works.



Suppose, we have to test the various modules such as **Login, Compose, Draft, Inbox, Sent Item, Spam, Chat, Help, Logout** of **Gmail** application.



We do Functional Testing on all Modules First, and then only we can perform integration testing and system testing.

In functional testing, at least we have one module to perform functional testing. So here we have the Compose Module where we are performing the functional testing.

## Compose

The different components of the Compose module are **To**, **CC**, **BCC**, **Subject**, **Attachment**, **Body**, **Sent**, **Save to Draft**, **Close**.

- First, we will do functional testing on the **To**

**Input**

**Results**

<b>Positive inputs</b>	
mike@gmail.com	Accept
Mike12@gmail.com	Accept
Mike@yahoo.com	Accept
<b>Negative inputs</b>	
Mike@yahoocom	Error
Mike@yaho.com	Error

- For **CC** & **BCC** components, we will take the same input as **To component**.
- For **Subject** component, we will take the following inputs and scenarios:

Input	Results
<b>Positive inputs</b>	
Enter maximum character	Accept
Enter Minimum character	Accept
Blank Space	Accept
URL	Accept

Copy & Paste	Accept
<b>Negative inputs</b>	
Crossed maximum digits	Error
Paste images / video / audio	Error

- **Maximum character**
- **Minimum character**
- **Flash files (GIF)**
- **Smiles**
- **Format**
- **Blank**
- **Copy & Paste**
- **Hyperlink**
- **Signature**
- For the **Attachment** component, we will take the help of the below scenarios and test the component.
  - **File size at maximum**
  - **Different file formats**
  - **Total No. of files**
  - **Attach multiple files at the same time**
  - **Drag & Drop**
  - **No Attachment**
  - **Delete Attachment**
  - **Cancel Uploading**
  - **View Attachment**
  - **Browser different locations**
  - **Attach opened files**

- For **Sent** component, we will write the entire field and click on the **Sent** button, and the Confirmation message; **Message sent successfully** must be displayed.
- For **Saved to Drafts** component, we will write the entire field and click on **aved to drafts**, and the Confirmation message must be displayed.
- For the **Cancel** component, we will write all fields and click on the Cancel button, and the **Window will be closed** or moved to **save to draft** or all fields must be refreshed.

Once we are done performing functional testing on compose module, we will do the Integration testing on Gmail application's various modules:

### Login

- First, we will enter the username and password for login to the application and Check the username on the Homepage.

### Compose

- Compose mail, send it and check the mail in Sent Item [sender]
- Compose mail, send it and check the mail in the receiver [Inbox]
- Compose mail, send it and check the mail in self [Inbox]
- Compose mail, click on Save as Draft, and check-in sender draft.
- Compose mail, send it invalid id (valid format), and check for undelivered message.
- Compose mail, close and check-in Drafts.

### Inbox

- Select the mail, reply, and check in sent items or receiver Inbox.
- Select the mail in Inbox for reply, Save as Draft and check in the Draft.
- Select the mail then delete it, and check in Trash.

### Sent Item

- Select the mail, Sent Item, Reply or Forward, and check in Sent item or receiver inbox.
- Select mail, Sent Item, Reply or Forward, Save as Draft, and verify in the Draft.
- Select mail, delete it, and check in the Trash.

## Draft

- Select the email draft, forward and check Sent item or Inbox.
- Select the email draft, delete and verify in Trash.

## Chat

- Chat with offline users saved in the inbox of the receiver.
- Chat with the user and verify it in the chat window.
- Chat with a user and check in the chat history.

***Note: During testing, we need to wait for a particular duration of time because system testing can only be performed when all the modules are ready and performed functional and integration testing.***