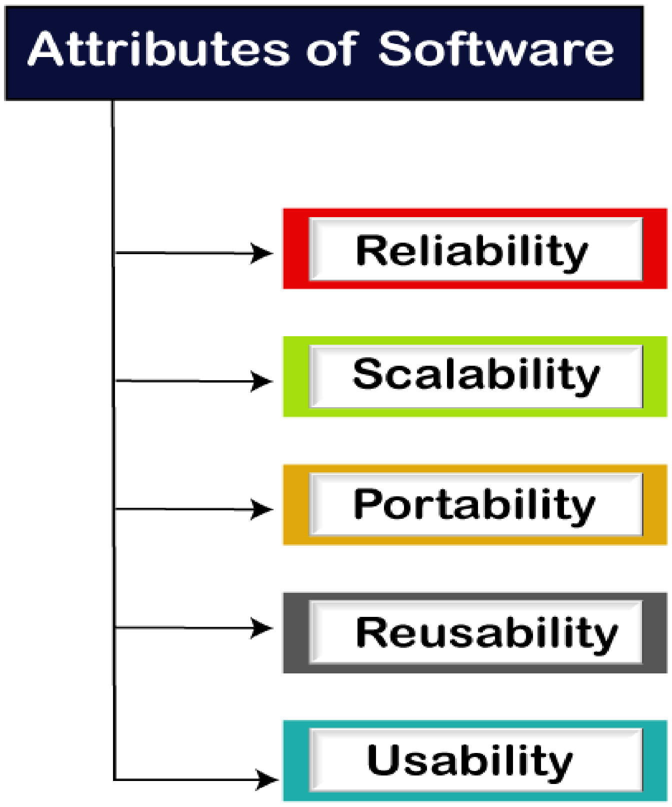
What is Software Testing

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.



Software testing provides an independent view and objective of the software and gives surety of fitness of the software. It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not. The process is also providing the client with information about the quality of the software.

Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.

What is Testing

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defect of application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing furnishes comparison that compares the behavior and state of software against mechanisms because the problem can be recognized by the mechanism. The mechanism may include past versions of the same specified product, comparable products, and interfaces of expected purpose, relevant standards, or other criteria but not limited up to these.

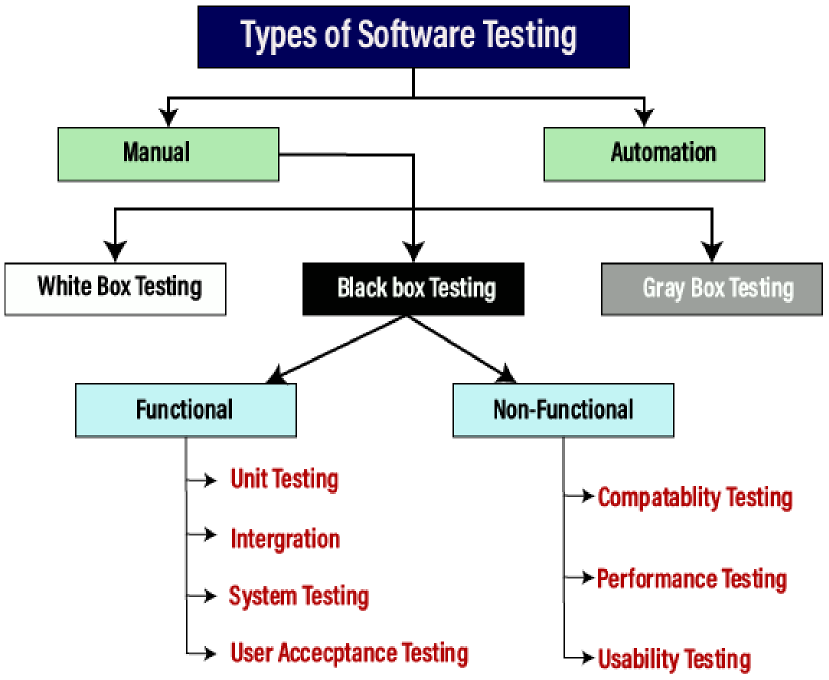
Testing includes an examination of code and also the execution of code in various environments, conditions as well as all the examining aspects of the code. In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

The success of software depends upon acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc. For example, the audience of banking is totally different from the audience of a video game. Therefore, when an organization develops a software product, it can assess whether the software product will be beneficial to its purchasers and other audience.

Type of Software testing

We have various types of testing available in the market, which are used to test the application or the software.

With the help of below image, we can easily understand the type of software testing:



Manual testing

The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

Manual testing can be further divided into three types of testing, which are as follows:

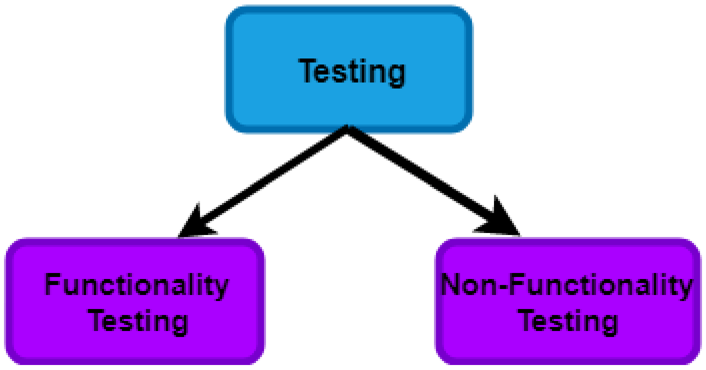
- **White box testing**
- **Black box testing**
- **Gray box testing**

Automation testing

Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing. With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

Software testing also defines as verification of application under test (AUT).

There are two types of testing:



Functional Testing:

It is a type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification. In functional testing, each function tested by giving the value, determining the output, and verifying the actual output with the expected value. Functional testing performed as black-box testing which is presented to confirm that the functionality of an application or system behaves as we are expecting. It is done to verify the functionality of the application.

Functional testing also called as black-box testing, because it focuses on application specification rather than actual code. Tester has to test only the program rather than the system.

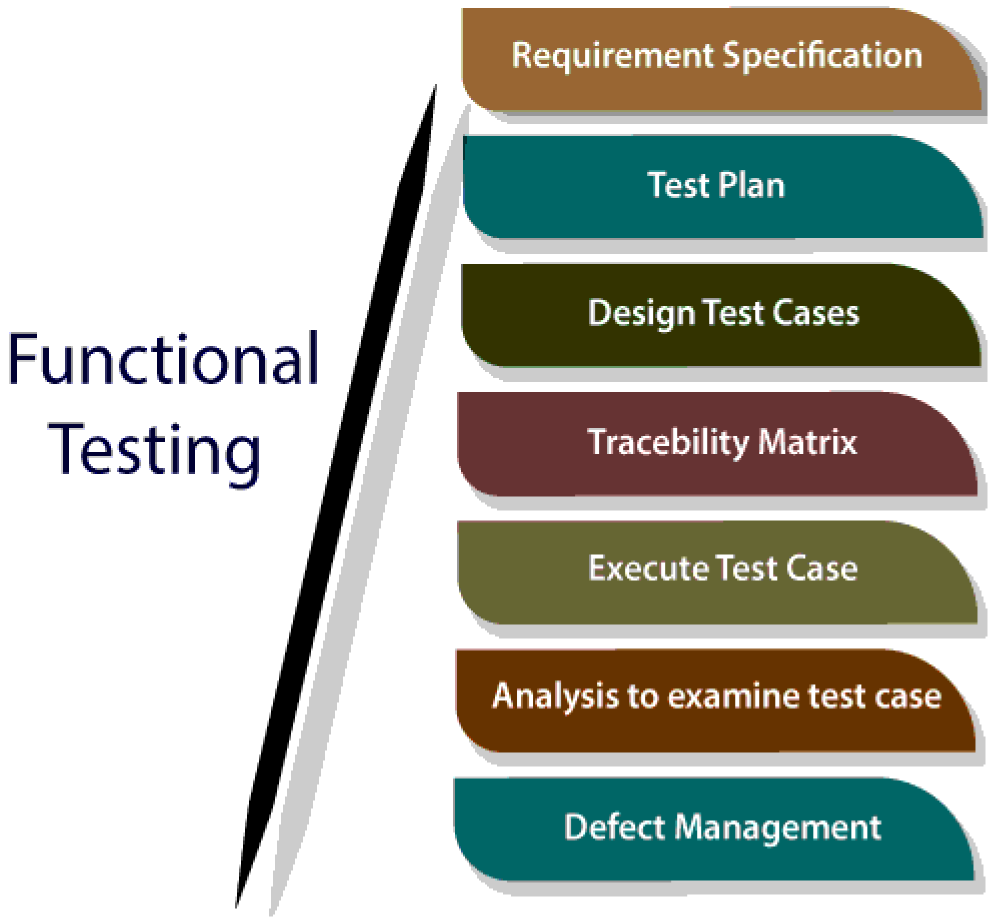
Goal of functional testing

The purpose of the functional testing is to check the primary entry function, necessarily usable function, the flow of screen GUI. Functional testing displays the error message so that the user can easily navigate throughout the application.

What is the process of functional testing?

Testers follow the following steps in the functional testing:

- Tester does verification of the requirement specification in the software application.
- After analysis, the requirement specification tester will make a plan.
- After planning the tests, the tester will design the test case.
- After designing the test, case tester will make a document of the traceability matrix.
- The tester will execute the test case design.
- Analysis of the coverage to examine the covered testing area of the application.
- Defect management should do to manage defect resolving.



What to test in functional testing? Explain

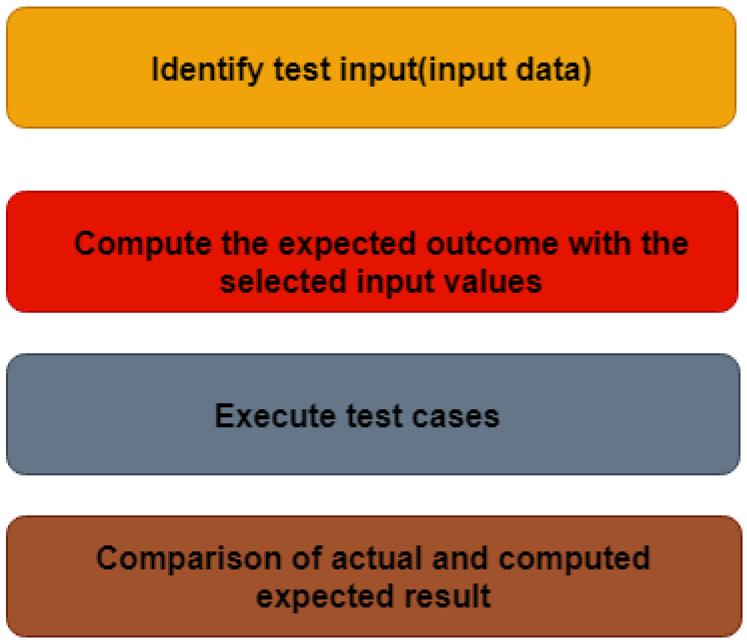
The main objective of functional testing is checking the functionality of the software system. It concentrates on:

- **Basic Usability:** Functional Testing involves the usability testing of the system. It checks whether a user can navigate freely without any difficulty through screens.
- **Accessibility:** Functional testing test the accessibility of the function.
- **Mainline function:** It focuses on testing the main feature.
- **Error Condition:** Functional testing is used to check the error condition. It checks whether the error message displayed.

Explain the complete process to perform functional testing.

There are the following steps to perform functional testing:

- There is a need to understand the software requirement.
- Identify test input data
- Compute the expected outcome with the selected input values.
- Execute test cases
- Comparison between the actual and the computed result

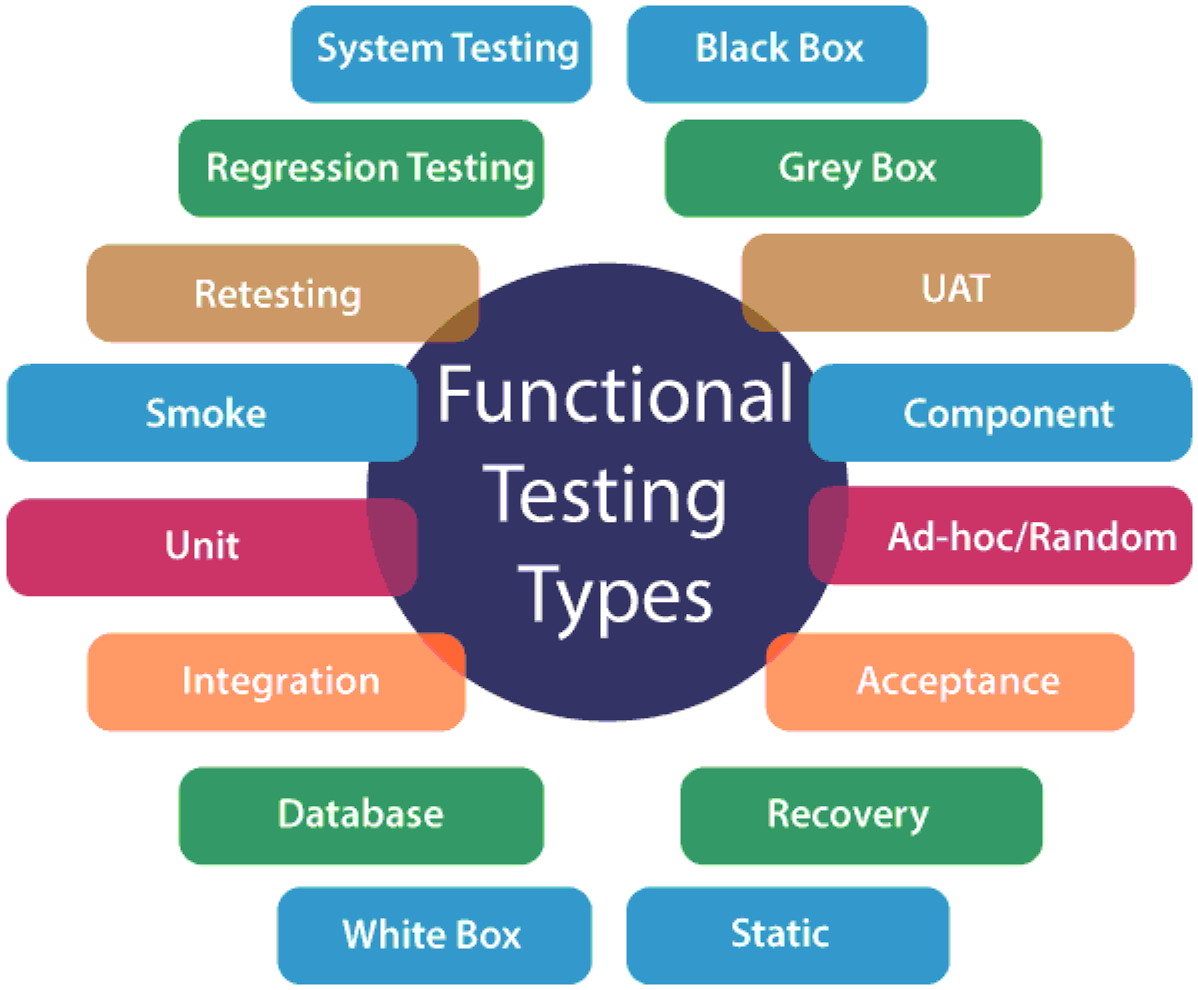

Identify test input(input data)
Compute the expected outcome with the selected input values
Execute test cases
Comparison of actual and computed expected result

Explain the types of functional testing.

The main objective of functional testing is to test the functionality of the component.

Functional testing is divided into multiple parts.

Here are the following types of functional testing.



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.

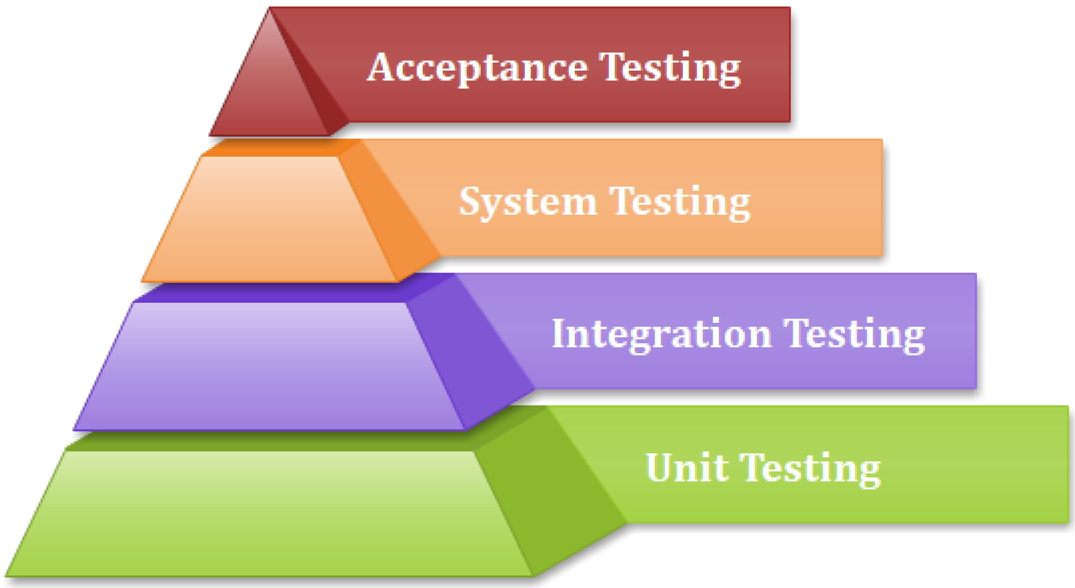
A unit is a single testable part of a software system and tested during the development phase of the application software.

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.

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

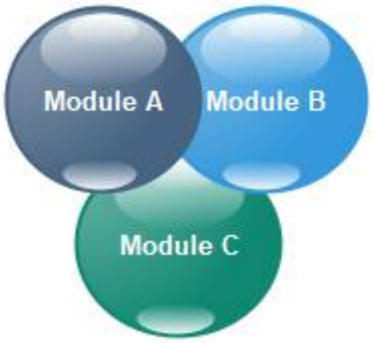
Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components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

Module A
Module B
Module C

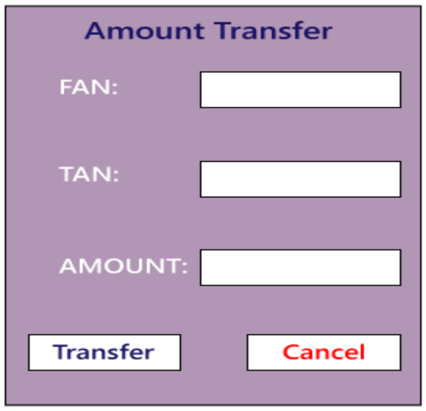
Tested in Unit Testing



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.



- 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.

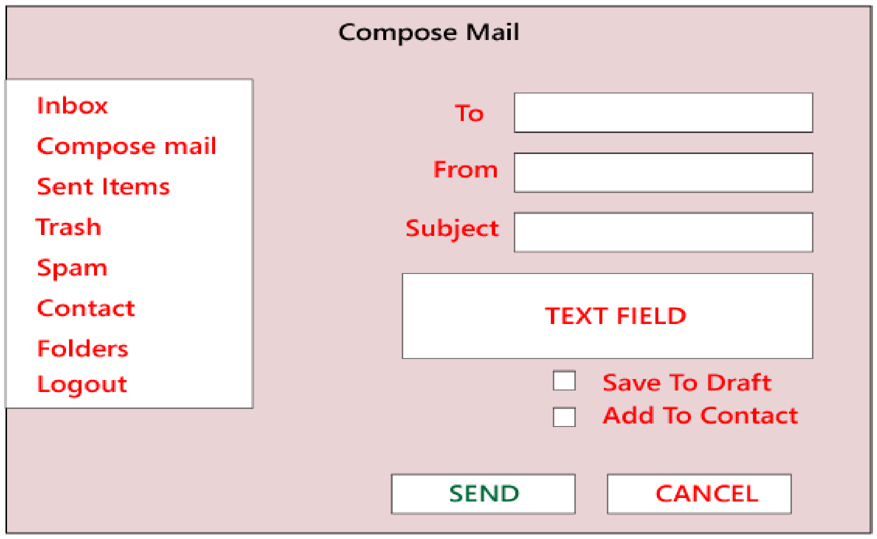
For example: In the Gmail application, the **Source** could be **Compose**, **Data** could be **Email** and the **Destination** could be the **Inbox**.

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:



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**.

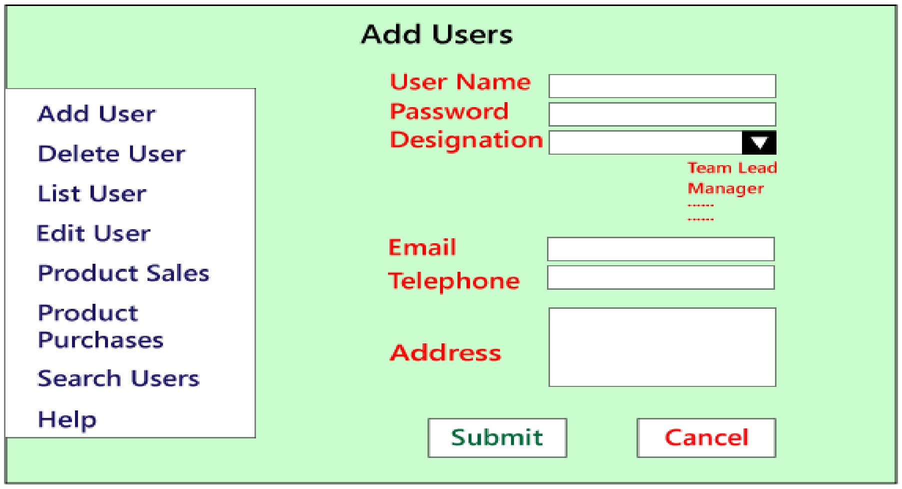
- After that, we send a **mail** to **Q** and verify in the **Send Items** folder of P to check if the send mail is there.
- Now, we will **log out** as P and login as Q and move to the **Inbox** and verify that if the mail has reached.

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**.



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

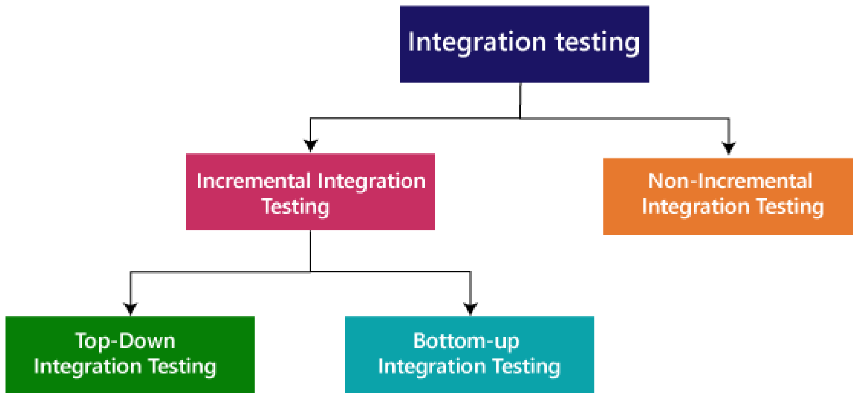
White Box Testing

- 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**

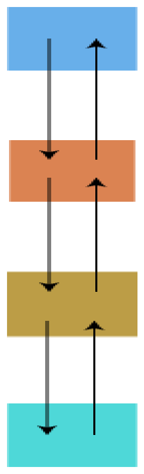

Integration testing
Incremental Integration Testing
Non-Incremental Integration Testing
Top-Down Integration Testing
Bottom-up Integration Testing

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:

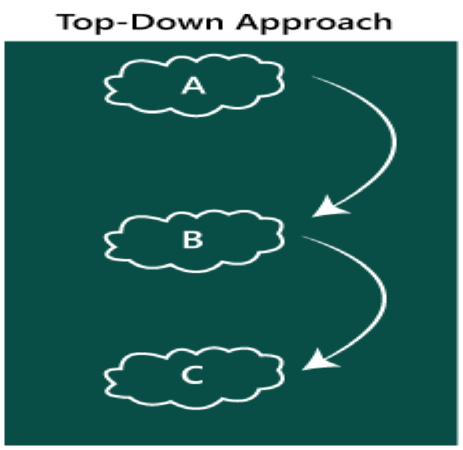
Flipkart→ Login→ Home → Search→ Add cart→Payment → Logout

Incremental integration testing is carried out by further methods:

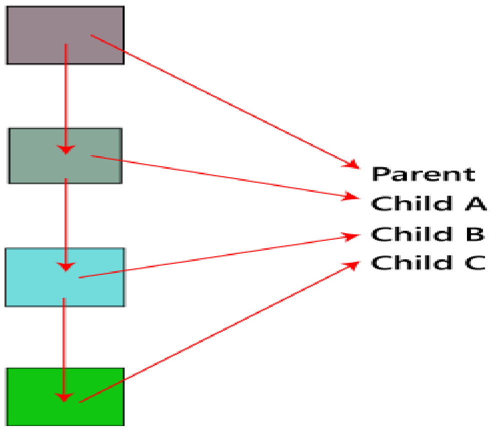
- Top-Down approach
- Bottom-Up approach

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.



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:

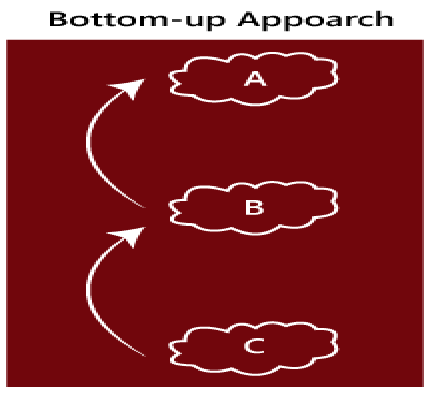
- Identification of defect is difficult.
- An early prototype is possible.

Disadvantages:

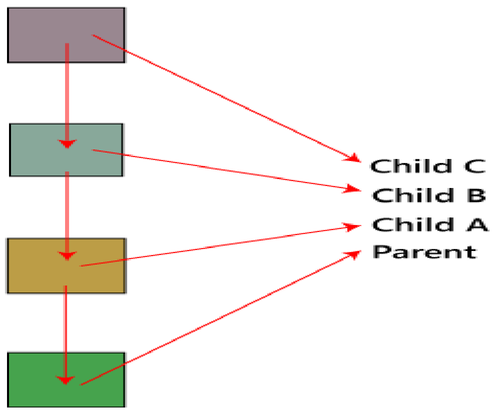
- Due to the high number of stubs, it gets quite complicated.
- Lower level modules are tested inadequately.
- Critical Modules are tested first so that fewer chances of defects.

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.



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.

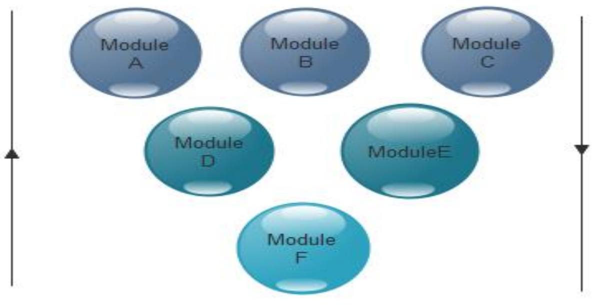
Disadvantages

- 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.



Hybrid Method

Advantages

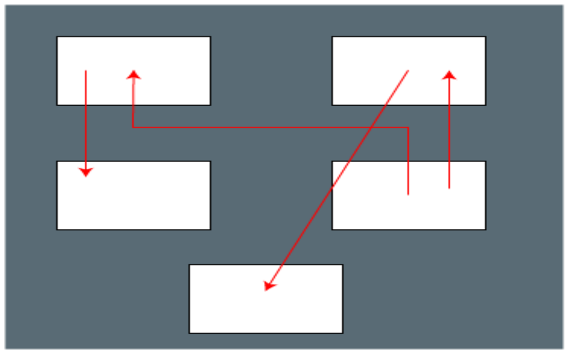
- The hybrid method provides features of both Bottom Up and Top Down methods.
- It is most time reducing method.
- It provides complete testing of all modules.

Disadvantages

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

Non- incremental integration testing

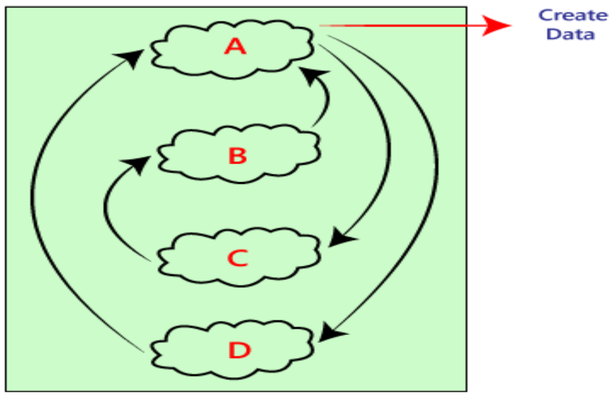
We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. Hence, it is also known as the **Big bang method**.



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.

Let us see examples for our better understanding of the non-incremental integrating testing or big bang method:

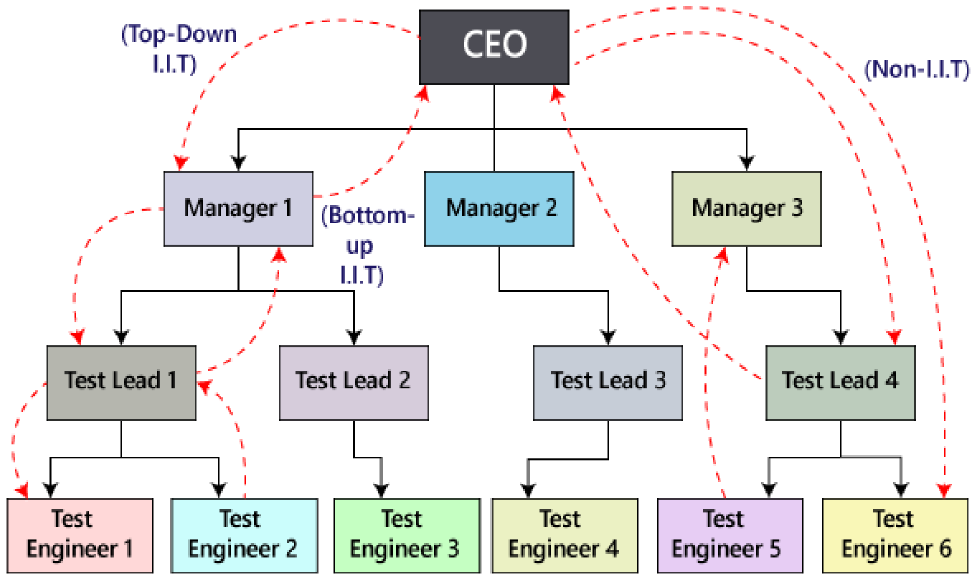
Example1

In the below example, the development team develops the application and sends it to the CEO of the testing team. Then the CEO will log in to the application and generate the username and password and send a mail to the manager. After that, the CEO will tell them to start testing the application.

Then the manager manages the username and the password and produces a username and password and sends it to the **test leads**. And the **test leads** will send it to the **test engineers** for further testing purposes. This order from the CEO to the test engineer is **top-down incremental integrating testing.**

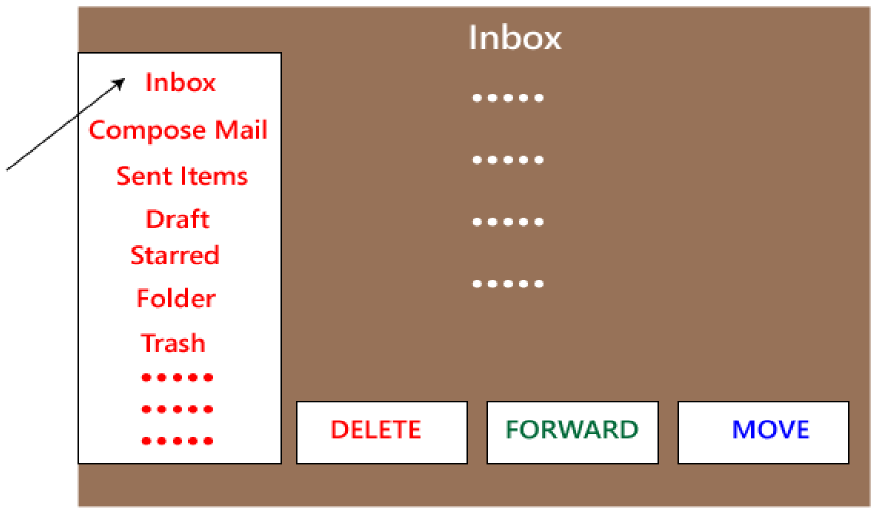
In the same way, when the test engineers are done with testing, they send a report to the **test leads**, who then submit a report to the **manager**, and the manager will send a report to the **CEO**. This process is known as **Bottom-up incremental integration testing** as we can see in the below image:

Note: The combination incremental integration testing (I.I.T) and non-incremental integration testing is known as sandwich testing.



Example2

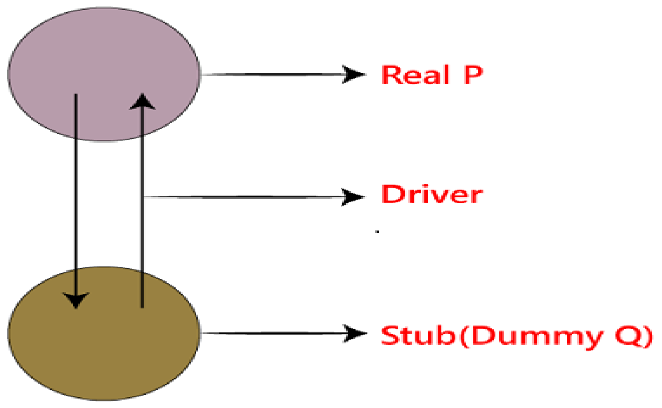
The below example demonstrates a home page of **Gmail's Inbox**, where we click on the **Inbox** link, and we are moved to the inbox page. Here we have to do **non-incremental integration testing** because there is no parent and child concept.



Note

Stub and driver

The **stub** is a dummy module that receives the data and creates lots of probable data, but it performs like a real module. When a data is sent from module P to Stub Q, it receives the data without confirming and validating it, and produce the estimated outcome for the given data.



The function of a driver is used to verify the data from P and sends it to stub and also checks the expected data from the stub and sends it to P.

The **driver** is one that sets up the test environments and also takes care of the communication, evaluates results, and sends the reports. We never use the stub and driver in the testing process.

In **White box testing**

, bottom-up integration testing is ideal because writing drivers is accessible. And in **black box testing**

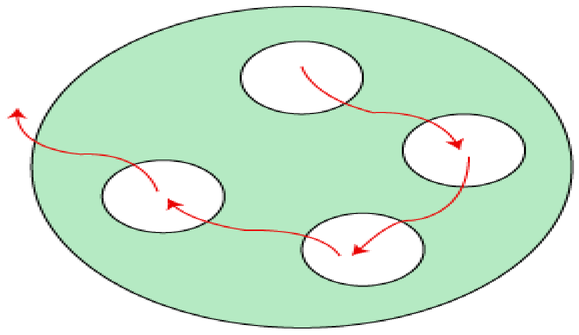
, no preference is given to any testing as it depends on the application.

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.

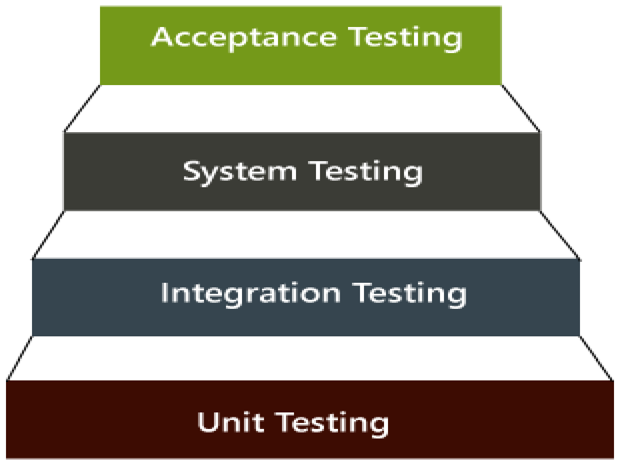
To check the end-to-end flow of an application or the software as a user is known as **System testing**. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

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 auser'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:

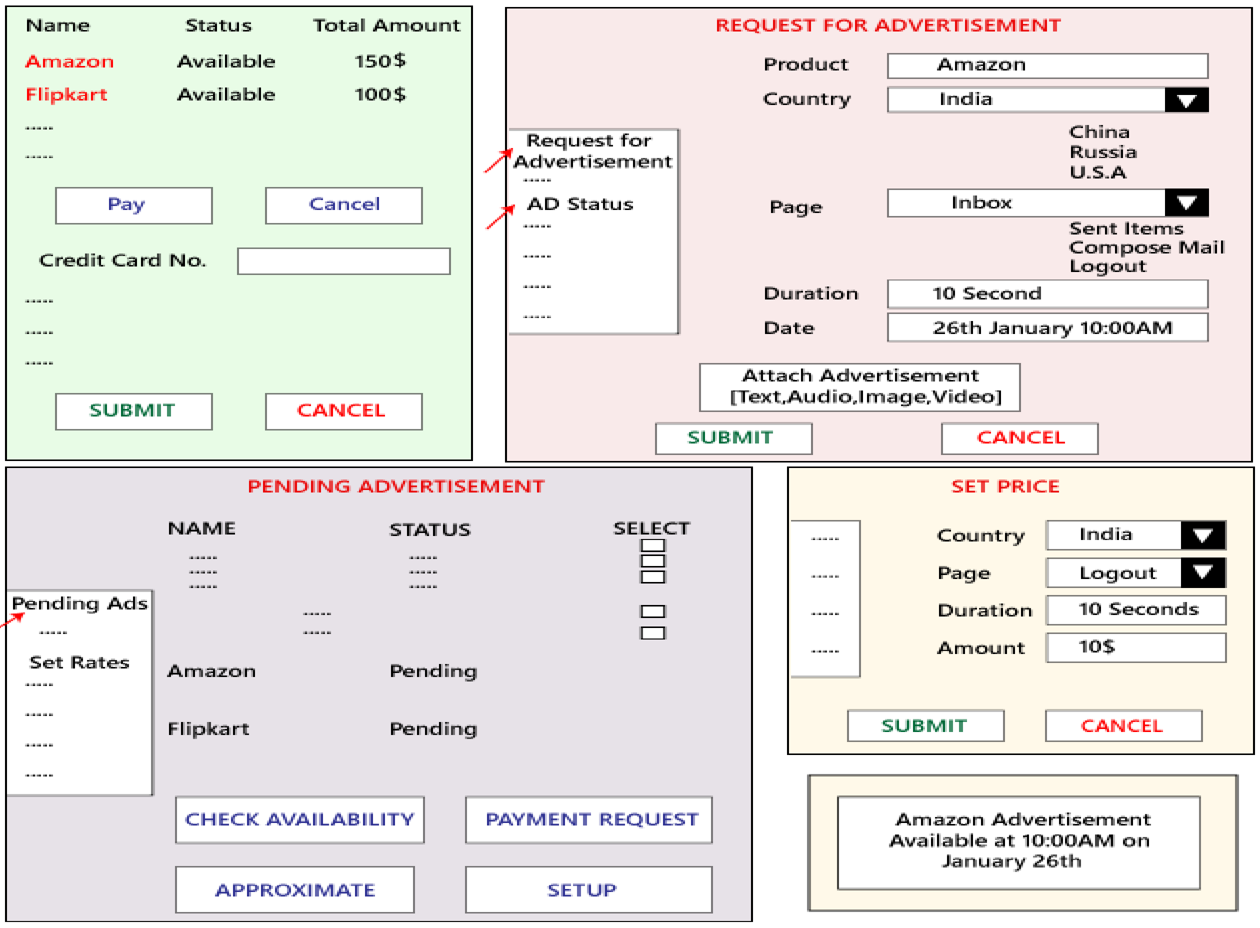
Scenario1: The first test is the general scenario, as we discussed above. The test engineer will do the system testing for the underlying situation where the Amazon manager creates a request for the Ad and that Ad is used at a particular date and time.

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	Amazon
Country	India (China, Russia, U.S.A)
Page	Inbox (Sent Items, Compose Mail, Logout)
Duration	10 Second
Date	26th January 10:00AM

PENDING ADVERTISEMENT

NAME	STATUS	SELECT
Amazon	Pending	
Flipkart	Pending	

SET PRICE

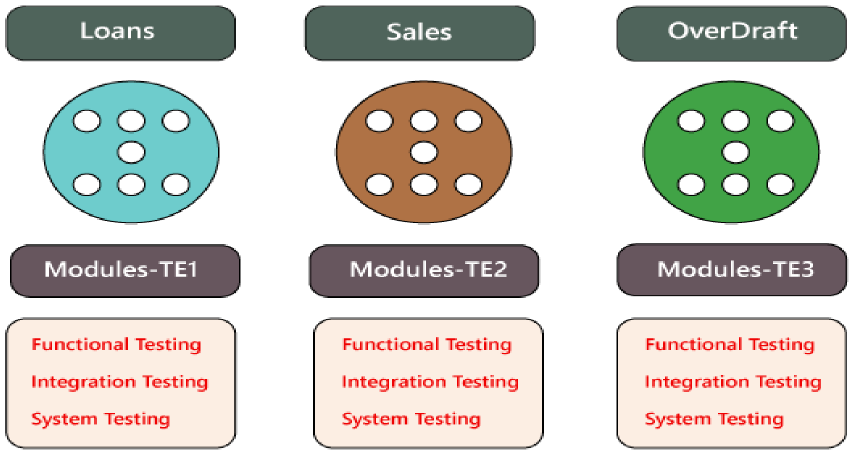
Country	India
Page	Logout
Duration	10 Seconds
Amount	10$

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.

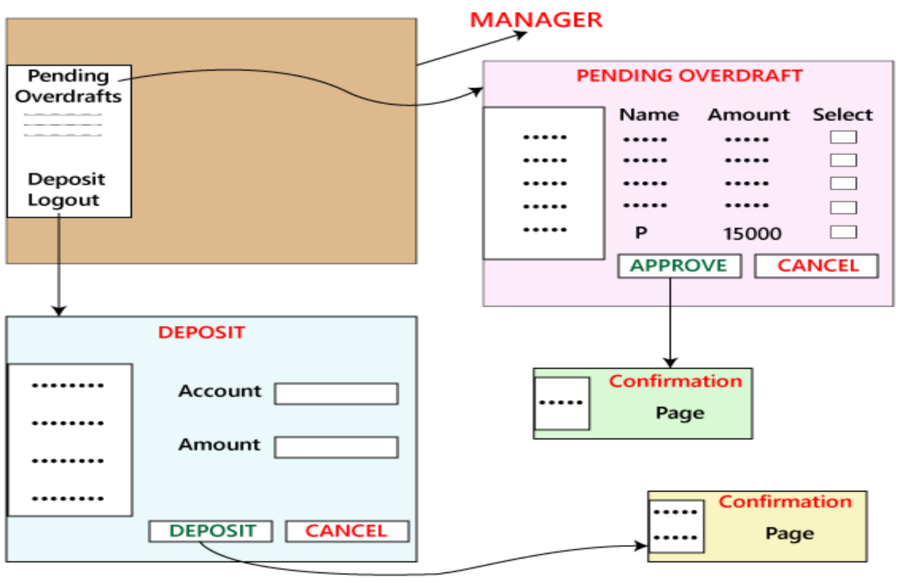
As we can see in the below image, we have three different modules like **Loans, Sales, and Overdraft**. And these modules are going to be tested by their assigned test engineers only because if data flow between these modules or scenarios, then we need to clear that in which module it is going and that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



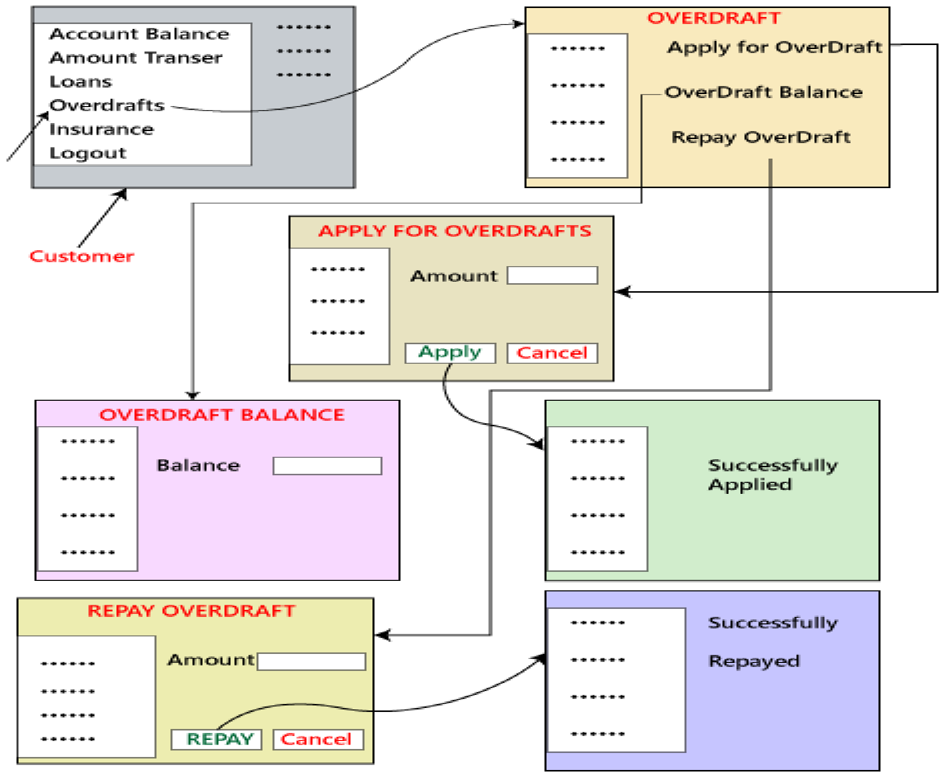
In this particular example, we have the following scenarios:

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.

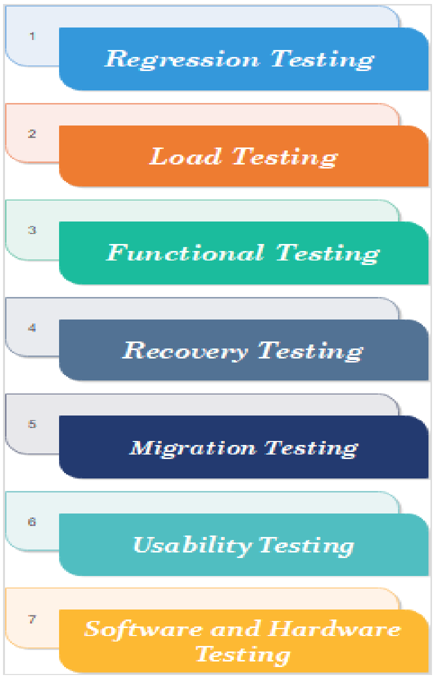
But then we get conflicts in the requirement, assume the client has applied for Rs15000 as Overdraft with the current process fee for Rs200. Before the Manager is yet to approve it, the bank decreases the process fee to Rs100.

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:


1
Regression Testing
2
Load Testing
3
Functional Testing
4
Recovery Testing
5
Migration Testing
6
Usability Testing
7
Software and Hardware Testing