Test Scenario Creation + Introduction to Test Design Technique

Why?

Test Scenario Creation:

To ensure comprehensive coverage of application functionality by identifying all possible testable conditions.

Simplifies communication with stakeholders by outlining high-level functionalities.

Acts as a foundation for creating detailed test cases.

Test Design Techniques:

To systematically derive effective and efficient test cases.

Helps in finding defects early by identifying key testing areas.

Reduces redundancy and improves testing focus.

What?

Test Scenario Creation:
The process of identifying high-level testable functionalities or features.
Focuses on **"what to test"**, not the detailed steps of **"how to test"**.
Example:
Feature: Login functionality.
Test Scenario: Validate user login with valid credentials.
Test Design Techniques:
Structured approaches for creating optimized test cases.
Types include:
Black-Box Techniques: Focus on functionality (e.g., Equivalence Partitioning, Boundary Value Analysis).
White-Box Techniques: Focus on code structure (e.g., Statement Coverage, Branch Coverage).
Experience-Based Techniques: Leverage intuition and domain knowledge (e.g., Exploratory Testing, Error Guessing).

Where?

Test Scenario Creation:

During the **test planning phase** to define high-level functionalities to test.

Applied across all features and functionalities of an application.

Example: In an e-commerce application, scenarios like "Validate checkout process" or "Validate search functionality."

Test Design Techniques:

Used during the **test case design phase** to generate detailed and efficient test cases.

Applied in various contexts:

Black-Box Techniques: For user-facing functionalities (e.g., login, payment processing).

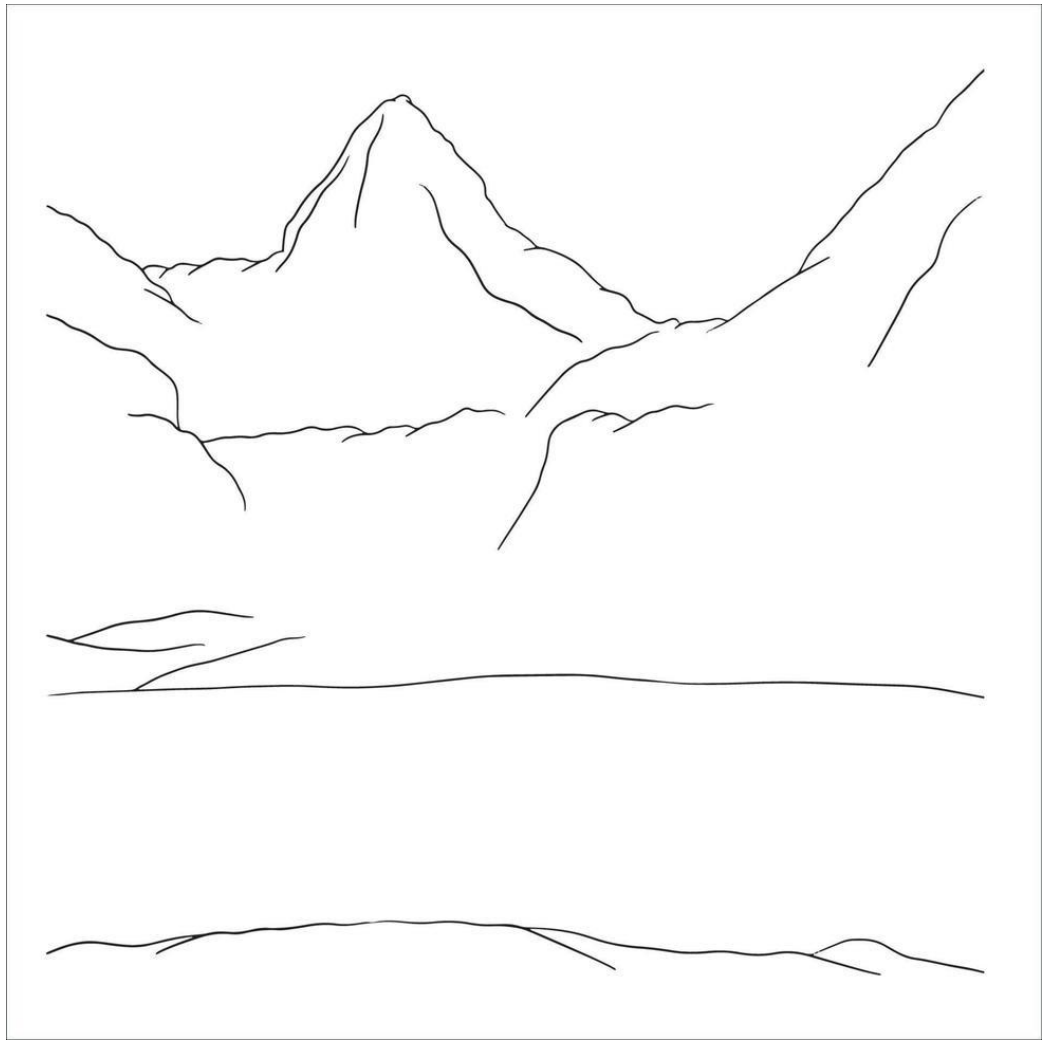
White-Box Techniques: For internal code-level testing (e.g., API testing, algorithm validation).

Experience-Based Techniques: For exploratory or ad-hoc testing (e.g., testing error-prone areas).

Introduction

A **Test Scenario** is a high-level description of a functionality or feature to be tested.

It focuses on what to test .



How to create a Test Scenario?

Creating effective test scenarios ensures comprehensive testing of an application. Here's a step-by-step description:

1. Study the Requirements

Understand the Documents: Read and analyze business requirements, functional requirements, and technical specifications.

Clarify Doubts: If any details are unclear, consult with stakeholders, developers, or team members.

2. Isolate Features and Actions

Identify Features: Break down the application into distinct features (e.g., Login, Search, Checkout).

List User Actions: For each feature, think about all possible actions a user can take (e.g., entering valid/invalid credentials for Login).

3. Align with Requirements

Requirement Matching: Ensure every test scenario corresponds to a requirement in the documentation. This avoids missing any functionality during testing.

Examples:

Requirement: "Users must log in with valid credentials."

Scenario: Validate user login with valid credentials.

4. Write Clear and Concise Scenarios

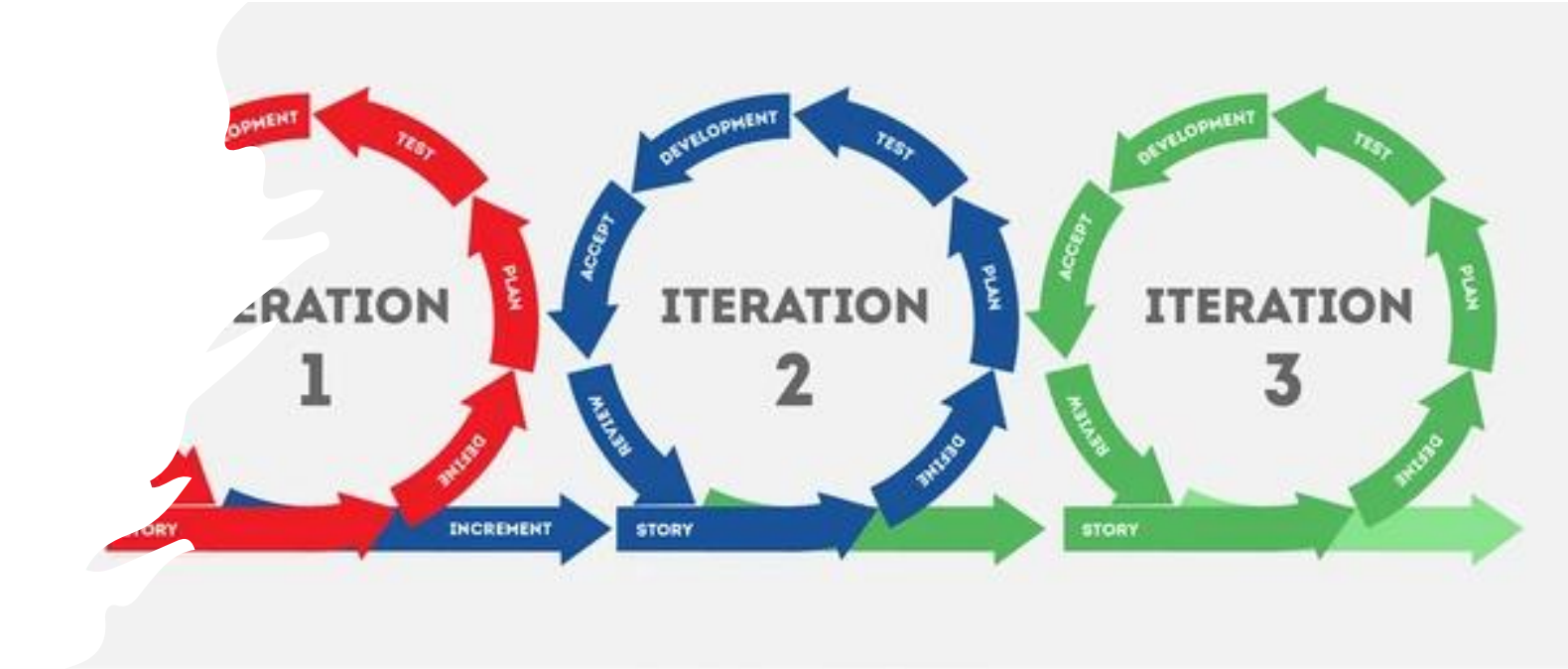
High-Level Description: Focus on **"what to test"** rather than detailed steps of **"how to test."**
Template Example:
Feature: Login
Scenario: Validate user login with invalid credentials.

5. Review and Collaborate

Feedback: Share the test scenarios with your supervisor, team leader, or stakeholders.
Validation: Get confirmation that the scenarios cover all required functionalities.

6. Iterate and Improve

Update as Needed: Revise scenarios as features change or new requirements arise.



Example: E-commerce Website - Checkout Feature

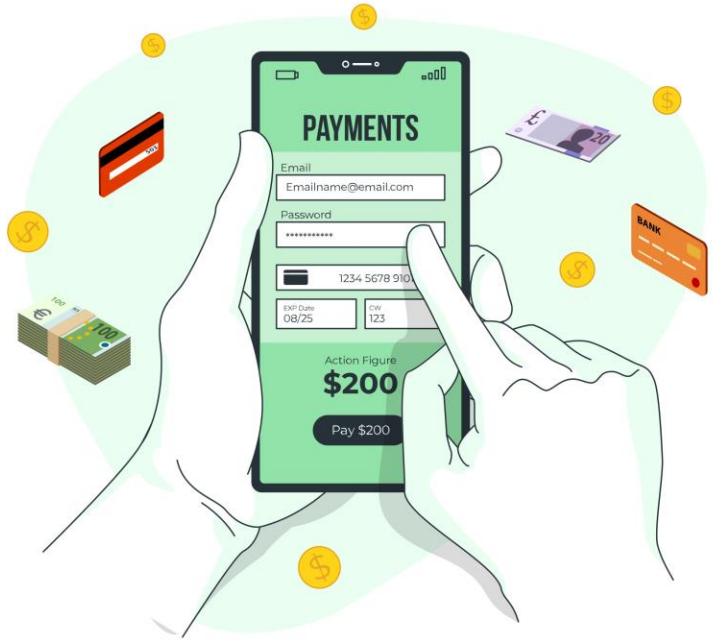
Feature: Checkout

Test Scenarios:

Validate successful checkout with valid payment details.

Validate error message when entering invalid card details.

Validate the option to apply discount codes during checkout.



Write the Test scenario of an e-commerce website - https://blinkit.com/

When Test Scenarios are not applicable?

Creating test scenarios is crucial in testing, but there are situations where it might not be necessary or efficient. Here's when to skip creating them:

1. Limited Time or Resources

Why: When deadlines are tight, or resources are scarce, focusing on quick ad hoc testing might be more practical than spending time drafting scenarios.

Example: A minor UI that needs validation.

Low-Risk Functionalities

Why: If the feature being tested has minimal impact on the system or user, detailed test scenarios may not justify the effort.

Example: Testing the font style of text on a webpage.

Advantages of Test Scenario

Covers the entire functionality of the software

Simulates a real-life situation from an end-user point of view

Significant time saver especially for agile teams

Equivalence Partitioning

Equivalence Partitioning is a software testing technique used in manual testing to systematically divide the input data of a system into groups or partitions that are expected to exhibit similar behavior. The goal is to test representative values from each partition to ensure that the software functions correctly and to optimize test coverage.

Examples of Equivalence Partitioning:

Example 1: Input Field Accepting Numbers from 1 to 100

Valid Partition: 1 to 100

Test case: Input = 50

Invalid Partition:

Below the range: Test case: Input = 0

Above the range: Test case: Input = 101

Non-numeric input: Test case: Input = "abc"

Boundary Value Analysis

Boundary Value Analysis (BVA) is a testing technique that focuses on testing the boundaries of input values for a given system. It is widely used in software testing to ensure that the application functions correctly at the edges of the input domain

Key Concepts:

Boundary Values:

Lower Bound: The smallest valid input value.

Upper Bound: The largest valid input value.

Edge Values: The values just above and below the boundaries.

Test Cases:

BVA involves testing at the boundaries and just beyond them to ensure robustness.

Test cases are designed for values at the lower bound, upper bound, and edges.

Example:

If a system accepts input values between 1 and 100, the boundary values would be 1, 100, 0 (just below the lower bound), and 101 (just above the upper bound).

Advantages:

Efficient at catching errors around boundaries.

Reduces the likelihood of defects in critical areas of the application.

Limitations:

Does not cover all possible scenarios.

May not be effective for non-numeric inputs.

Steps to Perform Boundary Value Analysis:

- **Identify Input Boundaries:**
- Determine the valid input range for each parameter.
- **Select Test Values:**
- Choose values at the lower bound, upper bound, and edges.
- **Create Test Cases:**
- Develop test cases using the selected values.
- **Execute Tests:**
- Execute the test cases to observe the system's behavior.
- **Analyze Results:**
- Evaluate the results to identify any boundary-related issues.

Decision Table Testing

“Decision Table Testing” is a black box test design technique in which test cases are designed to execute the combinations of inputs shown in a decision table. Decision table testing is a good way to deal with combinations of inputs.

Example 1 – How to Create a Login Screen Decision Base Table

Let's make a login screen with a decision 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.

Explanation

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.

State Transition Testing

The state transition testing is conducted to verify the change in states of a software under varying inputs. If the circumstances under which the provided input are modified, then there are updates to the states of the software.

The software state transition testing comes under the black box testing and is performed to verify the characteristics of the software for multiple sets of input conditions that are fed to it in a specific order. It includes verification of both positive and negative flows. This type of testing is adopted where various state transitions of the software need to be verified.

Example

Let us take an example of a banking application where we would create a state transition diagram on login module features listed below –

- User enters correct credentials in the first attempt, user logs into the banking system.
- User enters correct credentials in the second attempt, user logs into the banking system.
- User enters correct credentials in the third attempt, user logs into the banking system.
- User enters correct credentials in the four attempts, user credentials are locked.


THANK
YOU!