### **What is Decision Table Testing?**

Decision table testing is used to identify different combinations of inputs and their corresponding actions. It’s especially useful when testing **business rules** or **complex logic**.

Real-Time Examples of Decision Table-Based Testing

**1. Online Shopping – Discount Logic**

**Scenario**:  
A customer gets a discount based on two conditions:

* Is the customer a **member**?
* Is the **purchase amount > ₹5000**?

|  |  |  |
| --- | --- | --- |
| **Member** | **Amount > ₹5000** | **Discount** |
| No | No | 0% |
| No | Yes | 5% |
| Yes | No | 10% |
| Yes | Yes | 15% |

**Note**: Helps testers verify that the correct discount is applied for each case.

**2. ATM Machine – Cash Withdrawal Rules**

**Scenario**:

* The user must enter the correct **PIN**.
* The **account must have sufficient balance**.

|  |  |  |
| --- | --- | --- |
| **PIN Valid** | **Sufficient Balance** | **Action** |
| No | No | Show Error |
| No | Yes | Show Error |
| Yes | No | Show Insufficient |
| Yes | Yes | Dispense Cash |

**Note**: Ensures secure and correct ATM operations.

**3. Login Functionality**

**Scenario**:  
A user can log in if:

* **Username** is valid.
* **Password** is correct.
* **Captcha** is correct.

|  |  |  |  |
| --- | --- | --- | --- |
| **Username** | **Password** | **Captcha** | **Result** |
| Invalid | Any | Any | Login Fail |
| Valid | Invalid | Any | Login Fail |
| Valid | Valid | Invalid | Login Fail |
| Valid | Valid | Valid | Login Pass |

**Note**: Helps QA ensure all combinations of login conditions are tested.

### **What is State Transition Testing?**

**State Transition Testing** is a black box technique used when a system behaves **differently based on its current state** and **user inputs/events cause transitions between states**. It helps testers validate whether the application behaves correctly when moving from one state to another.

### **Key Concepts:**

* **States**: Different conditions the system can be in.
* **Events/Inputs**: Actions that cause state changes.
* **Transitions**: Movement from one state to another.
* **Actions**: Outputs or changes that occur after a transition.

### **1.Real-Time Example: ATM Machine**

Let’s say we’re testing an **ATM Withdrawal Flow**.

#### ****States:****

1. Idle
2. Card Inserted
3. PIN Entered
4. Transaction Selected
5. Cash Dispensed
6. Card Ejected

#### ****Transitions & Events:****

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Event/Input** | **Next State** | **Action** |
| Idle | Insert Card | Card Inserted | Show PIN screen |
| Card Inserted | Enter PIN | PIN Entered | Validate PIN |
| PIN Entered | Select Withdrawal | Transaction Selected | Show account balance |
| Transaction Selected | Enter Amount | Cash Dispensed | Dispense cash |
| Cash Dispensed | Take Cash | Card Ejected | Eject card |

This technique helps ensure that:

* No invalid state transitions happen (e.g., trying to dispense cash without inserting card).
* The system returns to the correct state after an operation.
* All possible valid and invalid transitions are covered.

### **2.Real-Life Example: Traffic Light System**

#### ****States:****

* Red
* Green
* Yellow

|  |  |  |
| --- | --- | --- |
| **Current State** | **Timer Event** | **Next State** |
| Red | Timer Up | Green |
| Green | Timer Up | Yellow |
| Yellow | Timer Up | Red |

### **Real-Time Examples of Error Guessing**

Error guessing is all about thinking **"Where could this break?"** or **"What would a user do that’s unexpected?"**. It complements other structured techniques by catching edge cases and sneaky bugs.

**1. Login Page**

**Error Guesses:**

* Entering **blank username/password**.
* Using **SQL Injection**: e.g., admin' OR 1=1--.
* Entering **long strings** to check buffer overflows.
* Typing **special characters or emoji’s**.

**2. Form Validation**

**Error Guesses:**

* Submitting form **without filling mandatory fields**.
* Entering **invalid email** format like user@@mail.
* Adding **JavaScript code** in input fields to test for script injection.
* Entering **future date** in DOB field.

**3. E-commerce Checkout**

**Error Guesses:**

* Applying **invalid or expired coupon codes**.
* Clicking **"Place Order" button multiple times** quickly (to check for duplicate orders).
* Trying to add **negative quantity** of a product.
* Skipping payment but still proceeding.

**4. File Upload Feature**

**Error Guesses:**

* Uploading **unsupported file formats**.
* Uploading **very large files** beyond the limit.
* Uploading **files with harmful extensions** (like .exe, .bat).
* Leaving upload in progress and refreshing the page.

### **What is Equivalence Class Partitioning (ECP)?**

**Equivalence Class Partitioning** is a **black box testing technique** where input data is **divided into partitions or classes**. Each class represents a set of values that are expected to be treated the **same way by the system**. So instead of testing every value, you just test **one value from each class**.

### **Why use it?**

* Reduces the total number of test cases.
* Ensures broad coverage with minimal effort.
* Helps focus on **valid and invalid input ranges**.

### **Types of Partitions:**

1. **Valid Equivalence Class** – Expected input values.
2. **Invalid Equivalence Class** – Unexpected or erroneous input values.

### **Real-Time Example:**

### **1. Age Validation Field**

**Scenario**:  
A form accepts **age between 18 and 60**.

#### Equivalence Classes:

|  |  |  |
| --- | --- | --- |
| **Type** | **Input Range** | **Example Test Value** |
| Invalid | Less than 18 | 17 |
| Valid | 18 to 60 | 30 |
| Invalid | Greater than 60 | 65 |

You don’t need to test **every number from 1 to 100**, just pick one value from each class.

### **2. Email Field**

**Rule**: System accepts a **valid email format**.

#### Equivalence Classes:

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example** |
| Valid | Proper email format | ravilella@gmail.com |
| Invalid | Missing '@' symbol | userexample.com |
| Invalid | Missing domain | user@ |
| Invalid | Blank input | (empty) |

### **Summary:**

Equivalence Partitioning is like **testing smart, not hard** — if one value in a class behaves correctly, it's assumed others will too.

### **What is Boundary Value Analysis (BVA)?**

**Boundary Value Analysis (BVA)** is a **black box testing technique** that focuses on testing the **edges (boundaries)** of input ranges where defects often occur. Instead of testing a wide range of values, we test values **at, just below, and just above the boundaries**.

### **Why use it?**

* Most errors happen **at the edges** of input ranges.
* Helps uncover **off-by-one errors** (like using < instead of <=).
* Simple yet highly effective for numeric inputs, date fields, and ranges.

### **Real-Time Example:**

### **Age Input Field**

**Scenario**:  
An online job application form accepts age between **18 to 60 years** (inclusive).

#### Test cases using BVA:

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Test Value** | **Expected Result** |
| Just below lower boundary | 17 | Rejected (Invalid) |
| At lower boundary | 18 | Accepted (Valid) |
| Just above lower boundary | 19 | Accepted (Valid) |
| Just below upper boundary | 59 | Accepted (Valid) |
| At upper boundary | 60 | Accepted (Valid) |
| Just above upper boundary | 61 | Rejected (Invalid) |

You test: **17, 18, 19, 59, 60, 61**

### **Password Length Rule**

**Rule**: Password must be between **8 and 16 characters**.

|  |  |  |
| --- | --- | --- |
| **Test Condition** | **Test Value Length** | **Expected Result** |
| Just below lower bound | 7 | Rejected (Too short) |
| At lower bound | 8 | Accepted |
| Just above lower bound | 9 | Accepted |
| Just below upper bound | 15 | Accepted |
| At upper bound | 16 | Accepted |
| Just above upper bound | 17 | Rejected (Too long) |