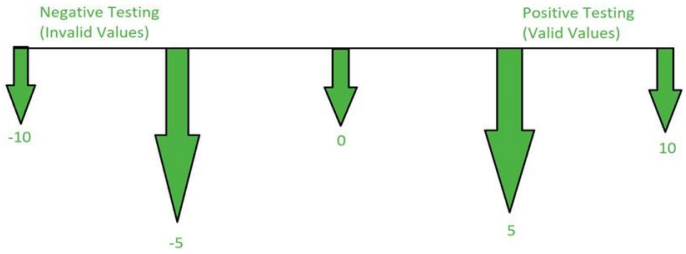
UNIT II – LEVELS OF TESTING

Black Box Testing: Requirements based Testing – Positive and Negative Testing – Boundary Value Analysis – Decision Tables – State Based Testing – Compatibility Testing – User Documentation Testing – Domain Testing. Levels of Testing: System and Acceptance Testing.

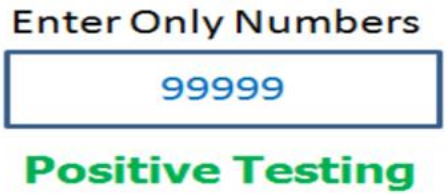
Positive and Negative Testing

Software Testing is the process of evaluating a software application to ensure that it functions according to specified requirements and delivers the expected results. Its primary goal is to identify defects, improve software quality, and ensure reliability. Software testing is generally categorized into two types: **Positive Testing** and **Negative Testing**.



Positive Testing

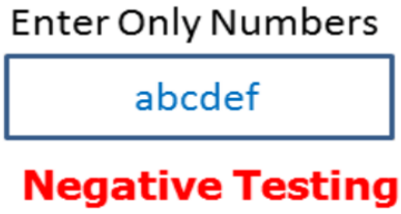
Positive Testing is a software testing technique in which valid input data is provided to verify that the application functions correctly. It ensures that the software behaves as expected and produces the correct output when given valid and expected inputs. The main objective of positive testing is to confirm that the system performs its intended functions successfully.



Example: If a text box accepts only numbers from 0 to 99999, positive testing involves entering valid numbers within this range and verifying that the system accepts them correctly.

Negative Testing

Negative Testing is a software testing technique in which invalid, incorrect, or unexpected input data is provided to verify how the application handles errors. It ensures that the software responds appropriately to invalid inputs, does not crash, and remains stable under unexpected conditions. The main objective of negative testing is to validate the application's error-handling and robustness.



Negative testing can be performed by entering characters A to Z or from a to z. Either software system should not accept the values or else it should throw an error message for these invalid data inputs.

Difference between Positive Testing and Negative Testing:

Positive Testing	Negative Testing
It is performed only for expected conditions.	It is performed for unexpected conditions.
It doesn't cover all possible cases.	It covers all possible cases.
It doesn't ensure a good quality product.	It ensures a good quality product.
It is less important as compared to Negative testing.	It is more important as compared to Positive testing.
It can be performed by people having less knowledge.	It can only be performed by professionals.
It takes less time.	It takes more time.
It is performed on every application.	It is performed where are chances of unexpected conditions.
It ensures software is normal.	It ensures 100% defect free software.

2.4 Decision Tables

What is Decision tables?

A decision table is a systematic and structured method used in software testing and system analysis to represent complex logical relationships between conditions and actions. It provides a visual and tabular representation of different combinations of input conditions and corresponding outcomes or actions. Decision tables help in designing test cases, analyzing decision-making logic, and ensuring comprehensive coverage of various scenarios within a software system.

What is Decision Table Based Testing?

Decision table based testing is a systematic testing technique that uses decision tables to design and execute test cases for software systems. This approach involves creating decision

tables that capture various combinations of input conditions and corresponding actions or outcomes. Testers analyze these decision tables to derive test cases that cover different scenarios based on the defined conditions.

Characteristics of Decision Tables

1. **Conditions**
   Conditions are the input values or criteria that affect a decision. They determine what needs to be checked before taking an action.
2. **Actions**
   Actions are the results or outputs that occur based on the given conditions.
3. **Rules**
   Each rule represents one possible combination of conditions and its corresponding action.
4. **Structured Format**
   Decision tables are arranged in a clear table format with conditions, actions, and rules, making them easy to read and understand.
5. **Complete Coverage**
   Decision tables consider all possible combinations of conditions, ensuring that no scenario is missed.
6. **Visual Representation**
   The tabular format makes complex decision logic simple to understand and analyze.
7. **Scalability**
   Decision tables can handle a large number of conditions and actions, making them useful for both simple and complex systems.
8. **Flexibility**
   Decision tables can be easily updated by adding or modifying conditions, actions, or rules when system requirements change.

Importance of Decision Tables :

Decision tables are useful in software testing because they make decision-making logic clear and help testers create better test cases.

Benefits of Decision Tables

1. **Clarity and Understanding:** They present complex decision logic in a simple table format that is easy to understand.
2. **Comprehensive Coverage**: They consider all possible combinations of conditions, reducing the chance of missing important scenarios.
3. **Better Test Case Design:** Testers can create effective test cases based on different condition combinations.
4. **Identify Dependencies:** They show how conditions and actions are related, helping testers find important interactions.

5. **Validate Logic:** They help verify that the system behaves correctly for different inputs and situations.
6. **Reduce Risk:** They minimize the chance of overlooking critical decision paths or errors.
7. **Improve Communication:** They provide a standard and easy-to-read format that helps developers, testers, and analysts work together.
8. **Scalability and Flexibility:** They can handle both simple and complex systems and can be easily updated when requirements change.

Components of Decision Table

Decision tables follow a structured format that includes specific components to represent the conditions, actions, and rules governing the decision-making process within a system. The design format typically consists of the following sections:

1. Condition Stub:

- **Purpose**: Lists all the conditions relevant to the decision process. Conditions are typically questions or statements that can be answered with Yes (Y), No (N), or Don't Care (-).
- **Location:** Left side of the table.

2. Condition Entries:

- **Purpose:** Contains entries (Y, N, -) corresponding to each condition for every rule defined in the decision table.
- '**Don't Care' Entries:** Sometimes a condition in a particular rule might not affect the

outcome, indicated by a '-' entry. This means that the action does not depend on this

condition's state.

- **Location:** Right of the condition stub, divided by a vertical line.

3. Action Stub:

- Purpose: Lists the possible actions that can be taken based on the evaluation of the

conditions.

- Location: Below the condition stub.

4. Action Entries:

- Purpose: Indicates which actions are to be taken for each rule by marking with an X or

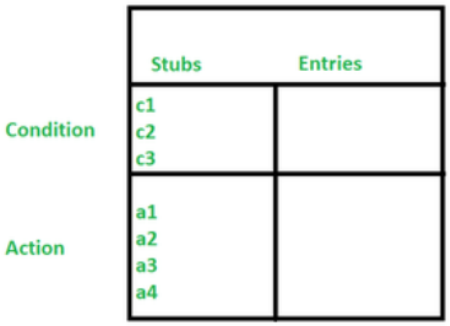
leave blank based on the condition entries above.

- Location: Below the condition entries, divided by a horizontal line.

5. Rules:

- Purpose: Represents a unique combination of condition entries and specifies the resulting actions. Each rule is a feasible scenario that might occur, and the table explores the outcomes.
- Location: Each column in the entry portion of the table

Decision Table Design Format



Example:

How to Create a Login Screen Decision Base Table

A login screen with E-mail and Password Input boxes.

The condition is simple − The user will be routed to the homepage if they give the right username and password. An error warning will appear if any of the inputs are incorrect.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Expalnation:

T - Make sure your login and password are correct.

F - Incorrect login or password

E - An error message appears.

H - The home screen appears.

Case 1 − Both the username and password were incorrect. An error message is displayed to the user.

Case 2 − The username and password were both right, however, the password was incorrect. An error message is displayed to the user.

Case 3 − Although the username was incorrect, the password was accurate. An error message is displayed to the user.

Case 4 − The user's username and password were both accurate, and the user went to the homepage.

Types of Decision Tables

Decision tables are categorized based on the number of conditions and the complexity of the decision logic.

1. **Limited Decision Table:**
   Used when each condition has only two possible values (Yes/No or True/False).
   Example: A login system checks whether the username and password are valid.

2. **Extended Decision Table:**
   Used when conditions can have more than two possible values.
   Example: A loan approval system considers income level, credit score, and debt ratio.

3. **Condition-Action Table:**
   Shows the relationship between different conditions and the actions to be performed.
   Example: An online shopping system provides discounts based on membership type and purchase value.

4. **Switch Table:**
   Used when the decision depends on a single condition with different possible values.
   Example: A traffic signal system changes actions based on whether the light is red, yellow, or green.

5. **Rule-Based Decision Table:**
   Used for complex systems where multiple conditions and business rules determine the final action.
   Example: An insurance system decides policy eligibility based on age, driving history, and location.

Advantages of Decision Tables

1. **Easy to Understand:**
   Decision tables clearly represent decision logic, making it easier for developers, testers, and stakeholders to understand the system.

2. **Complete Test Coverage:**
   They help test all possible condition combinations, including boundary and special cases.

3. **Simplifies Complex Logic:**
   They organize complicated decision-making into a simple and structured table.

4. **Supports Test Automation:**
   Decision tables can be easily used to create automated and repeatable test cases.

5. **Improves Communication:**
   They provide a common format that helps developers, testers, and business analysts communicate effectively.

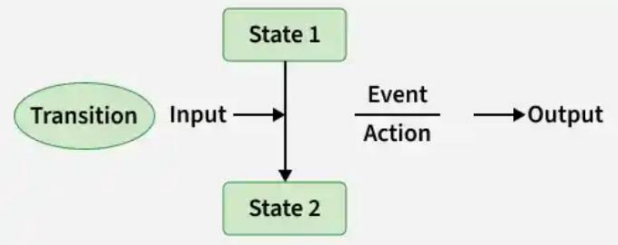
2.5 State Based Testing

State-Based Testing (also called State Transition Testing) is a black-box testing technique that checks how a software system changes from one state to another when different inputs or events occur. It verifies that the system behaves correctly during each state transition and produces the expected output.

- Examines how the system moves from one state to another when different inputs or events occur.
- Confirms that the system correctly handles both valid and invalid state transitions.
- Checks whether the system reaches the correct starting, intermediate, and ending states.
- Tests the system's behavior for different sequences of events, including exceptional and boundary conditions.

State Transition

A **State Transition Diagram** is a graphical model that illustrates how a system moves from one **state** to another when specific **inputs or events** occur. It helps testers and developers understand the system's behavior by showing the different states and the transitions between them under various conditions.



State Transition Diagram

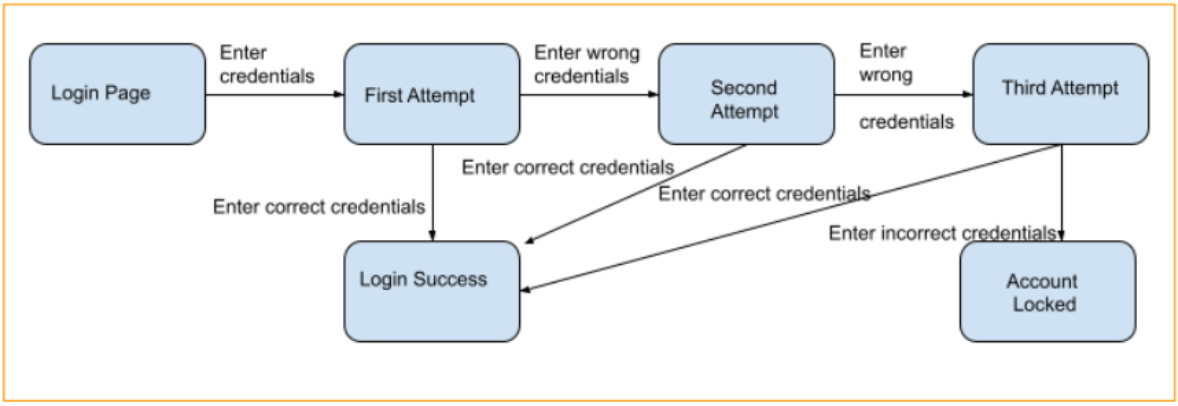
Components of a State Transition Diagram

- **States:** The different conditions or situations in which the system can be at a given time.
- **Transitions:** The change of the system from one state to another.
- **Events:** The inputs or actions that trigger a state transition.
- **Actions:** The tasks or responses performed by the system during a state transition.

Example

Banking application where we would create a state transition diagram on login module.

- User enters **correct credentials on the first attempt** → User logs into the banking system.
- User enters **correct credentials on the second attempt** (after one failed attempt) → User logs into the banking system.
- User enters **correct credentials on the third attempt** (after two failed attempts) → User logs into the banking system.
- User enters **incorrect credentials on four consecutive attempts** → User account is locked.



The total number of test cases that can be designed from the above state transition diagram are:

Test Case 1 − User is in the Login Page, and then he enters the correct credentials in the first attempt, resulting in the successful login.

Test Case 2 − User is in the Login Page, and then he enters the correct credentials in the second attempt, resulting in the successful login.

Test Case 3 − User is in the Login Page, and then he enters the correct credentials in the third attempt, resulting in the successful login.

Test Case 4 − User is in the Login Page, and then he enters the incorrect credentials in the third attempt, resulting in the account locked.

Advantages of State Transition Testing

1. **Improves Understanding:**
   It helps create a state transition diagram that clearly shows all the states and transitions of the software, making it easier to understand and document.
2. **Supports Better Test Case Design:**
   It helps design effective test cases by covering both valid (positive) and invalid (negative) state transitions.
3. **Finds Defects Early:**
   It helps identify errors in state changes during the early stages of the **Software Development Life Cycle (SDLC)**, reducing the cost of fixing defects later.

Disadvantages of State Transition Testing

1. **Some States May Be Missed:**
   It is possible to overlook certain states of the software during testing.
2. **Does Not Test All Input Combinations:**
   It focuses on state changes and may not cover every possible combination of input values.
3. **Incomplete Test Coverage:**
   If any state or transition is missed, the testing becomes incomplete and some defects may remain undetected.