Unit V

Testing Techniques /Software Testing Fundamentals:

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability.

It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:
1. **Verification:** it refers to the set of tasks that ensure that the software correctly implements a specific function.
2. **Validation:** it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Verification: "Are we building the product right?"
Validation: "Are we building the right product?"

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
- Validation emphasizes on user requirements.

Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
- Verifications concentrates on the design and system specifications.

Target of the test are -

- **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
- **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

What are different types of software testing?

Software Testing can be broadly classified into two types:

1. **Manual Testing:** Manual testing includes testing software manually, i.e., without using any automation tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.
Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

2. **Automation Testing:** Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.
Apart from regression testing, automation testing is also used to test the application from a load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money when compared to manual testing.

What are the different techniques of Software Testing Techniques ?

Software testing techniques can be majorly classified into two categories:

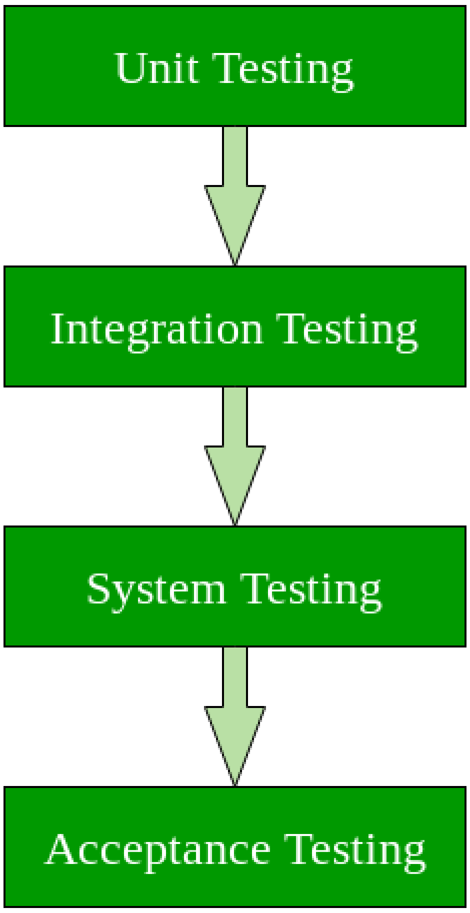
1. **Black Box Testing:** The technique of testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without any concern with the internal logical structure of the software is known as black-box testing.
2. **White-Box Testing:** The technique of testing in which the tester is aware of the internal workings of the product, has access to its source code, and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.

Black Box Testing	**White Box Testing**
Internal workings of an application are not required.	Knowledge of the internal workings is a must.

Black Box Testing	White Box Testing
Also known as closed box/data-driven testing.	Also known as clear box/structural testing.
End users, testers, and developers.	Normally done by testers and developers.
This can only be done by a trial and error method.	Data domains and internal boundaries can be better tested.

What are different levels of software testing?
Software level testing can be majorly classified into 4 levels:

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



Software Testing Fundamentals

Testability

Software testability is simply how easily a computer program can be tested.The following characteristics lead to testable software:

1. **Operability:** The better it works, the more efficient it can be tested.High quality software can be tested in a better manner. This is because if the software is designed and implemented considering quality, then comparatively fewer errors will be detected during the execution of tests.
2. **Observability:** What you see is what you test. Testers can easily identify whether the output generated for certain input is accurate simply by observing it.
3. **Controllability:** The better we can control the software, the more the testing can be automated and optimized. Software and Hardware states and Variables can be controlled directly by the test engineer.
4. **Decomposability:** By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting.By breaking

software into independent modules, problems can be easily isolated and the modules can be easily tested.

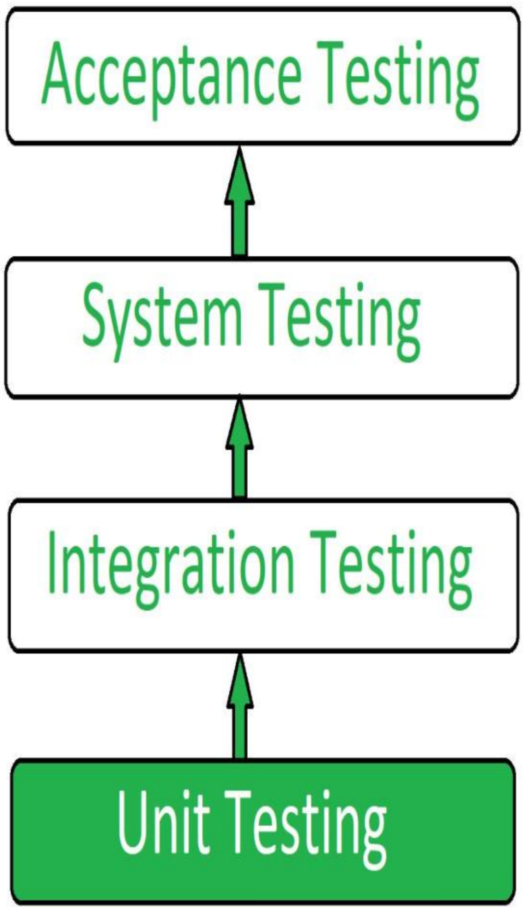
5. **Simplicity:** The less there is to test, the more quickly we can test it.
6. **Stability:** The fewer the changes, the fewer the disruptions to testing
7. **Understandability:** The more information we have, the smarter we will test.Software that is easy to understand can be tested in an efficient manner. Software can be properly understood by gathering maximum information about it. For example, to have a proper knowledge of the software, its documentation can be used, which provides complete information of the software code thereby increasing its clarity and making the testing easier.

Unit Testing

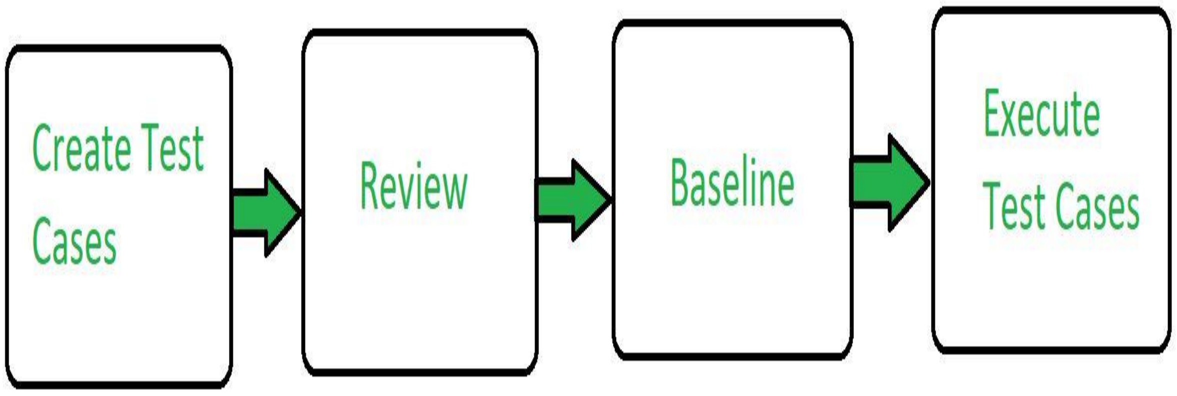
Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent modules are tested to determine if there are any issue by the developer himself. It is correlated with functional correctness of the independent modules.
Unit Testing is defined as a type of software testing where individual components of a software are tested.
Unit Testing of software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.

In SDLC or V Model, Unit testing is first level of testing done before integration testing. Unit testing is such type of testing technique that is usually performed by the developers. Although due to reluctance of developers to tests, quality assurance engineers also do unit testing.

Objective of Unit Testing:
The objective of Unit Testing is:
1. To isolate a section of code.
2. To verify the correctness of code.
3. To test every function and procedure.
4. To fix bug early in development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help for code reuse.



Types of Unit Testing:
There are 2 type of Unit Testing: **Manual**, and **Automated**.
Workflow of Unit Testing:



Unit Testing Tools:
Here are some commonly used Unit Testing tools:
1. Jtest
2. Junit
3. NUnit
4. EMMA
5. PHPUnit

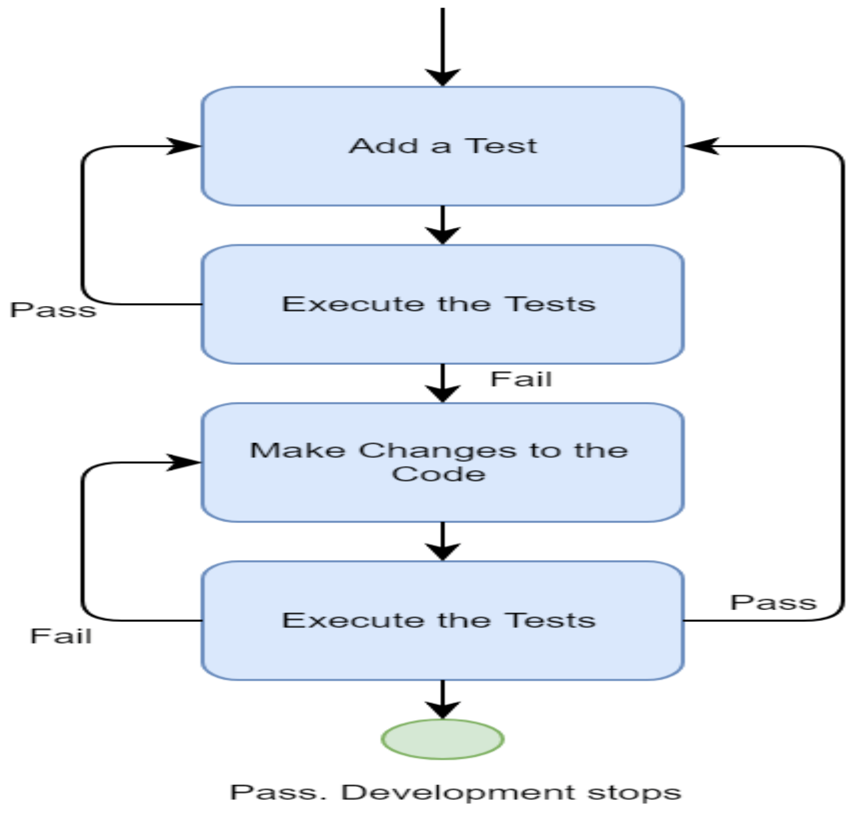
Advantages of Unit Testing:

- Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refine code and make sure the module works properly.
- Unit testing enables to test parts of the project without waiting for others to be completed.

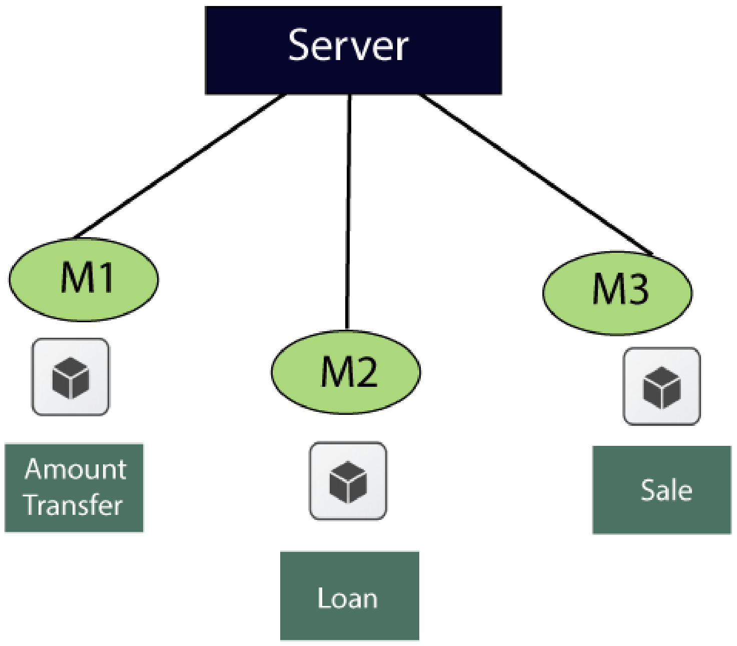
Some crucial reasons are listed below:

- 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 Flow Chart Representation for unit testing



Example 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

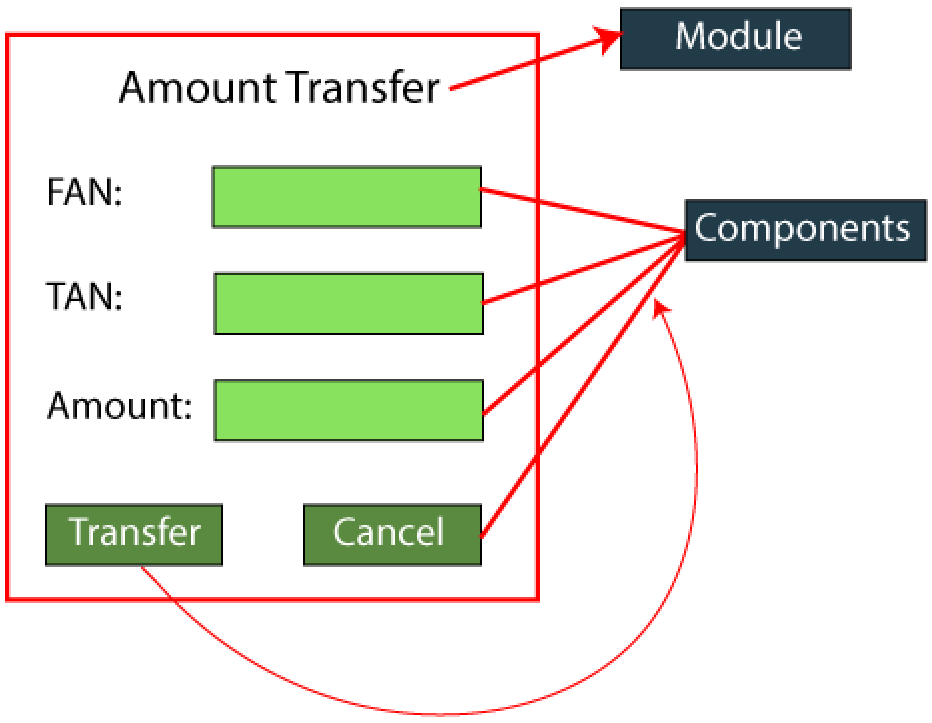
While performing unit testing, we should follow some rules, which are as follows:

- 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.

Benefits of Unit Testing

Unit tests can be a great investment if done correctly.

Unit testing helps in finding bugs early.

Developers can write unit tests as soon as they finish writing code without having to wait for others. This makes it easier for developers to identify and fix bugs as they are usually quite familiar with the code they recently worked on.

Unit testing makes the team in the long run.

Software development is an interactive process. Design and implementation changes are commonplace. If you have unit tests in place, developers can quickly run them and get feedback on the quality of their work.

Unit testing makes debugging easier.

Unit testing simplifies the debugging process. When a test fails, only the newest changes made in the code need to be checked. If you don’t have unit test cases, small changes made over the range of several weeks or months need to be inspected.

Unit testing can be automated.

Unit testing can be easily integrated into the software build process, making it easy to report errors quickly.

Unit testing decreases the total testing cost.

Since unit testing can be automated, it can be performed daily. When bugs are nipped in the bud, integration and system testing also becomes more effective.

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

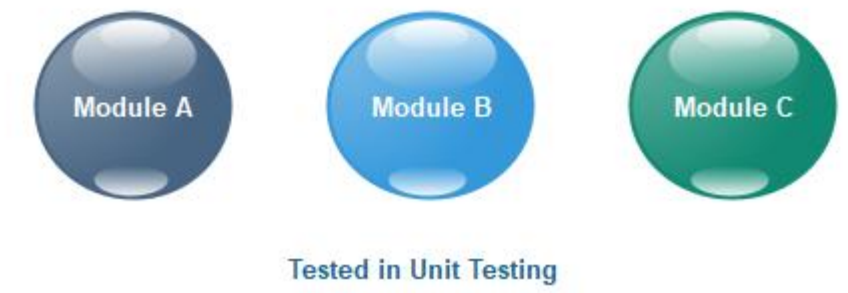
- It cannot identify integration or broad level error as it works on units of the code.
- In the unit testing, evaluation of all execution paths is not possible, so unit testing is not able to catch each and every error in a program.
- It is best suitable for conjunction with other testing activities.

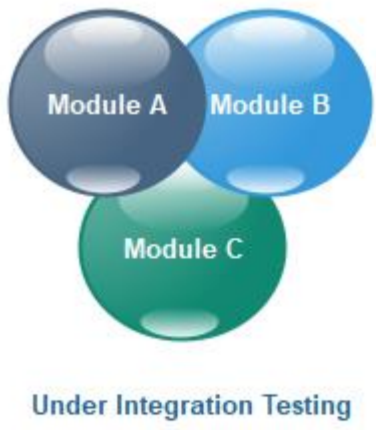
Integration testing

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.

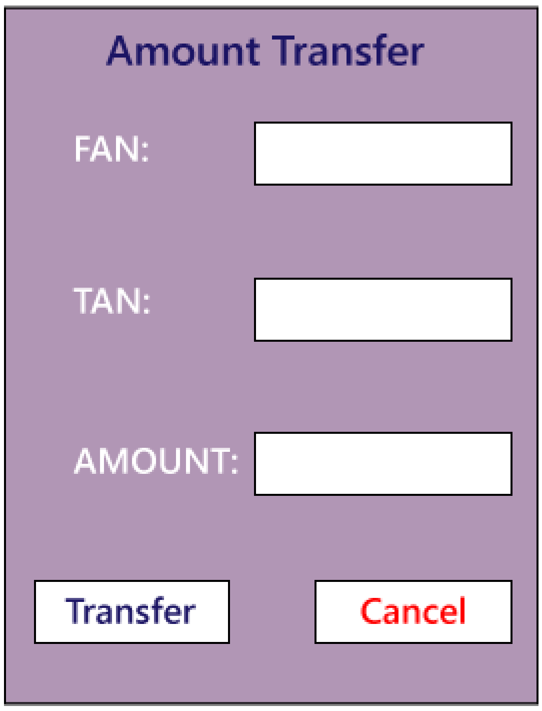
Unit testing

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.





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**.



- 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.

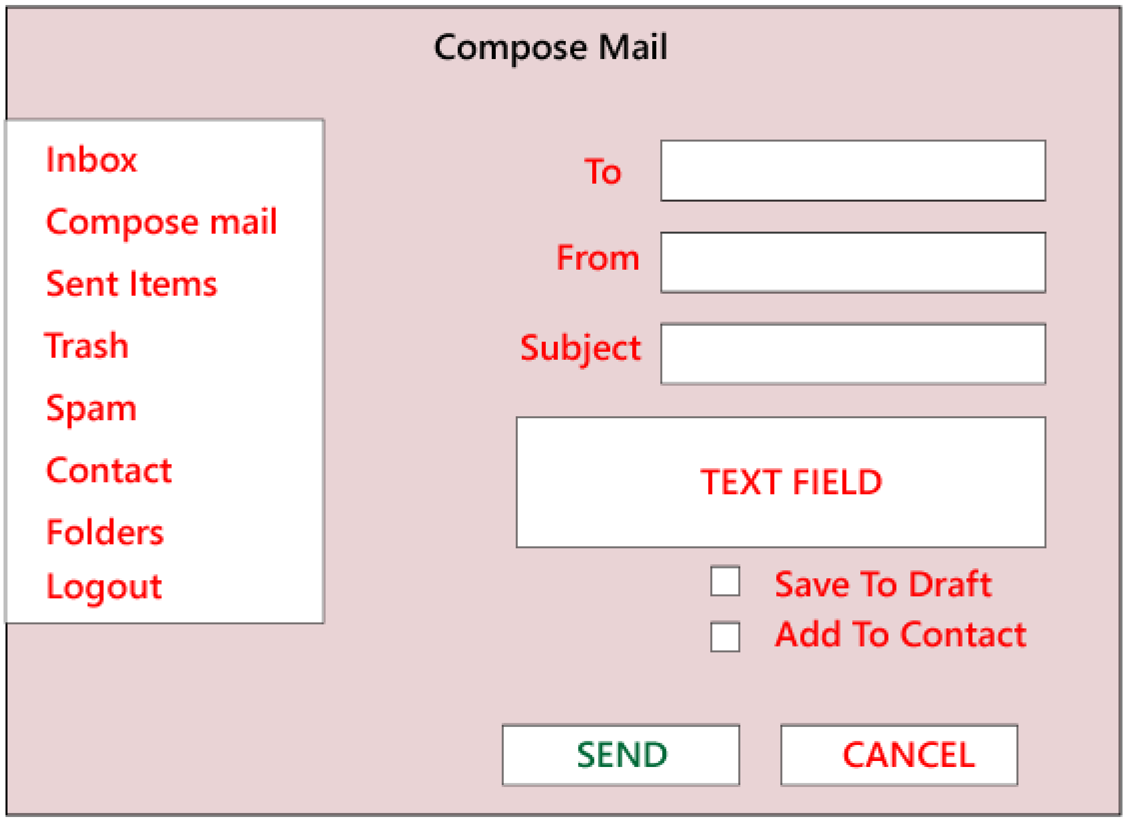
For example: In the Gmail application, the **Source** could be **Compose**, **Data** could be **Email** and the **Destination** could be the **Inbox**.

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:

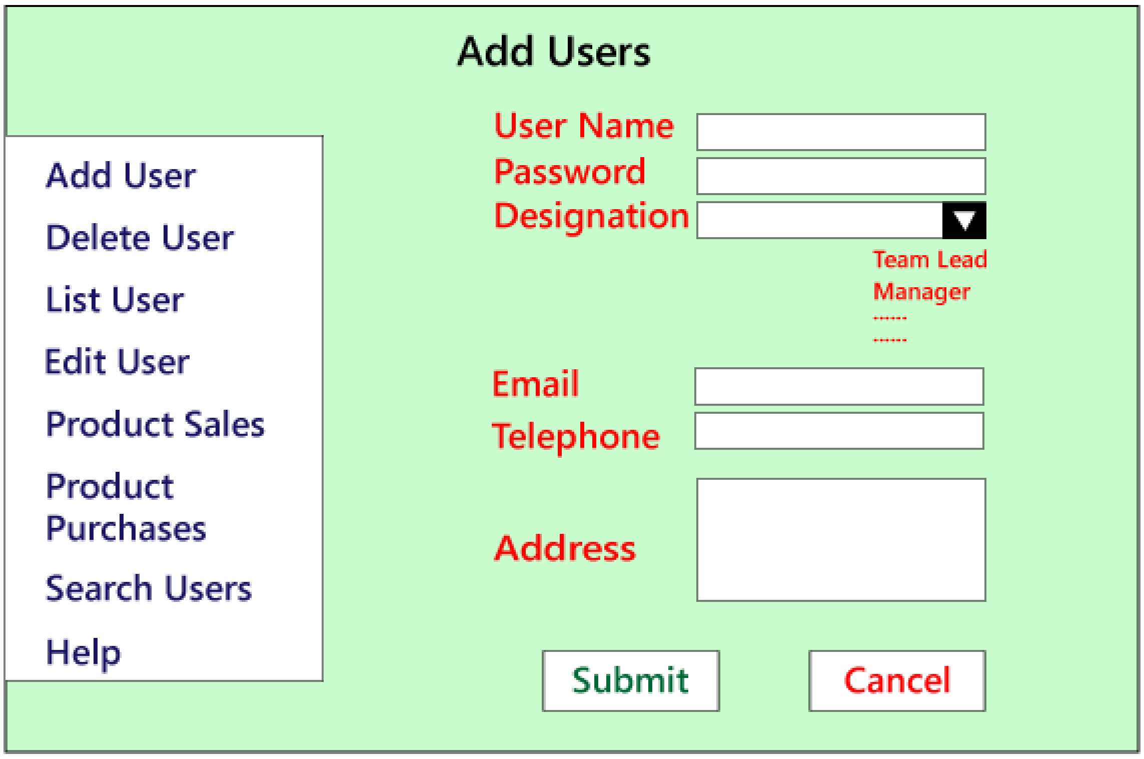
- First, we login as **P** users and click on the **Compose** mail and performing the functional testing for the specific components.
- Now we click on the **Send** and also check for **Save Drafts**.
- After that, we send a **mail** to **Q** and verify in the **Send Items** folder of P to check if the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

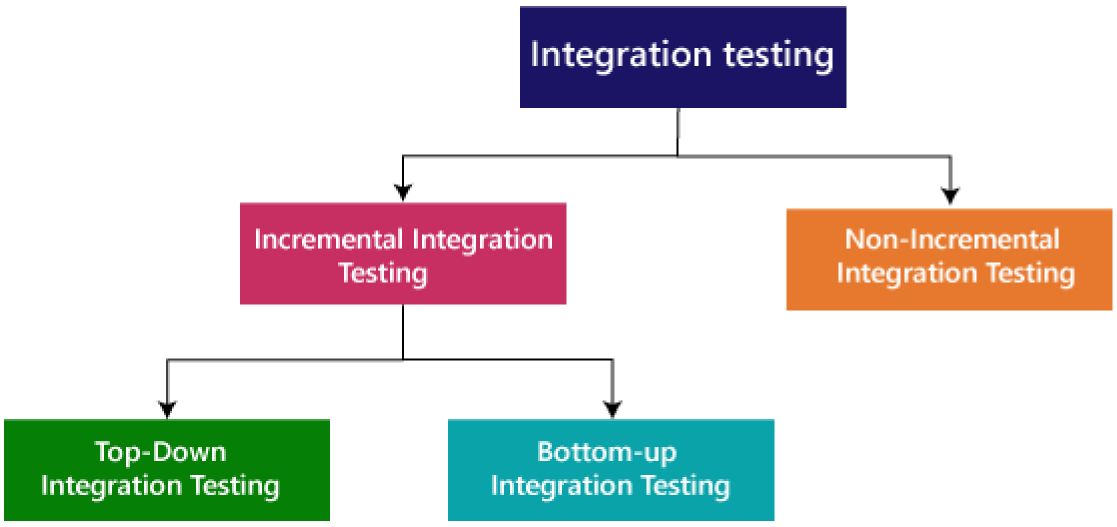
- Data flow testing
- Control Flow Testing

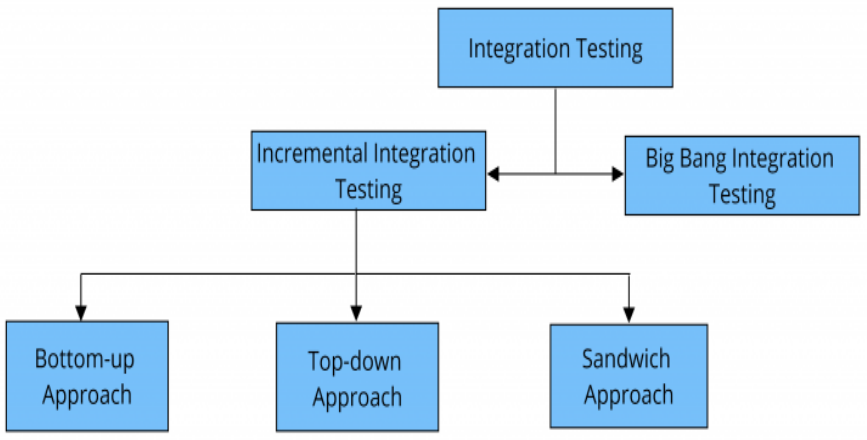
- Branch Coverage Testing
- Decision Coverage Testing

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

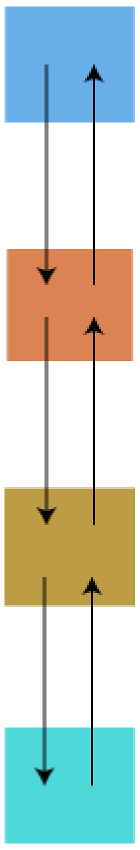


Incremental Approach

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:

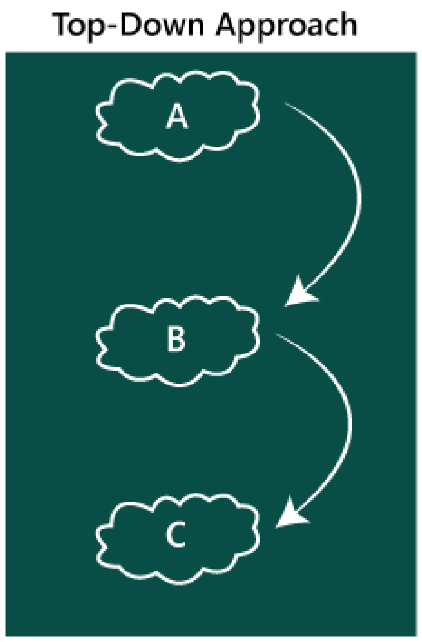
Flipkart→ Login→ Home → Search→ Add cart→Payment → Logout

Incremental integration testing is carried out by further methods:

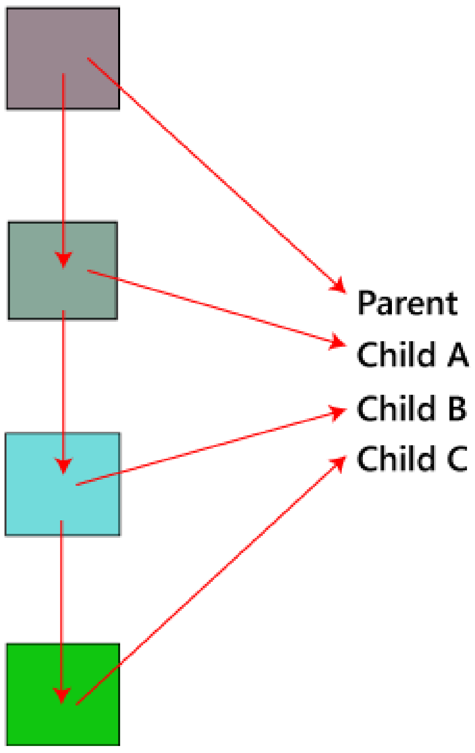
- Top-Down approach
- Bottom-Up approach

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:

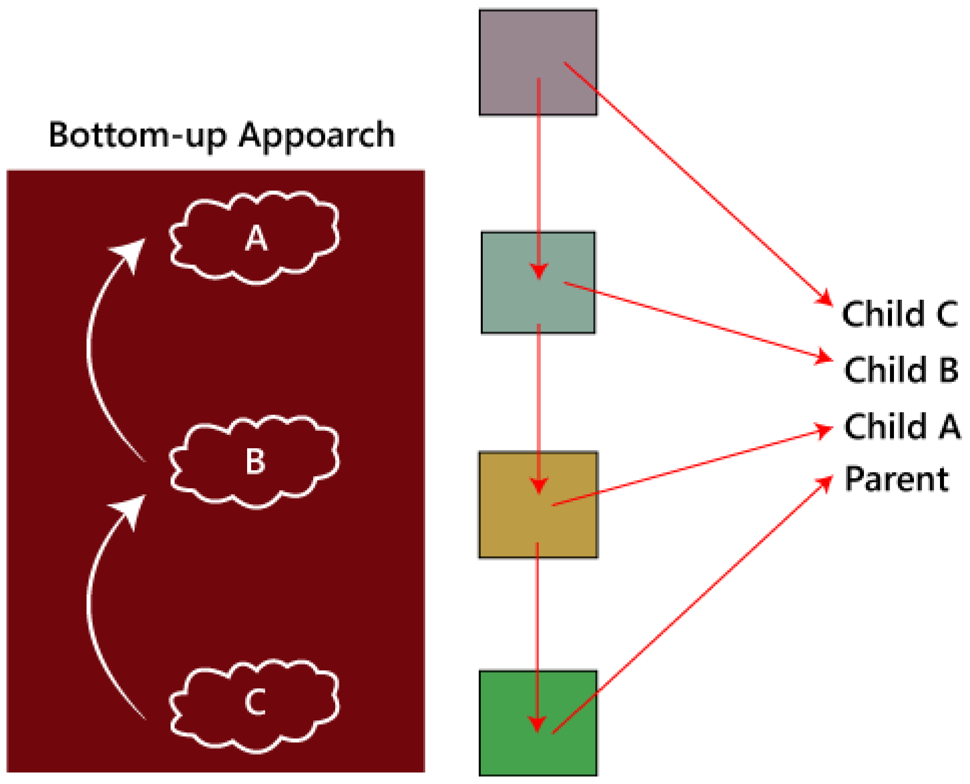
- Identification of defect is difficult.
- An early prototype is possible.

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.



Advantages

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

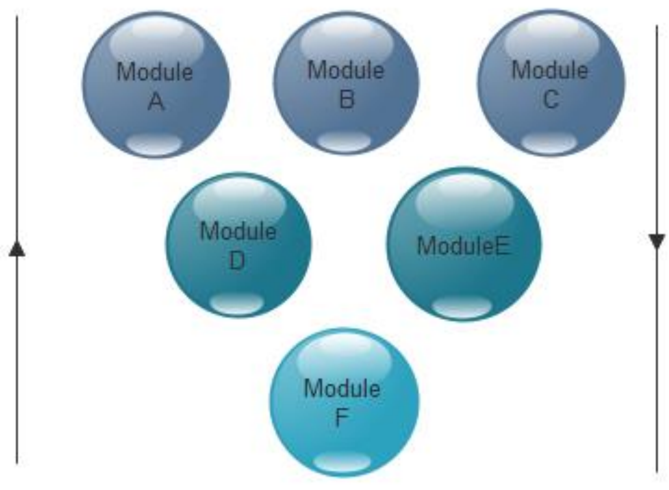
Disadvantages

- 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

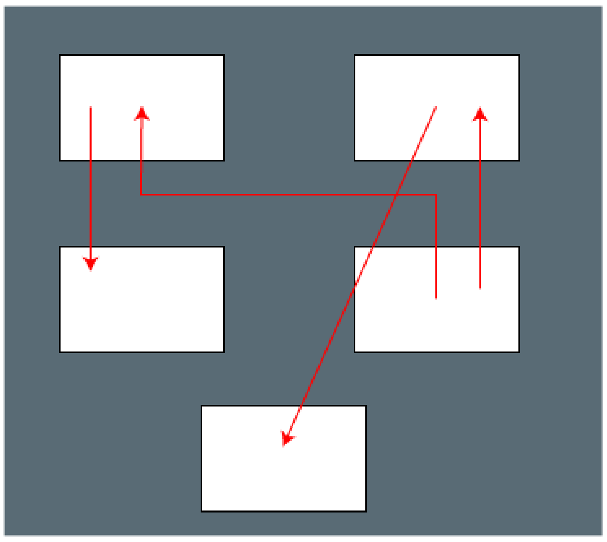
- The hybrid method provides features of both Bottom Up and Top Down methods.
- It is most time reducing method.
- It provides complete testing of all modules.

Disadvantages

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

Non- incremental integration testing

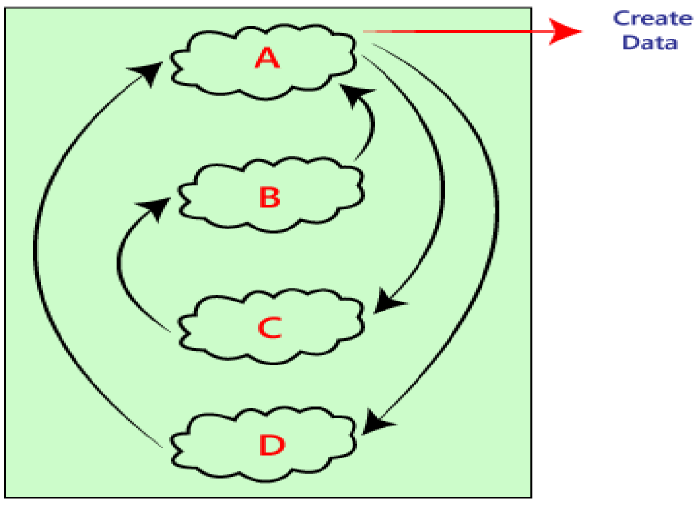
We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. Hence, it is also known as the **Big bang method**.



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.



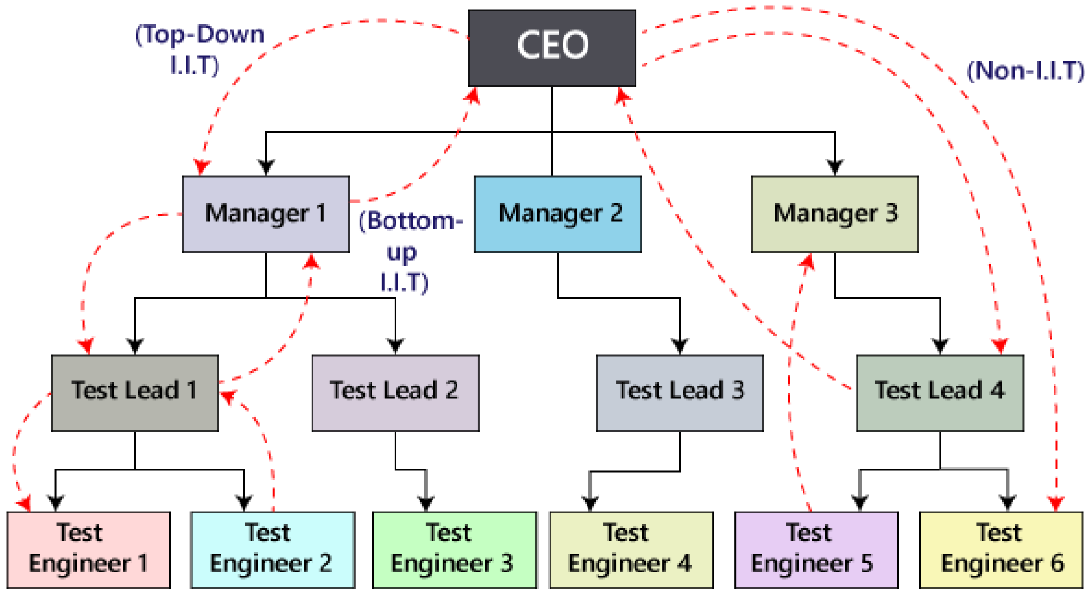
Advantages:

- It is convenient for small size software systems.

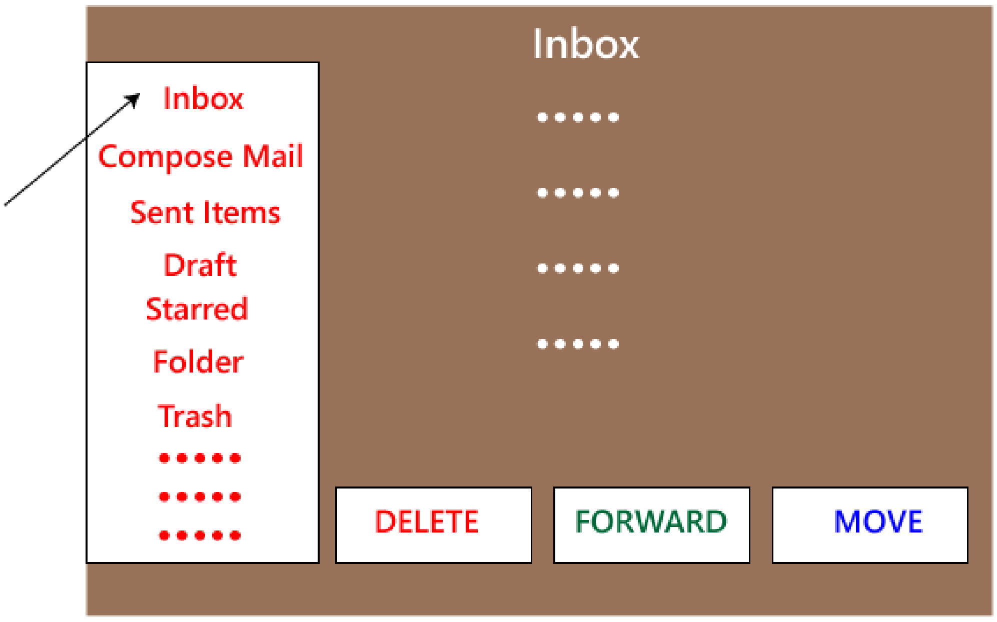
Disadvantages:

- 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:

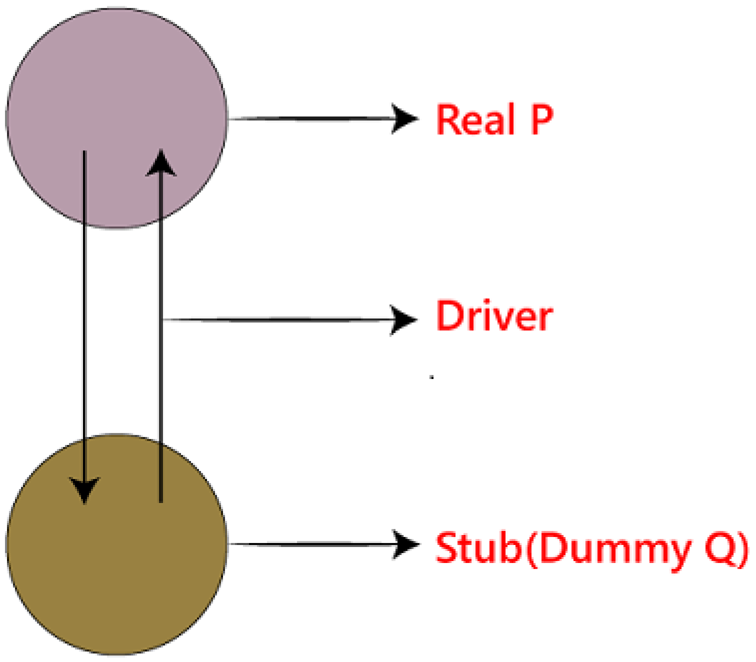
- *Note: The combination incremental integration testing (I.I.T) and non-incremental integration testing is known as sandwich testing.*



- **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.

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.

```
class Condition {
  public static void main (String[]
args) {
    boolean learning = true;

    if (learning) {
      System.out.println ("Java
programmer");
    }
```

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.

White Box Testing Techniques

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It identifies areas of a program that are not exercised by a set of test cases. Once gaps are identified, you create test cases to verify untested parts of the code, thereby increasing the quality of the software product

There are automated tools available to perform Code coverage analysis. Below are a few coverage analysis techniques a box tester can use:

Statement Coverage:- This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.

Branch Coverage – This technique checks every possible path (if-else and other conditional loops) of a software application.

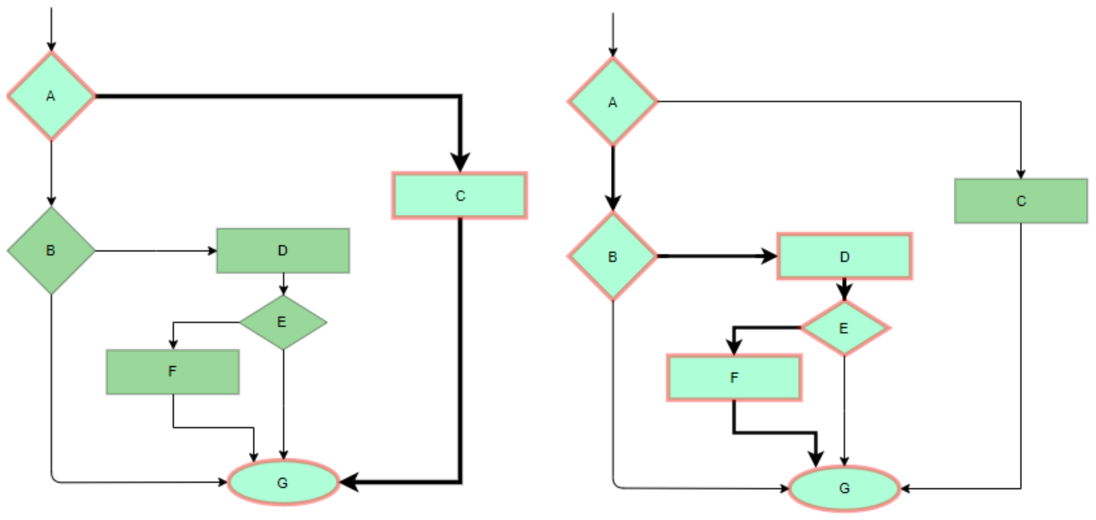
Apart from above, there are numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc. Each technique has its own merits and attempts to test (cover) all parts of software code. **Using Statement and Branch coverage you generally attain 80-90% code coverage which is sufficient.**

Following are important WhiteBox Testing Techniques:

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Multiple Condition Coverage
- Finite State Machine Coverage
- Path Coverage
- Control flow testing
- Data flow testing

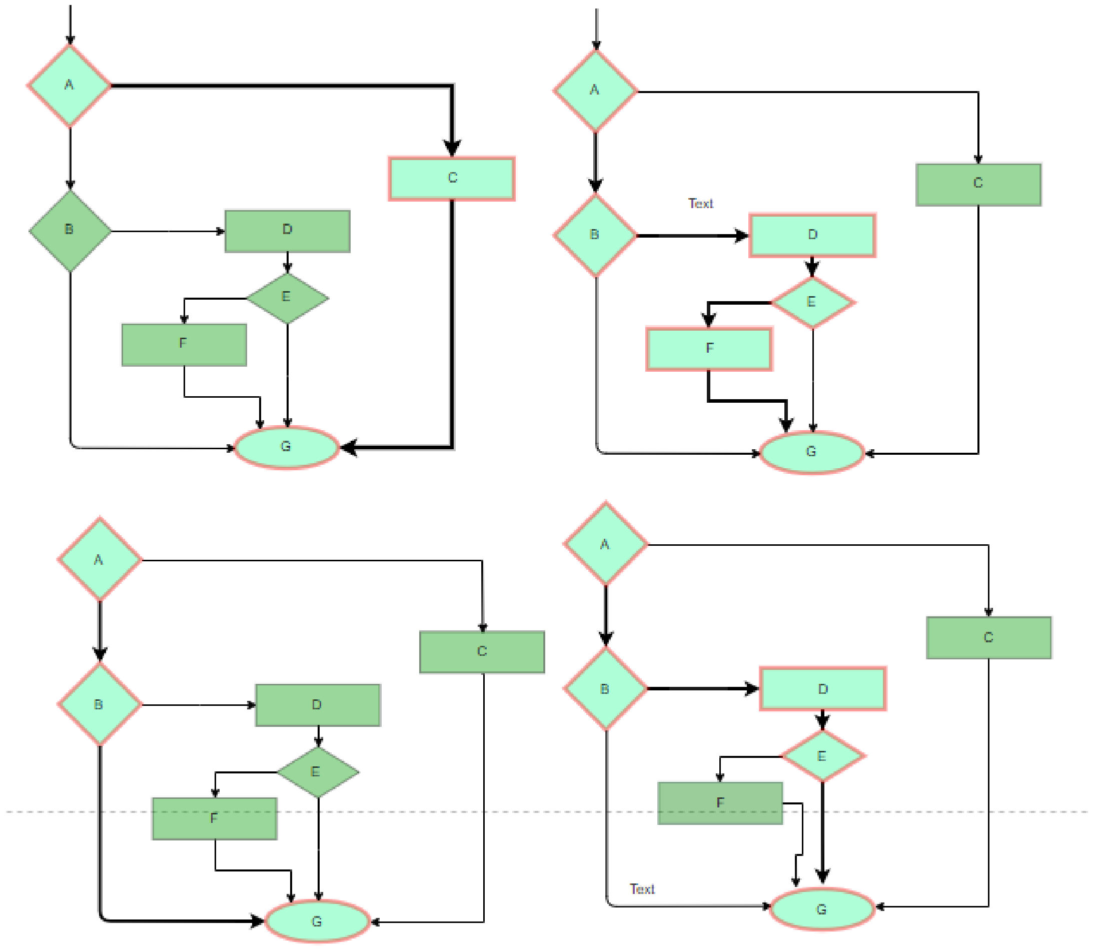
Testing techniques:

- **Statement coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.



Statement Coverage Example

- **Branch Coverage:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.



4 test cases required such that all branches of all decisions are covered, i.e, all edges of flowchart are covered

- **Condition Coverage:** In this technique, all individual conditions must be covered as shown in the following example:

1. READ X, Y
2. IF(X == 0 || Y == 0)
3. PRINT ‘0’

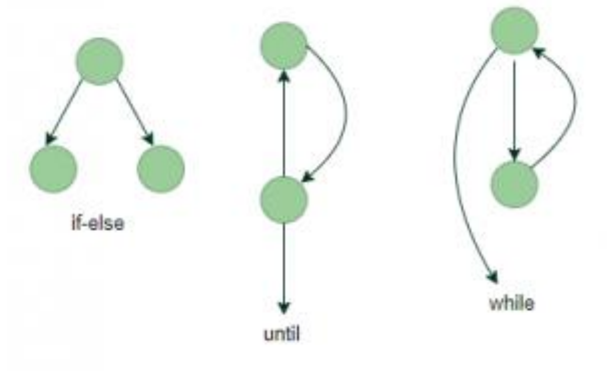
In this example, there are 2 conditions: X == 0 and Y == 0. Now, test these conditions get TRUE and FALSE as their values. One possible example would be:

  - #TC1 – X = 0, Y = 55
  - #TC2 – X = 5, Y = 0

- **Multiple Condition Coverage:** In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let’s consider the following example:
  0. READ X, Y
  1. IF(X == 0 || Y == 0)
  2. PRINT ‘0’
     - #TC1: X = 0, Y = 0
     - #TC2: X = 0, Y = 5
     - #TC3: X = 55, Y = 0
     - #TC4: X = 55, Y = 5

  Hence, four test cases required for two individual conditions.
  Similarly, if there are n conditions then 2^n test cases would be required.
- **Basis Path Testing:** In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.
  Steps:
  0. Make the corresponding control flow graph
  1. Calculate the cyclomatic complexity
  2. Find the independent paths
  3. Design test cases corresponding to each independent path

  Flow graph notation: It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point. A predicate node is the one that represents a decision point that contains a condition after which the graph splits. Regions are bounded by nodes and edges.

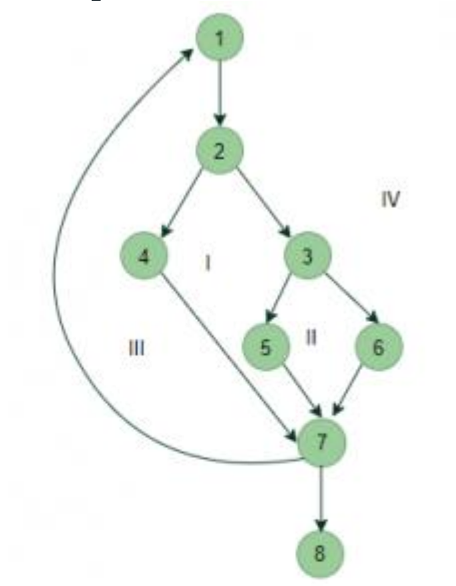


Cyclomatic Complexity: It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G,

V(G) is its cyclomatic complexity.
Calculating V(G):

4. V(G) = P + 1, where P is the number of predicate nodes in the flow graph
5. V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
6. V(G) = Number of non-overlapping regions in the graph

Example:



V(G) = 4 (Using any of the above formulae)
No of independent paths = 4

- #P1: 1 – 2 – 4 – 7 – 8
- #P2: 1 – 2 – 3 – 5 – 7 – 8
- #P3: 1 – 2 – 3 – 6 – 7 – 8
- #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

- **Loop Testing:** Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.
  0. **Simple loops:** For simple loops of size n, test cases are designed that:
     - Skip the loop entirely
     - Only one pass through the loop
     - 2 passes
     - m passes, where m < n
     - n-1 ans n+1 passes
  1. **Nested loops:** For nested loops, all the loops are set to their minimum count and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
  2. **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each.
     If they're not independent, treat them like nesting.

Advantages:

1. White box testing is very thorough as the entire code and structures are tested.

2. It results in the optimization of code removing error and helps in removing extra lines of code.
3. It can start at an earlier stage as it doesn't require any interface as in case of black box testing.
4. Easy to automate.

Disadvantages:

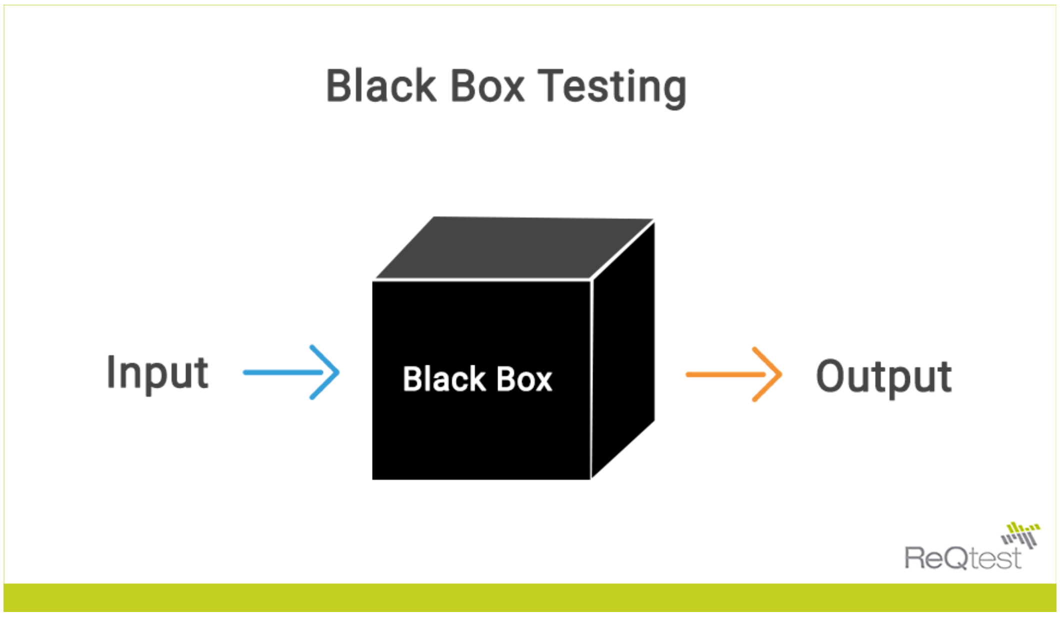
1. Main disadvantage is that it is very expensive.
2. Redesign of code and rewriting code needs test cases to be written again.
3. Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.
4. Missing functionalities cannot be detected as the code that exists is tested.
5. Very complex and at times not realistic.

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.

Various parameters checked in black box testing are:

- Accurate actions performed by users
- System’s interaction with the inputs
- The response time of the system
- Use of data structures Issues in the user interface
- Usability issues
- Performance issues
- Abrupt application failure, unable to start or finish

What are Black Box testing techniques?

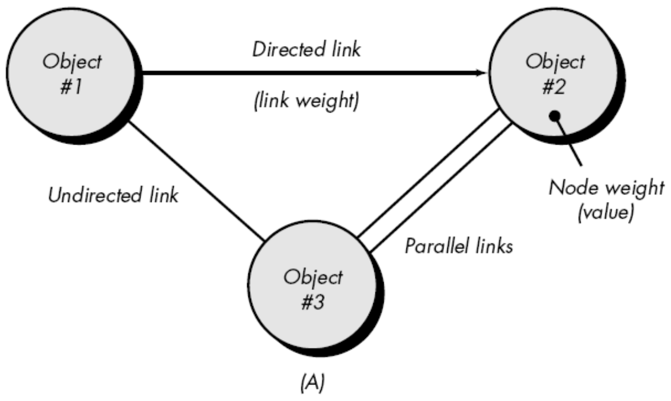
There are various test case design techniques applied for black-box testing:

1. Boundary Value Analysis
2. Equivalence partitioning
3. Graph-Based Testing
4. Orthogonal Array Testing:

Graph based testing method:

(1) Draw a graph of objects and relations

(2) Devise test cases t uncover the graph such that each object and its relationship exercised.



Equivalence partitioning

1. divides the input domain of a program into classes of data from which test cases can be derived.
2. An equivalence class represents a set of valid or invalid states for input conditions(i.e. specific numeric value, a range of values, a set of related values, or a Boolean)

Boundary Value analysis:

1. Select input from equivalence classes such that the input lies at the edge of the equivalence classes
2. Set of data lies on the edge or boundary of a class of input data or generates the data that lies at the boundary of a class of output data

Orthogonal Array Testing:

1. Orthogonal array testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.
2. It is particularly useful in finding region faults

Advantages of Black Box Testing :

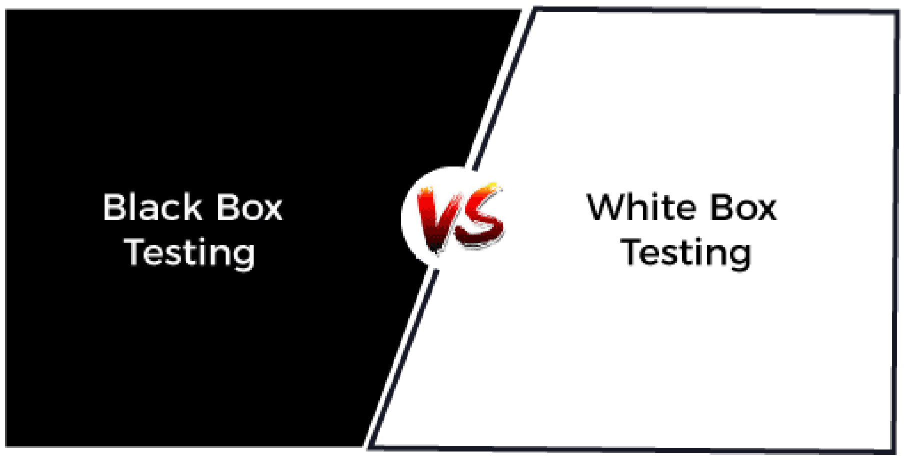
- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user’s or client’s point of view.
- Test cases are easily reproducible.
- It is used in finding the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing :

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.

- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

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.

1.	**Basic**	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.	**Also known as**	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.	**Programming knowledge**	In black-box testing, there is less programming knowledge is required.	In white-box testing, there is a requirement of programming knowledge.
4.	**Algorithm testing**	It is not well suitable for algorithm testing.	It is well suitable and recommended for algorithm testing.

5.	**Usage**	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.	**Automation**	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.	**Tested by**	It is mainly performed by the software testers.	It is mainly performed by developers.
8.	**Time-consuming**	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.	**Base of testing**	The base of this testing is external expectations.	The base of this testing is coding which is responsible for internal working.
10.	**Exhaustive**	It is less exhaustive than White Box testing.	It is more exhaustive than Black Box testing.
11.	**Implementation knowledge**	In black-box testing, there is no implementation knowledge is required.	In white-box testing, there is a requirement of implementation knowledge.
12.	**Aim**	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.	**Defect detection**	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.	**Testing method**	It can be performed by trial and error technique.	It can test data domain and data boundaries in a better way.

15.	**Types**	Mainly, there are three types of black-box testing**: functional testing, Non-Functional testing,** and **Regression testing**.	The types of white box testing are – **Path testing, Loop testing,** and **Condition testing**.
16.	**Errors**	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.

Debugging

Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system.

Debugging process

- Debugging process is not a testing process, but it is the result of testing.
- This process starts with the test cases.
- The debugging process gives two results, i.e the cause is found and corrected second is the cause is not found.

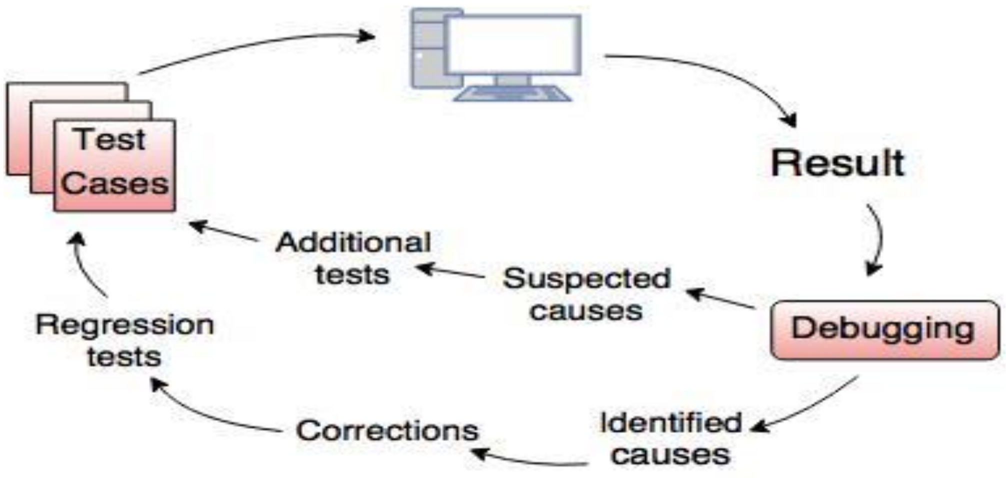


Fig. - Debugging process

Need for debugging:
Once errors are known during a program code, it’s necessary to initial establish the precise program statements liable for the errors and so to repair them.

Challenges in Debugging:
There are lot of problems at the same time as acting the debugging. These are the following:

- Debugging is finished through the individual that evolved the software program and it’s miles difficult for that person to acknowledge that an error was made.
- Debugging is typically performed under a tremendous amount of pressure to fix the supported error as quick as possible.
- It can be difficult to accurately reproduce input conditions.
- Compared to the alternative software program improvement activities, relatively little research, literature and formal preparation exist at the procedure of debugging.

Debugging Approaches:
The following are a number of approaches popularly adopted by programmers for debugging.

- **Brute Force Method:**
  This is the foremost common technique of debugging however is that the least economical method. during this approach, the program is loaded with print statements to print the intermediate values with the hope that a number of the written values can facilitate to spot the statement in error.

- **Backtracking:**
  This is additionally a reasonably common approach. during this approach, starting from the statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered. Backtracking is successfully used in small programs.

- **Cause Elimination Method:**
  In this approach, a listing of causes that may presumably have contributed to the error symptom is developed and tests are conducted to eliminate every error. A connected technique of identification of the error from the error symptom is that the package fault tree analysis.

- **Program Slicing:**
  This technique is analogous to backtracking. Here the search house is reduced by process slices. A slice of a program for a specific variable at a particular statement is that the set of supply lines preceding this statement which will influence the worth of that variable

Debugging Guidelines:
Debugging is commonly administrated by programmers supported their ingenuity. The subsequent are some general tips for effective debugging:

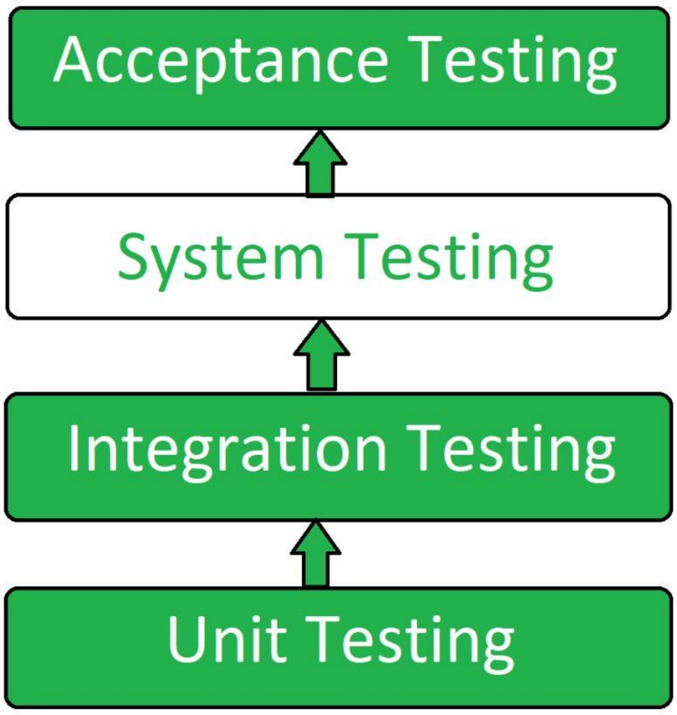
- Many times debugging needs an intensive understanding of the program style. making an attempt to rectify supported a partial understanding of the system style and implementation might need an excessive quantity of effort to be placed into debugging even straightforward issues.

- Debugging might generally even need a full plan of the system. In such cases, a typical mistake that novice programmers usually create is trying to not fix the error however its symptoms.

- One should be watched out for the likelihood that a slip correction might introduce new errors. so when each spherical of error-fixing, regression testing should be administrated.

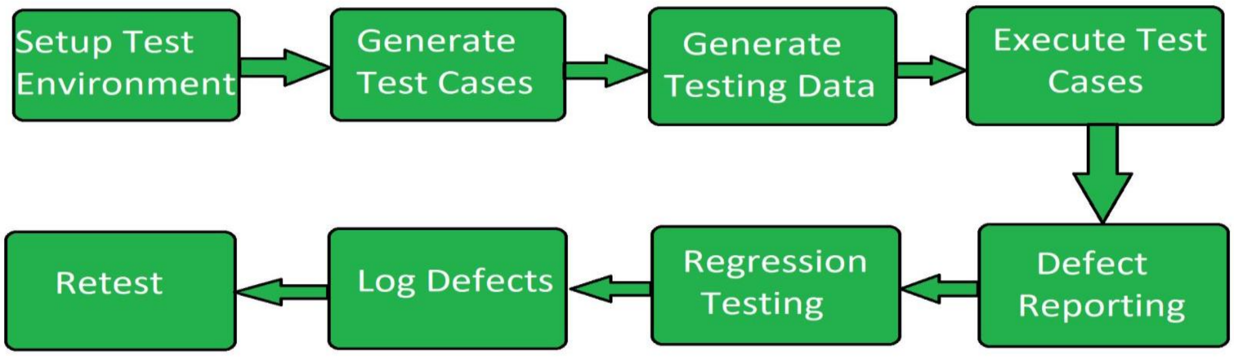
System Testing

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested. **System Testing** is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS). System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing. **System Testing is a black-box testing**. System Testing is performed after the integration testing and before the acceptance testing.



System Testing Process: System Testing is performed in the following steps:

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



Types of System Testing:

Performance Testing:

Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.

Load Testing:

Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.

Stress Testing:

Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.

Scalability Testing:

Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

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.

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.

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.

Tools used for System Testing :

1. JMeter
2. Gallen Framework
3. Selenium

Advantages of System Testing :

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing.

Disadvantages of System Testing :

- This testing is time consuming process than another testing techniques since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of entire software.
- It needs good debugging tool otherwise the hidden errors will not be found.