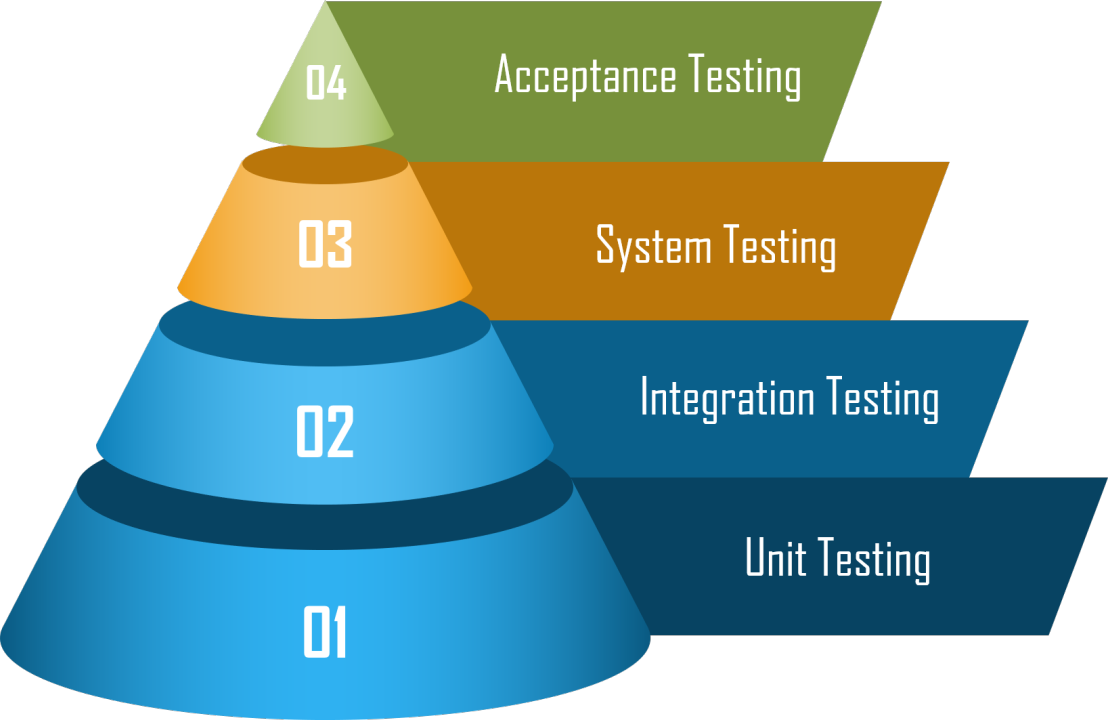
**LEVELS OF TESTING**



Software testing is typically conducted in multiple levels to ensure quality and functionality. The main levels of testing include:

1. **Unit Testing**
   * Tests individual components or functions of the software are working properly.
   * Typically performed by developers.
2. **Integration Testing**
   * Tests how different modules or services work together.
   * Ensures data flow between components is correct.
   * Can involve APIs, databases, or external services.
   * performed by Testers

***Types:***

* + - **Top-down Approach** (test high-level components first):

The higher level modules are tested first then lower level modules test  and integrated in order to check the software functionality.

* Stubs are used for testing if some modules are not ready.
* **Bottom-up Approach** (test low-level components first):

The lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules.

* Drivers are used when some modules are missing and unavailable at time of testing of a specific module.
* **Big Bang Approach** (test everything at once):

All the components or modules are integrated together at once and then tested as a unit.

* If all of the components in the unit are not completed, the integration process will not execute.
* Convenient for small systems

**Unit Integration**: Checks integration of small, related units (e.g., function-to-function calls).

**System Integration**: Tests integration of subsystems or external dependencies like databases, APIs, or third-party services.

1. **System Testing**
   * Testing the fully integrated application, this is also called end to end scenario testing.
   * Ensures the software meets functional and non-functional requirements.

**4.Acceptance Testing**

* + testing the software by the user or client to determine whether it can be accepted or not.
  + Typically done by end users or stakeholders.
  + Types:
    - **Alpha Testing** – Performed by internal users before release the final product. performed by testers /developers as user.
    - **Beta Testing** – Done by real users in their environment. performed by real users. Direct feedback from customer side is advantage.

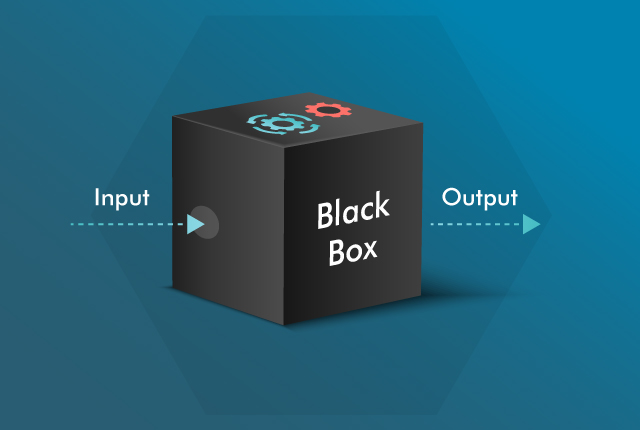
UAT is the final phase of testing where real users validate if the system meets business requirements before going live. It ensures the software works as expected in real-world scenarios.

Each level of testing plays a crucial role in delivering a high-quality software product.

**Testing Methods**

**Black Box (Closed Box) Testing Methods**

Black box testing focuses on testing software functionality without knowing the internal code or logic. It ensures the system behaves as expected.



**Types of Black Box Testing Methods:**

\* Equivalence Class Partitioning

 \*Boundary Value Analysis

\*Decision Table Testing

\*State Transition Testing

\****Experience Based Testing:***

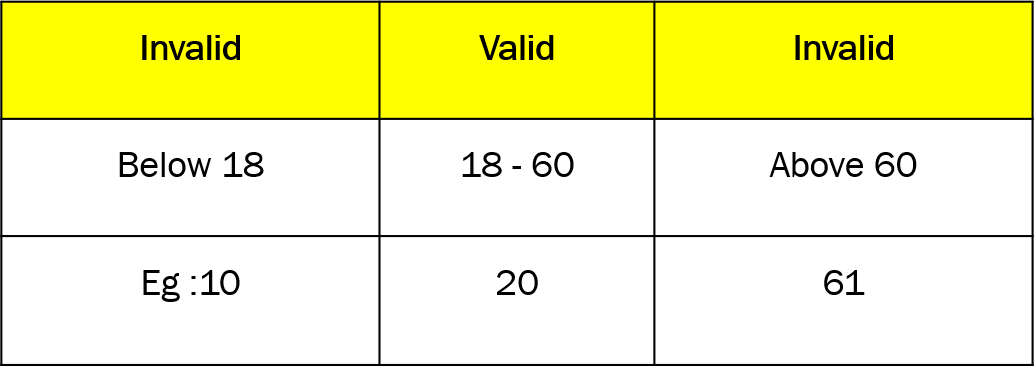
1.Error Guessing.

2.Exploratory Testing.

3.Ad-hoc Testing.

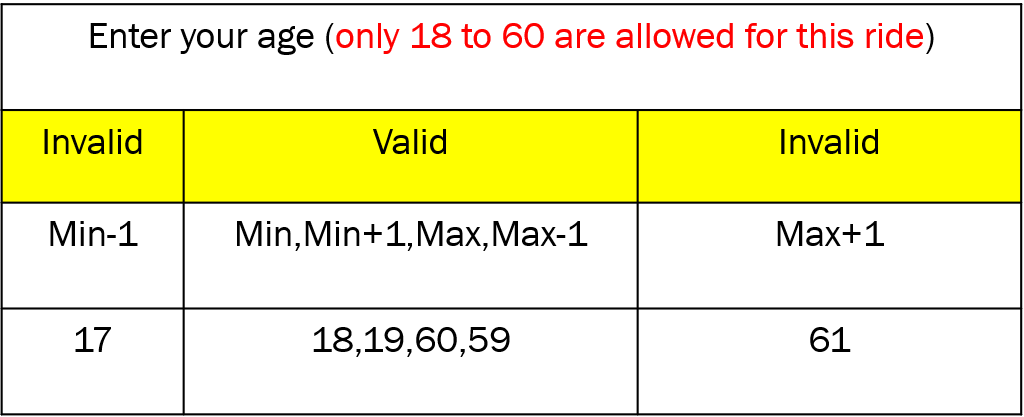
**a. Equivalence Partitioning**Divides input data into valid and invalid groups.

Tests one value from each group instead of all possible values.



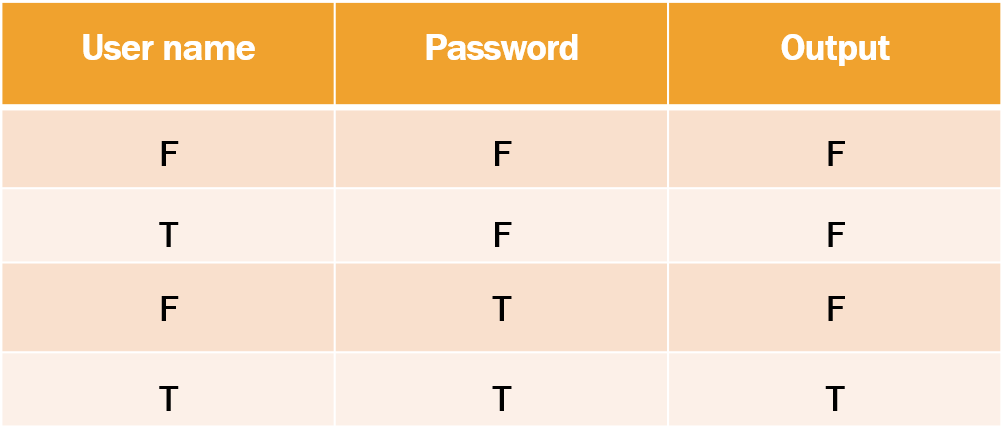
**b. Boundary Value Analysis**

* Tests values at the boundaries (minimum and maximum limits).



**c. Decision Table Testing**

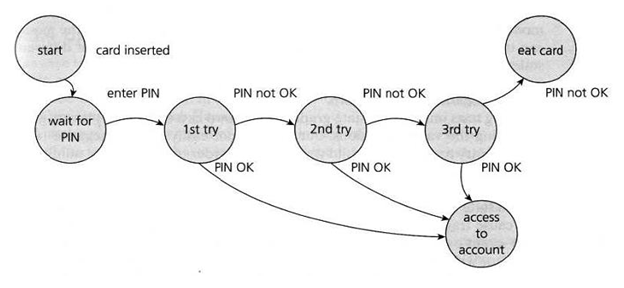
* Uses a table to check different input conditions and expected outputs.
* Example:
  + User name and password fields in a login page



**d. State Transition Testing**Tests how the system behaves when changes in input conditions(both Positive and negative) affect the system's state and output.

A state transition model has four basic parts:(consider the ATM example)  
The **states** that the software may occupy (funded/insufficient funds);  
The **transitions** from one state to another (not all transitions are allowed);  
The **events** that cause a transition (withdrawing money); –> Inputs  
The **actions** that result from a transition (an error message, or being given your cash). –> Output

**The figure shows an example of entering a Personal Identity Number (PIN) to a bank account. The states are shown as circles, the transitions as lines with arrows and the events as the text near the transitions.**

****

***Experience-Based Testing Methods:***

**1. Error Guessing.**Error guessing is when testers use their experience to predict where a system might fail, Instead of following a structured test plan, they think about past mistakes and common errors in similar applications.

**Example:**Imagine you're testing a login page. Based on past experience, you might try:

* **Entering incorrect passwords multiple times.**
* **Using special characters in the username field.**
* **Leaving fields empty and trying to log in.**

These are common areas where errors might occur, so you test them first.

**2. Exploratory Testing**  
Exploratory testing means testing a system without a predefined plan. The tester explores the application, learns about it, and finds defects on the go.

**Example:**  
If you’re testing an e-commerce website, you might:

* **Add a product to the cart, then remove it.**
* **Try to place an order without entering shipping details.**
* **Check how the website works on different browsers.**

This approach is great for finding unexpected issues.

**3. Ad-hoc Testing**  
Ad-hoc testing is unstructured testing where the tester randomly clicks and interacts with the application to find bugs. It doesn’t follow any rules or test cases.

**Example:**  
You're testing a mobile app and randomly:

* **Press buttons quickly to see if it crashes.**
* **Minimize and reopen the app multiple times.**
* **Change phone settings (e.g., turn off the internet) while using the app to see what happens**.

This method is useful for finding issues quickly.

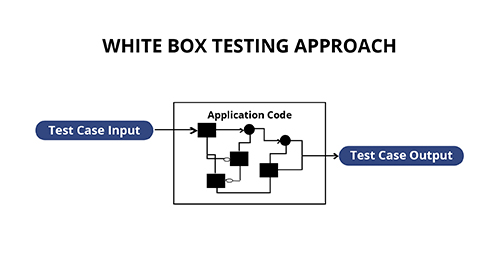
***Summary:***

| **Testing Type** | **What it does** | **Example** |
| --- | --- | --- |
| Error Guessing | Predict common mistakes | Try incorrect passwords |
| Exploratory Testing | Learn and test as you go | Randomly test e-commerce checkout |
| Ad-hoc Testing | Random unstructured testing | Click buttons randomly |

**White Box (Glass Box) Testing Methods**

White box testing is a software testing technique where the internal structure and code are tested. It is done by developers or testers with programming knowledge.

White box testing helps find hidden errors early by analyzing the code directly.



**Methods of White Box Testing:**

Statement Coverage

Decision Coverage

Condition Coverage

Branch Coverage

1. **Statement Coverage**
   * Ensures every line of code is executed at least once.
   * Statement Coverage = No.of executed statement \*100

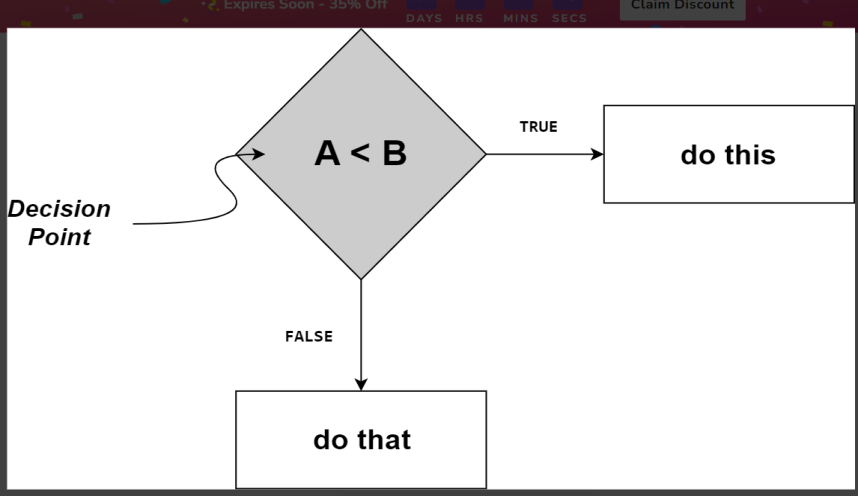
Total no.of statement

**2.Decision Coverage**

check and ensure that each branch of every possible decision point is executed at least once. It ensures that every possible outcome of the entire decision is tested.

Decision coverage= No.of decision executed\*100

                Total number of decision



**3.Condition Coverage**

It is a testing method used to evaluate each variable within a conditional statement. It tests every possible outcome of each sub-expression to ensure that all scenarios are covered.

Example:  if (x > 1 && y > 0) condition coverage requires testing:

x = 2, y = 1

x = 2, y = -1

x = -1, y = 1

x = -1, y = -1

**4.Branch Coverage**

* Tests all possible branches (true/false conditions) in decision-making statements.

***Conclusion:***

* **Statement Coverage:** ensures all lines are executed.
* **Decision Coverage**: ensures true and false conditions are tested.
* **Condition Coverage**: ensures individual conditions are tested separately.
* **Branch Coverage:** ensures all branches (if, else if, else) are executed.

**Grey Box Testing Method**

It is a combination of Black Box and White Box technique.

* + In the Black Box Testing technique, the tester is unaware of the internal structure of the item being tested and

in White Box Testing the internal structure is known to the tester.

* + The internal structure is partially known in Gray Box Testing.
  + the software program is like a semitransparent or gray box inside which the tester can partially see.
  + It is based on requirement test case generation because it has all the conditions presented before the program is tested.

