Learning Objectives

By the end of this lesson, you will:

- Understand what test scenarios are and how to create them.
- Learn the importance of Test Design Techniques in ensuring comprehensive testing.
- Explore common test design techniques with examples.

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).

What is a Test Scenario?

Analogy
Imagine you are planning a road trip. A Test Scenario is like the high-level idea of the destinations and checkpoints you want to visit, without detailing the specific steps to get there. It focuses on what to do rather than how to do it.

A Test Scenario is a high-level description of a functionality or feature to be tested, outlining the conditions under which the test will be performed. It focuses on what to test rather than how to test, serving as a guide for test case creation and ensuring comprehensive coverage of the application.

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. Use Visual Aids

Prototypes and Wireframes: If available, use UI prototypes or mockups to understand the design and flow of the application.
Better Test Coverage: Visual aids help you imagine user interactions and identify edge cases.

---

4. 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.

---

5. 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.

---

6. 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.

---

7. Iterate and Improve

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

Let’s Put It Into Practice

Example: E-commerce Website - Checkout Feature

1. Feature: Checkout
2. 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.

When Not to use Test Scenarios?

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. Simple Systems

Why: If the system is basic, with no complex interactions or dependencies, creating detailed scenarios is excessive.
Example: Testing a calculator app with straightforward functions like addition or subtraction.

---

2. 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 tweak that needs immediate validation.

---

3. 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.

---

4. Preference for Exploratory Testing

Why: When exploratory testing is more suitable, testers can use their intuition and experience to find unexpected bugs that predefined scenarios might miss.
Example: Testing a newly integrated third-party API where behavior might vary widely.

Advantages of Test Scenario

Covers the entire functionality of the software
Simulates a real-life situation from an end-user point of view
Helps testing teams control the testing processes
Easy to create and maintain
Significant time saver especially for agile teams

Best Practices to Create Test Scenarios

Crafting effective test scenarios is essential for ensuring software quality and reliability. Here are some best practices to follow for thorough application testing:

1. Understand the Requirements: Before writing test scenarios, gain a comprehensive understanding of the software requirements. This ensures your scenarios encompass all necessary functionalities and user stories.

2. Define Clear Objectives: Each test scenario should have a specific objective. Clearly state what aspect of the software is being tested, whether it's a particular function, performance metric, or user experience feature.
3. Keep Scenarios Simple and Concise: Avoid unnecessary complexity. Each scenario should be straightforward enough to understand and execute without ambiguity, making it easier to pinpoint issues if a test fails.
4. Prioritize Test Scenarios: Recognize that not all scenarios carry the same weight. Prioritize them based on user impact, functionality criticality, and the likelihood of failure.
5. Include Positive and Negative Test Cases: Ensure your scenarios address both positive paths (normal operating conditions) and negative paths (error conditions and edge cases).
6. Ensure Reusability and Maintainability: Write test scenarios with an eye toward reusability in future testing cycles, which can save time and effort over the long term.
7. Automate When Feasible: Consider automating repetitive and high-volume test scenarios to enhance efficiency and consistency in your testing processes.
8. Review and Update Regularly: As the software evolves, so should your test scenarios. Regularly review and update them to keep them relevant and effective.
9. Collaborate and Communicate: Foster collaboration among team members, including developers, testers, and business analysts, to create robust and effective test scenarios together.

3.1 Test Design Principles

Test design principles are guidelines and strategies used by software testers to create effective and efficient test cases. These principles help ensure that the testing process is thorough, covers relevant scenarios, and identifies defects in the software. Here are some important test case design principles along with examples:

3.1.1. 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.

Here's an explanation of Equivalence Partitioning with examples:

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"

Example 2: Password Field (6 to 12 Characters)

Valid Partition: 6 to 12 characters
    Test case: Input = "pass123"
Invalid Partition:
    Less than 6 characters: Test case: Input = "abc"
    More than 12 characters: Test case: Input = "longpassword123"
    Empty input: Test case: Input = ""

Example 3: Dropdown Menu with Options (Red, Green, Blue)

Valid Partition: Options in the dropdown
    Test case: Input = "Red"
Invalid Partition:
    Options not in the dropdown: Test case: Input = "Yellow"
    Empty input: Test case: Input = ""

3.1.2 Boundary Value Analysis

Title: Boundary Value Analysis in Test Design

Introduction: Boundary Value Analysis (BVA) is a black-box 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. By examining values at the extremes, BVA helps identify potential errors and vulnerabilities that may occur near the boundaries.

Objective of Boundary Value Analysis: The primary goal of Boundary Value Analysis is to discover errors in the boundary conditions of input values, as these are often more prone to defects. This testing technique is particularly effective in uncovering issues related to incorrect data handling, array bounds, and other boundary-dependent functionalities.

Key Concepts:

1. 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.
2. 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.
3. 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).
4. Advantages:
   - Efficient at catching errors around boundaries.
   - Reduces the likelihood of defects in critical areas of the application.
5. Limitations:
   - Does not cover all possible scenarios.
   - May not be effective for non-numeric inputs.

Steps to Perform Boundary Value Analysis:

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

Best Practices:

1. Include Invalid Values:
   - Test cases should also include values just outside the valid range to catch potential errors.
2. Combine with Other Techniques:
   - BVA is most effective when combined with other testing techniques for comprehensive coverage.
3. Document Test Cases:
   - Maintain detailed documentation of test cases for future reference and regression testing.

Boundary Value Analysis is a powerful testing technique that aids in detecting defects near the boundaries of input domains. By systematically testing at the edges and just beyond, it helps ensure

the robustness and reliability of software applications. When incorporated into a comprehensive testing strategy, BVA contributes significantly to delivering high-quality software.

3.1.3 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.

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

T - Make sure your login and password are correct.
F - Incorrect login or password
E - An error message appears.
H - The home screen appears.

Interpretation

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.

We may generate two situations when converting this to a test case.

Enter the right username and password and click Login; the intended consequence is that the user will be sent to the homepage.

And one from the situation below.

If the user types in the erroneous username and password and then clicks Login, the user should receive an error message.
When you provide the proper username and incorrect password and click Login, the user should see an error message.
If the user types the erroneous username and password and then clicks Login, the user should receive an error message.

3.1.4 State Transition Testing

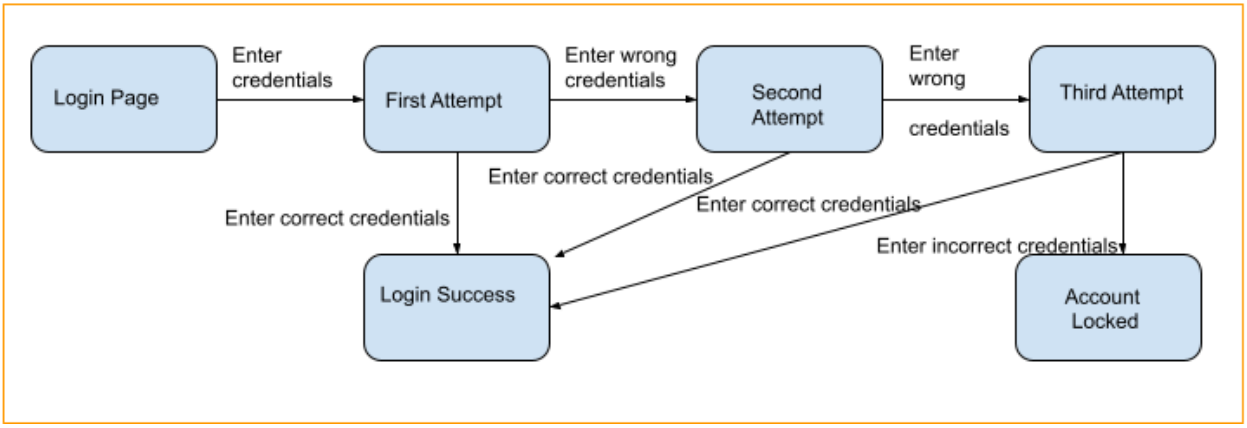
The state transition testing is conducted to verify the change in states of a software under varying inputs. If the circumstances under which the provided input are modified, then there are updates to the states of the software.

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.



3.6 Use Case Testing

It helps to identify test cases that cover entire system on a transaction by transaction basis from start to end. Test cases are the interactions between users and software application. Use case testing helps to identify gaps in software application that might not be found by testing individual software components.

Features of Software Use Case Testing

The features of the software use case testing are listed below −

- The use case testing mainly works on the user interactions, and scenarios with the software. It verifies the outputs generated by software with a specific set of user inputs.
- The use case testing uses all the user interactions with the system as a framework for managing the tests. It confirms that the software is able to function at par in order to support the users in the real environment.
- The use case testing ensures that the software has all the necessary specifications to validate all the use cases. It also verifies if the functional requirements are working correctly.
- The use case testing involves both positive and negative testing which identifies issues in different circumstances.
- The use case testing supports the integration testing by identifying defects while various modules, and sub-modules communicate with each other.