Module 6

All testing concepts

Unit Testing:

- Unit testing is a type of software testing that focuses on individual units or components of a software system.
- The purpose of unit testing is to validate that each unit of the software works as intended and meets the requirements.
- Unit testing is typically performed by developers, and it is performed early in the development process before the code is integrated and tested as a whole system.

- Unit tests are automated and are run each time the code is changed to ensure that new code does not break existing functionality.
- Unit tests are designed to validate the smallest possible unit of code, such as a function or a method, and test it in isolation from the rest of the system.
- This allows developers to quickly identify and fix any issues early in the development process, improving the overall quality of the software and reducing the time required for later testing.

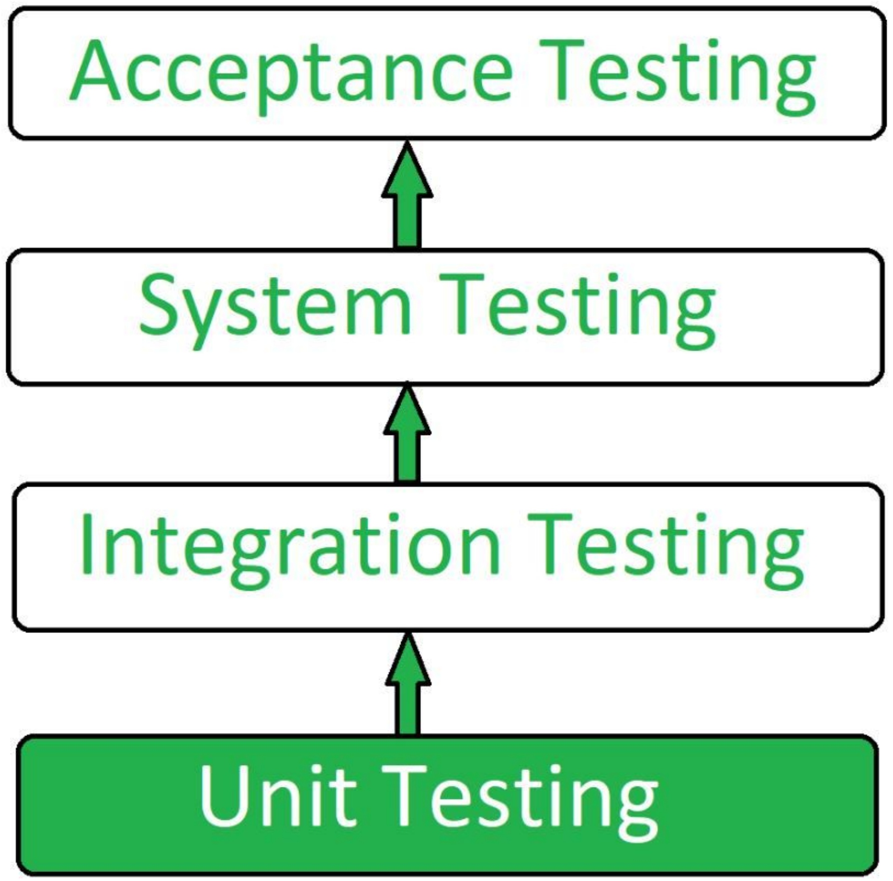
- **Unit Testing** is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not.
- It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself.
- It is correlated with the 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 the 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 the first level of testing done before integration testing. Unit testing is such a type of testing technique that is usually performed by developers.
- Although due to the reluctance of developers to test, quality assurance engineers also do unit testing.

Objective of Unit Testing:

- The objective of Unit Testing is:

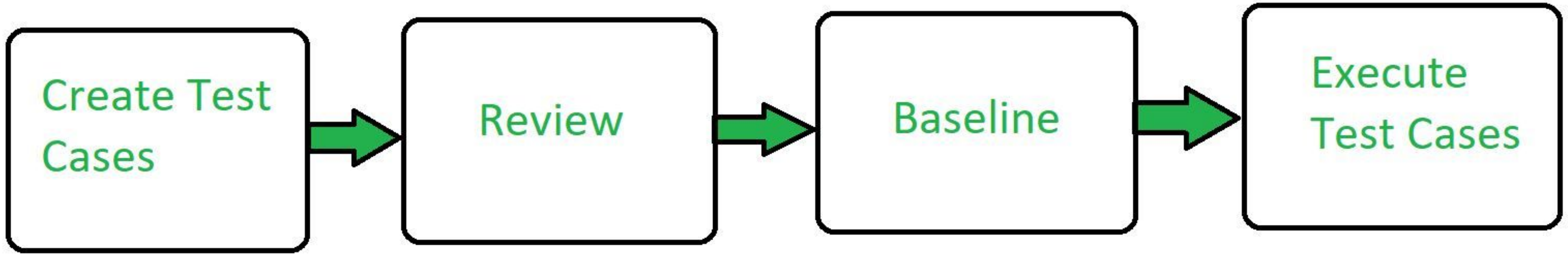
1. To isolate a section of code.
2. To verify the correctness of the code.
3. To test every function and procedure.
4. To fix bugs early in the development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help with code reuse.


Acceptance Testing
System Testing
Integration Testing
Unit Testing

Types of Unit Testing:

- There are 2 types of Unit Testing: **Manual**, and **Automated**.

Workflow of Unit Testing:



Unit Testing Techniques:

- There are 3 types of Unit Testing Techniques. They are

1. **Black Box Testing:** This testing technique is used in covering the unit tests for input, user interface, and output parts.
2. **White Box Testing:** This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
3. **Gray Box Testing:** This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

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:

1. 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.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project without waiting for others to be completed.
4. Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.
5. Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
6. Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.

1. Faster Development: Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.
2. Better Documentation: Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.
3. Facilitation of Refactoring: Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.
4. Reduced Time and Cost: Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.

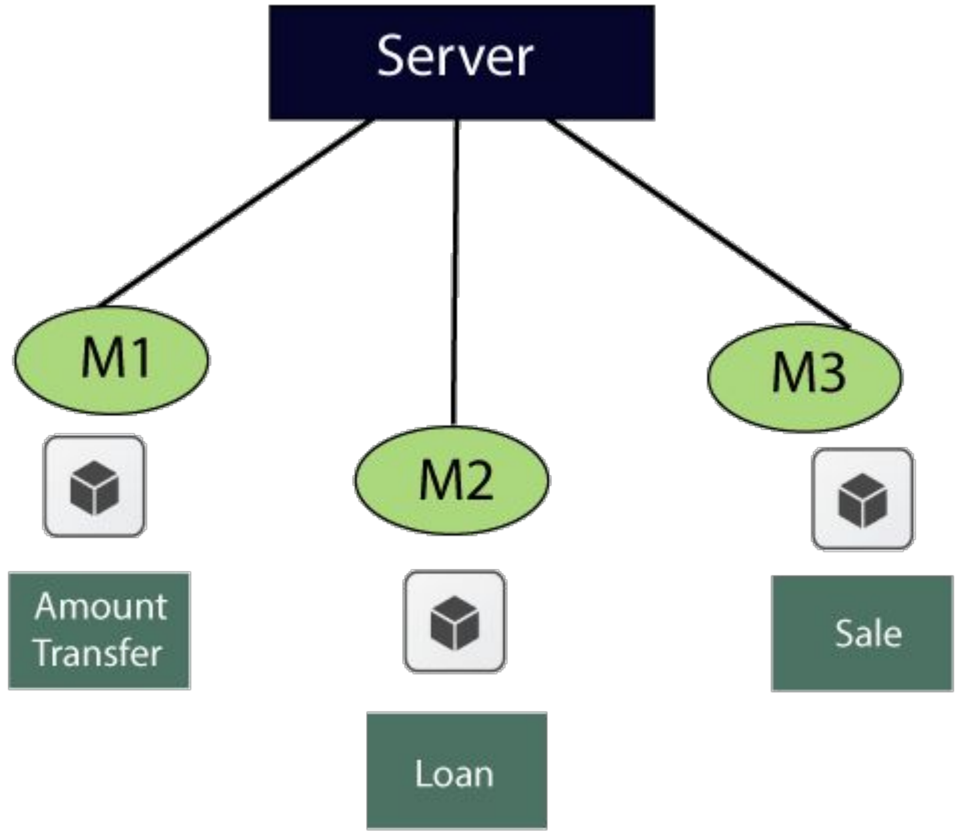
Disadvantages of Unit Testing:

1. The process is time-consuming for writing the unit test cases.
2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
3. Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
4. It requires more time for maintenance when the source code is changed frequently.
5. It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.
6. Time and Effort: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.
7. Dependence on Developers: The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.
8. Difficulty in Testing Complex Units: Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.

1. Difficulty in Testing Interactions: Unit testing may not be sufficient for testing interactions between units, as it only focuses on individual units.
2. Difficulty in Testing User Interfaces: Unit testing may not be suitable for testing user interfaces, as it typically focuses on the functionality of individual units.
3. Over-reliance on Automation: Over-reliance on automated unit tests can lead to a false sense of security, as automated tests may not uncover all possible issues or bugs.
4. Maintenance Overhead: Unit testing requires ongoing maintenance and updates, as the code and test cases must be kept up-to-date with changes to the software.

Example of Unit testing

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



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

1.	Amount transfer
1.1	From account number (FAN)→ Text Box
1.1.1	FAN→ accept only 4 digit
1.2	To account no (TAN)→ Text Box
1.2.1	TAN→ Accept only 4 digit
1.3	Amount→ Text Box
1.3.1	Amount → Accept maximum 4 digit
1.4	Transfer→ Button
1.4.1	Transfer → Enabled
1.5	Cancel→ Button
1.5.1	Cancel→ Enabled

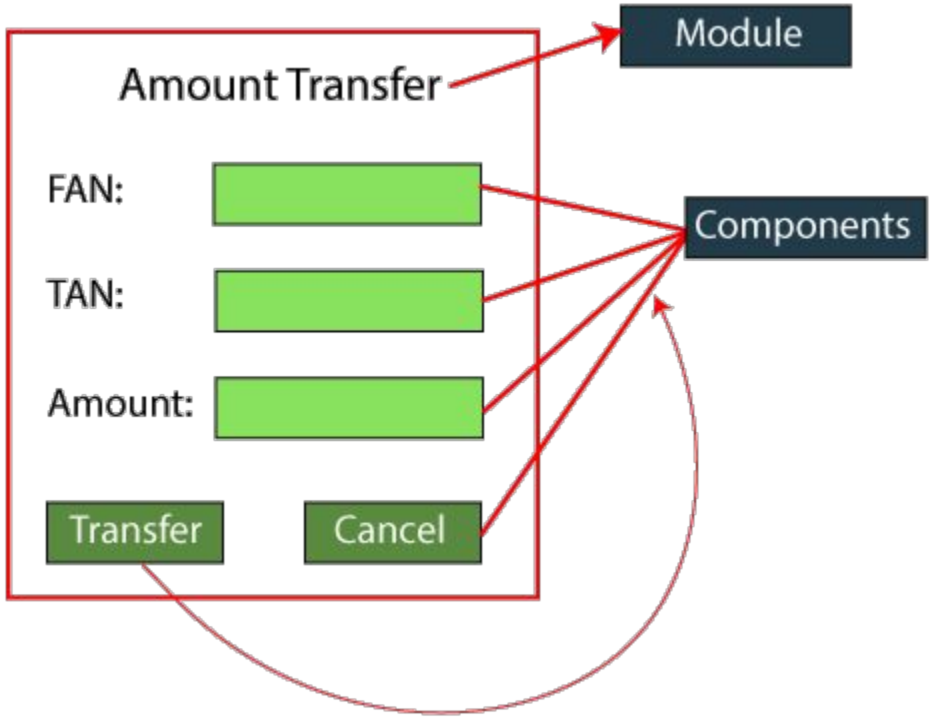
Below are the application access details, which is given by the customer

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

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

- To start unit testing, at least we should have one module.
- Test for positive values
- Test for negative values
- No over testing
- No assumption required
- When we feel that the **maximum test coverage** is achieved, we will stop the testing.

- Now, we will start performing the unit testing on the different components such as
- **From account number(FAN)**
- **To account number(TAN)**
- **Amount**
- **Transfer**
- **Cancel**


Amount Transfer
FAN:
TAN:
Amount:
Transfer
Cancel
Module
Components

For the FAN components

Values	Description
1234	accept
4311	Error message→ account valid or not
blank	Error message→ enter some values
5 digit/ 3 digit	Error message→ accept only 4 digit
Alphanumeric	Error message → accept only digit
Blocked account no	Error message
Copy and paste the value	Error message→ type the value
Same as FAN and TAN	Error message

For the TAN component

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

For Amount component

- Provide the values just like we did in FAN and TAN components.
- **For Transfer component**
- Enter valid FAN value
- Enter valid TAN value
- Enter the correct value of Amount
- Click on the Transfer button→ amount transfer successfully(confirmation message)

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

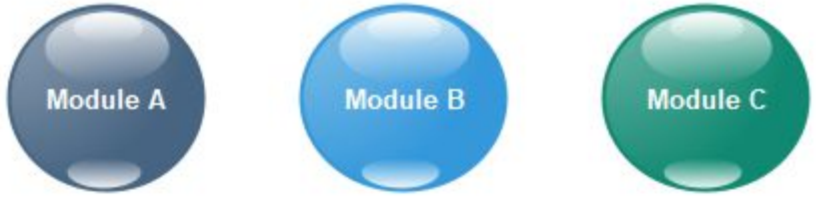
Unit Testing Techniques:

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

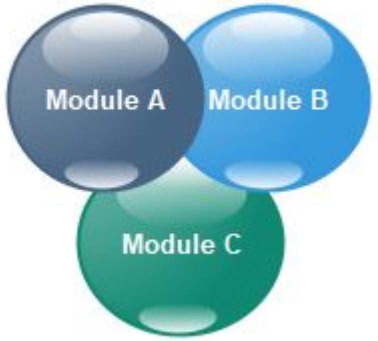
- Data flow Testing
- Control Flow Testing
- Branch Coverage Testing
- Statement Coverage Testing
- Decision Coverage Testing

Integration testing:

- Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.
- Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.



Tested in Unit Testing



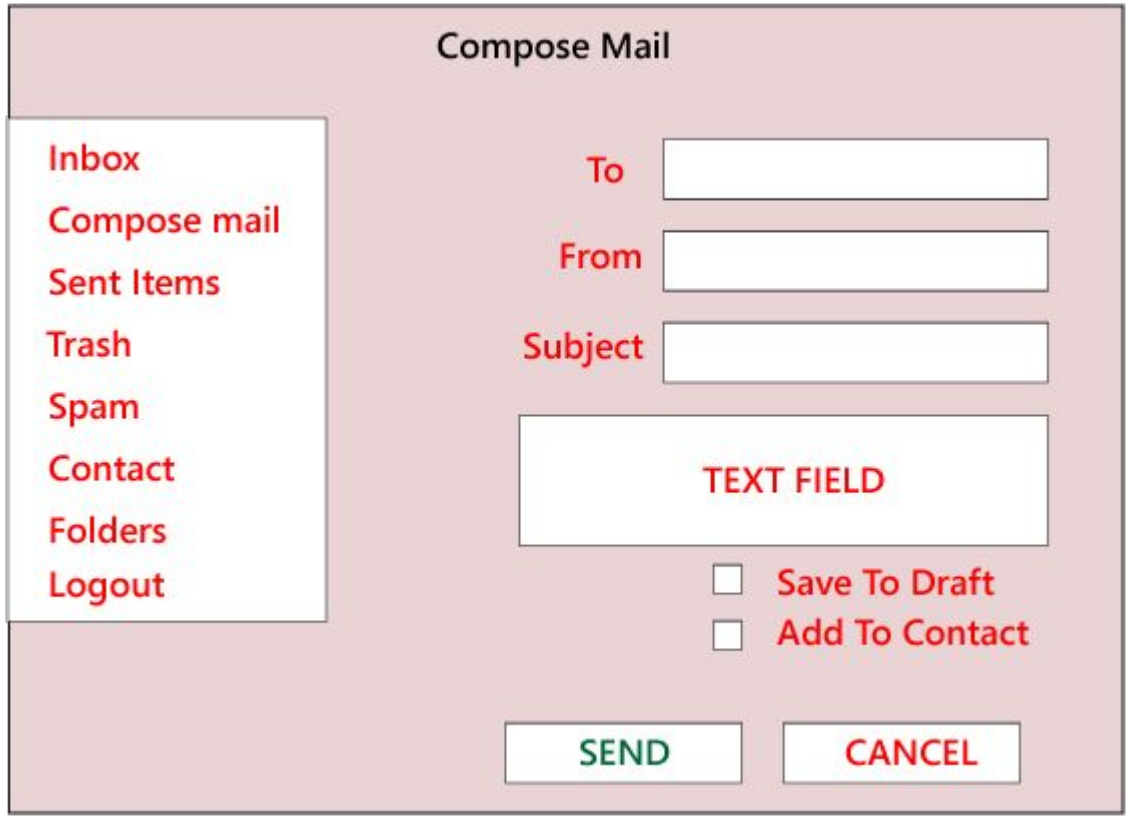
Under Integration Testing

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

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:



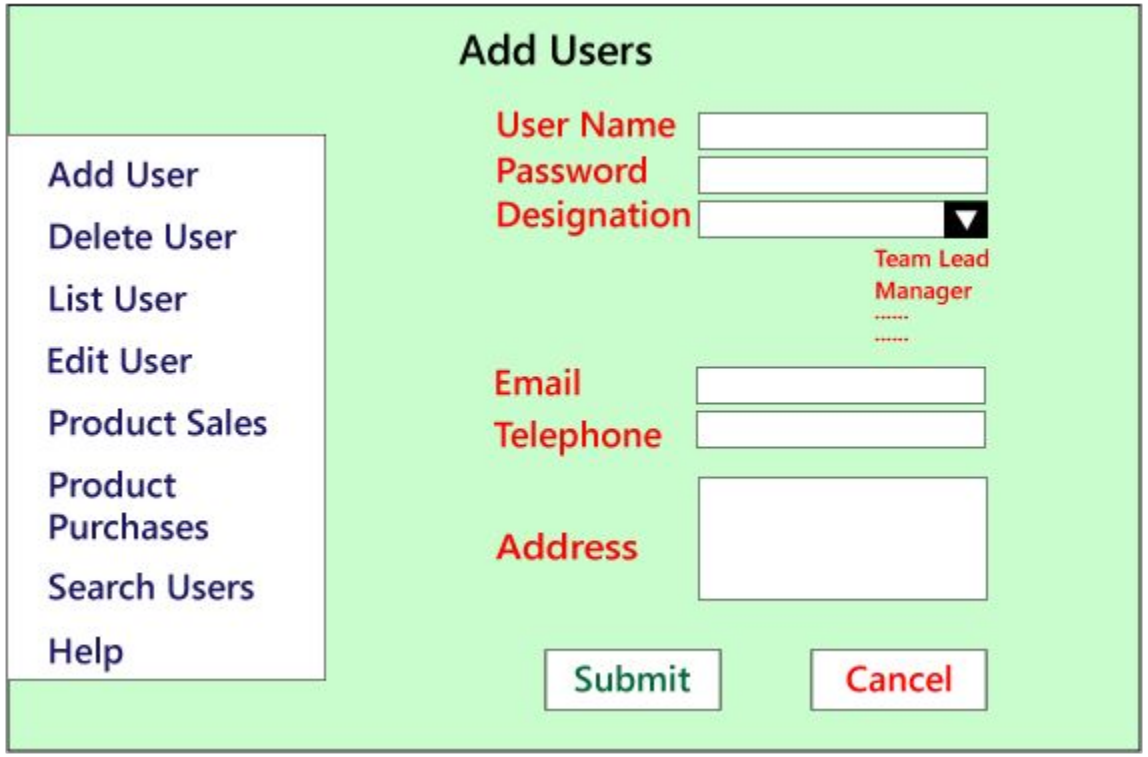
Scenarios1:

- 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.

- 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**.


Add Users
Add User
Delete User
List User
Edit User
Product Sales
Product Purchases
Search Users
Help
User Name
Password
Designation
Team Lead
Manager
......
......
Email
Telephone
Address
Submit
Cancel

- 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.

White Box Testing

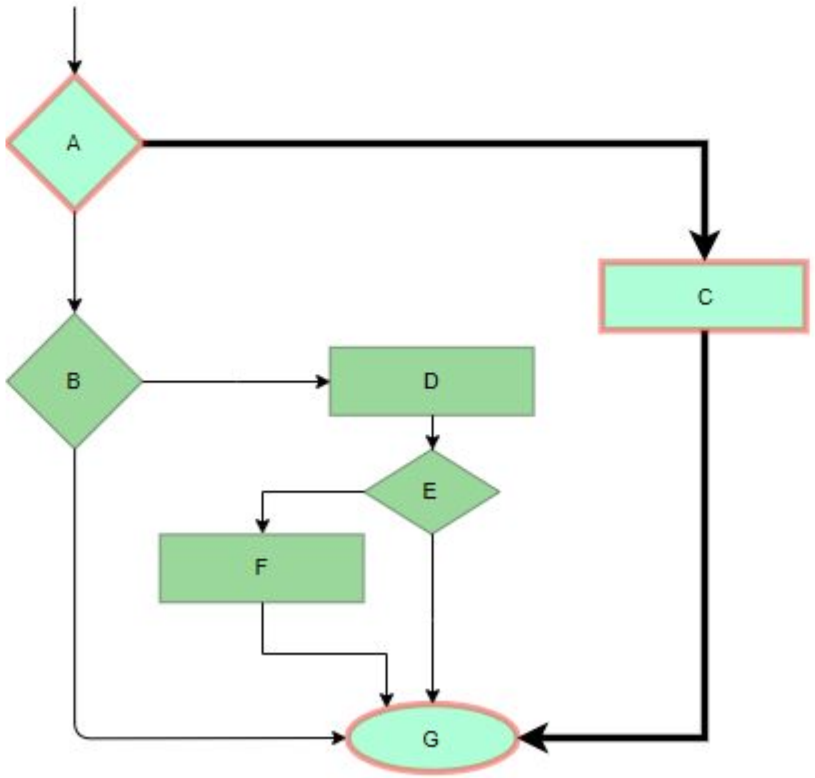
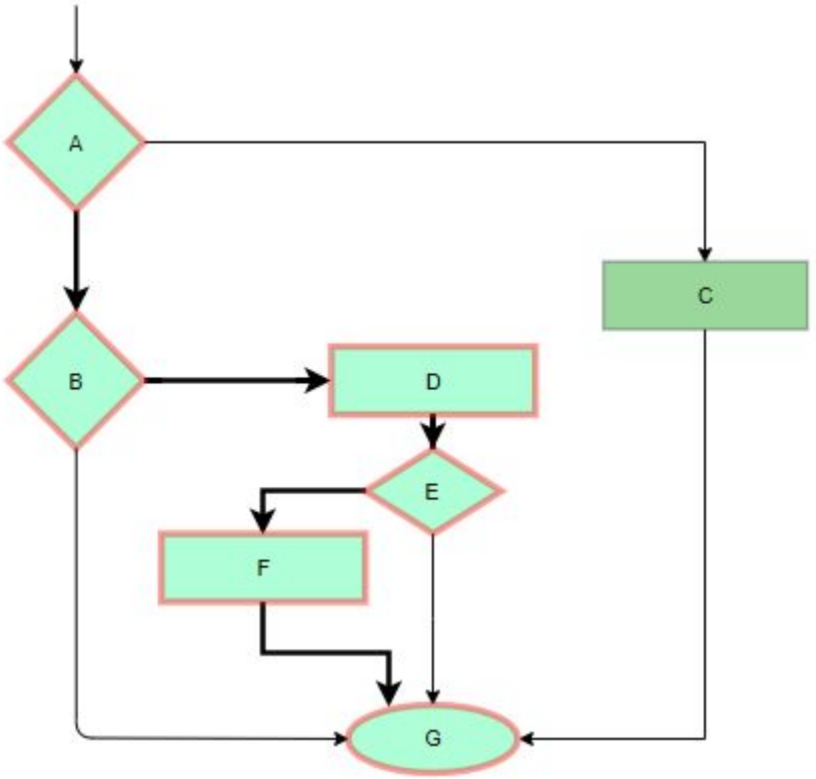
- **White box testing** techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing, open box testing.
- White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

- White box testing is also known as structural testing or code-based testing, and it is used to test the software’s internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure that they meet the specified requirements.

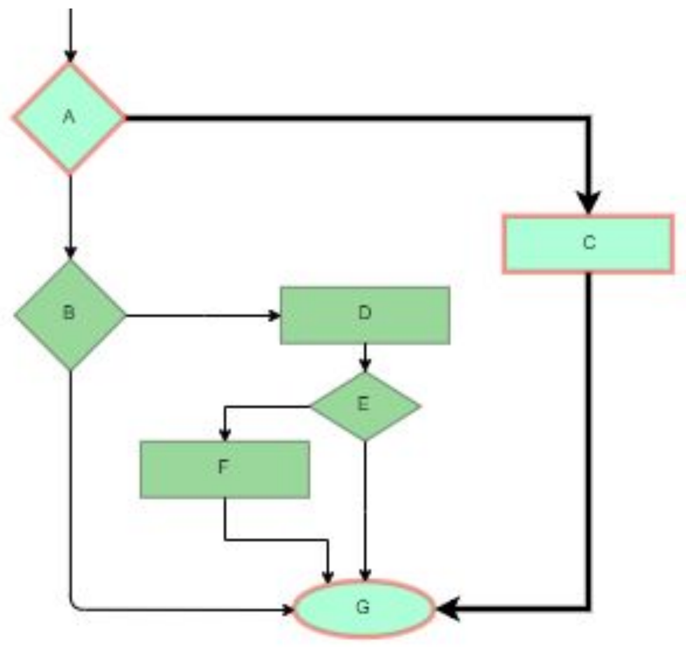
- **Working process of white box testing:**
- **Input:** Requirements, Functional specifications, design documents, source code.
- **Processing:** Performing risk analysis for guiding through the entire process.
- **Proper test planning:** Designing test cases so as to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- **Output:** Preparing final report of the entire testing process.

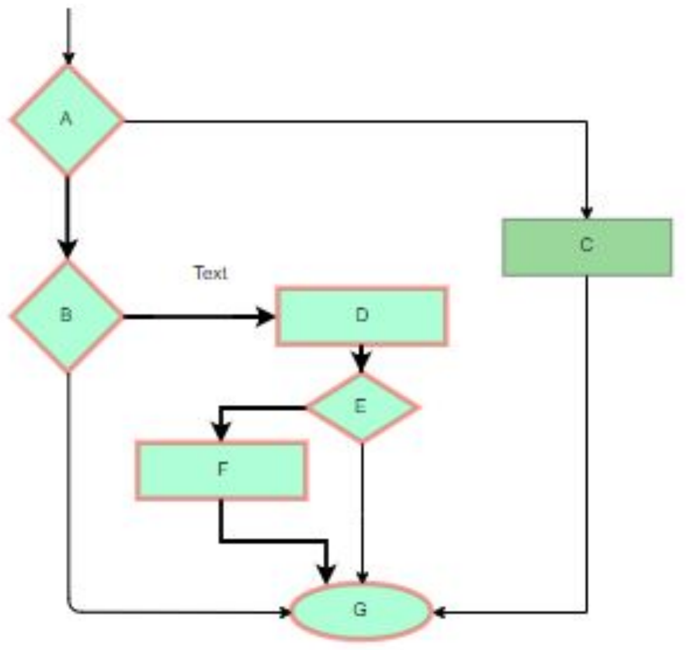
Testing techniques:

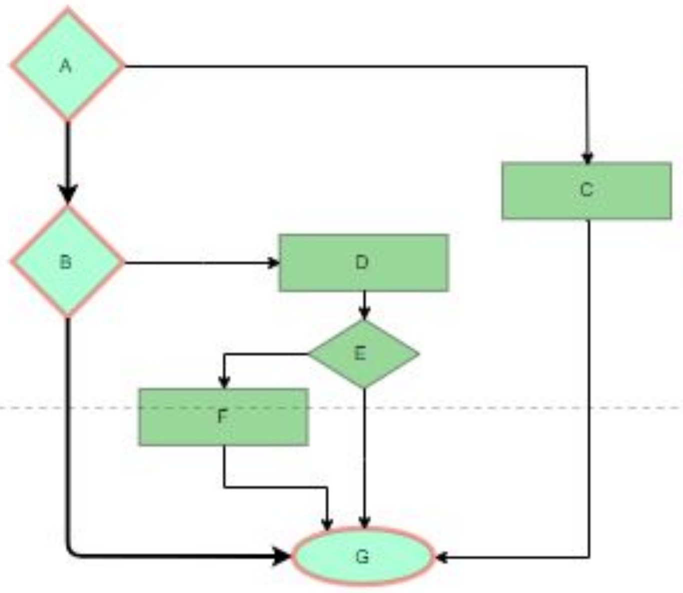
- **Statement coverage:** In this technique, the aim is to traverse all statement at least once.
- Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.

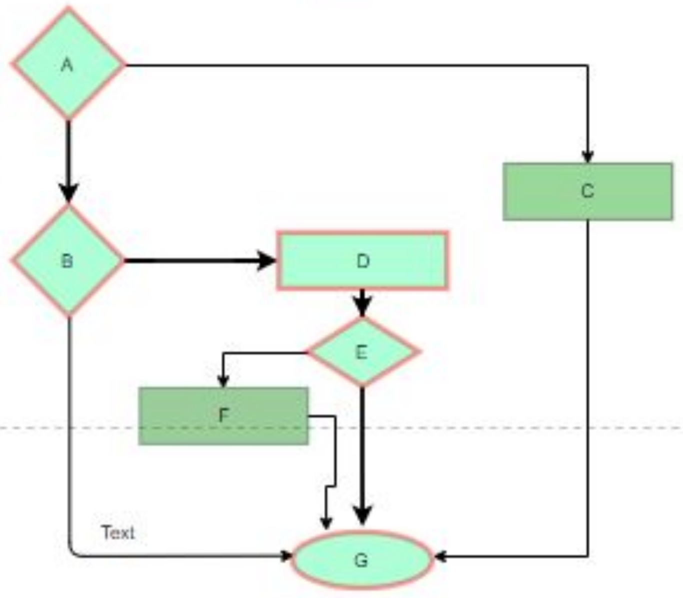

A
B
C
D
E
F
G

A
B
C
D
E
F
G

- **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.


A
B
C
D
E
F
G


A
Text
B
C
D
E
F
G


A
B
C
D
E
F
G


A
B
C
D
E
F
Text
G

- **Condition Coverage:** In this technique, all individual conditions must be covered as shown in the following example:
  - READ X, Y
  - IF(X == 0 || Y == 0)
  - PRINT ‘0’
  - #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:
  - READ X, Y
  - IF(X == 0 || Y == 0)
  - PRINT ‘0’
  - #TC1: X = 0, Y = 0
  - #TC2: X = 0, Y = 5
  - #TC3: X = 55, Y = 0
  - #TC4: X = 55, Y = 5

- **White Testing is Performed in 2 Steps:**
- 1. Tester should understand the code well
- 2. Tester should write some code for test Cases and execute them

- **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.
5. White box testing can be easily started in Software Development Life Cycle.
6. Easy Code Optimization.

- **Disadvantages:**

1. 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.
6. Much more chances of Errors in production.

Difference between white-box testing and black-box testing

White-box testing	Black box testing
The developers can perform white box testing.	The test engineers perform the black box testing.
To perform WBT, we should have an understanding of the programming languages.	To perform BBT, there is no need to have an understanding of the programming languages.
In this, we will look into the source code and test the logic of the code.	In this, we will verify the functionality of the application based on the requirement specification.
In this, the developer should know about the internal design of the code.	In this, there is no need to know about the internal design of the code.

- https://www.javatpoint.com/black-box-testing
- https://www.javatpoint.com/white-box-testing