Learning Objectives

By the end of this lesson, you will:

- Understand the purpose and process of test case creation.
- Learn about the key components of a test case and how to write effective test cases.
- Explore the principles of experience-based test design techniques.
- Understand when and how to apply experience-based methods for effective defect detection.

Why? (Purpose of the Technique)

- To ensure comprehensive, consistent, and repeatable testing.
- To fill gaps in documentation, find subtle bugs, and leverage expertise.

What? (What It Is)

- Writing structured test cases with steps, data, and expected outcomes.
- Testing using intuition, error prediction, and real-time exploration.

Where? (Where It Is Applied)

- Functional, regression, and system testing.
- New features, high-risk areas, and dynamic environments.

Introduction

Experience-based test design techniques leverage the tester's past experience, intuition, and knowledge to identify defects. These techniques complement structured methods by addressing gaps in requirement documentation and discovering subtle issues.

Together, test case creation and experience-based techniques form a balanced approach for comprehensive software testing.

Test case creation + experience based testing = balanced approach for effective testing

Key Concepts and Definitions

Test Case

A test case is a structured document containing inputs, execution conditions, and expected results for testing a specific functionality of a system.

Key Components of a Test Case:

1. Test Case ID: A unique identifier for tracking.
2. Test Scenario: A high-level idea of what is being tested.
3. Test Steps: Sequential actions required to execute the test.
4. Test Data: Inputs required to carry out the test.
5. Expected Result: The predicted behavior of the system under test.
6. Actual Result: The observed behavior during execution (filled in later).
7. Pass/Fail Status: Indicates whether the test case was successful.

Types of Experience-Based Techniques

Error Guessing

Error guessing is an experience-based software testing technique where testers anticipate possible errors based on their knowledge, intuition, and previous experience with similar systems. Unlike structured approaches (e.g., boundary value analysis or equivalence partitioning), error guessing relies on creativity and expertise to identify defects.

Why Use Error Guessing?

To detect defects missed by formal testing techniques.
To address undocumented edge cases or unusual scenarios.
To test "what could go wrong" based on past patterns of failure.

Example 1: Login Form

User enters an empty username or password.
Username exceeds the maximum allowed length.
Password contains unsupported special characters.

Test Cases:

1. Input: Username = `null`, Password = `password123`.
2. Input: Username = `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa` (50+ characters).

3. Input: Username = `admin`, Password = `!@#$%^&*()`.

Example 2: E-Commerce Checkout

Interrupt the payment process (e.g., close the browser mid-transaction).
Test with invalid credit card numbers.
Use expired coupons or promotions.

Test Cases:

1. Perform checkout and close the tab during payment.
2. Enter a card number with fewer than 16 digits.
3. Apply a coupon code that expired last month.

Exploratory Testing

Exploratory Testing is an unscripted and dynamic testing approach where the tester actively explores the software application to identify defects. Unlike traditional testing methods that rely on predefined test cases, exploratory testing leverages the tester's experience, intuition, and creativity to discover issues in real time.

Test while exploring the application, adjusting focus based on observations.

Characteristics of Exploratory Testing

Unscripted: No predefined steps or expected results; testers design and execute tests on the fly.

Adaptive: Changes focus based on system behavior and observed defects.

Intuition-Driven: Relies on the tester’s experience, creativity, and understanding of the application.

When to Use Exploratory Testing?

When requirements or documentation are incomplete or ambiguous.

During initial testing phases to quickly understand the application’s behavior.

To supplement structured testing techniques for uncovering subtle defects.

In Agile or rapid development cycles where time for formal test planning is limited.

Examples of Exploratory Testing

1. Login Feature

Test valid and invalid login credentials.

Leave the password field empty and submit.

Enter an email address as the username to check error handling.2. E-commerce Checkout

Add items to the cart, proceed to checkout, and close the browser midway.

Apply expired or invalid promo codes to test error handling.

Test concurrent orders from two different accounts using the same inventory item.

Checklist-Based Testing

Checklist-based testing is a systematic approach to software testing where testers validate functionality using a predefined list of test items or scenarios. This list ensures that all critical aspects of the application are covered during testing, reducing the likelihood of missing defects.
Example Checklist for Web App:
- Input field validation.
- Cross-browser compatibility.
- Proper error message display.

Characteristics of a Good Checklist

Comprehensive: Covers all critical aspects of the application.

Organized: Categorized by features, functionalities, or workflows.

Clear and Concise: Easy to understand and follow.

Customizable: Adapted to the specific requirements of the application or project.

When to Use Checklist-Based Testing?

Regression Testing: To ensure existing features still work after changes.

Exploratory Testing: To guide dynamic testing sessions.

Time-Constrained Testing: When there is limited time for test design.

Examples of Checklist-Based Testing

Example 1: Web Application Testing

Checklist:

Verify all input fields accept valid data.

Validate proper error messages are displayed for invalid inputs.

Test responsiveness on different screen sizes and devices.

Check all links and buttons for proper navigation.

Ensure session timeout functionality works as expected.

Verify cross-browser compatibility (e.g., Chrome, Firefox, Safari).

Example 2: E-commerce Application Testing

Checklist:

Validate product search functionality.

Test adding and removing items from the cart.

Verify checkout process with valid and invalid payment details.

Ensure proper application of promo codes.

Check order confirmation emails are sent correctly.

Test product filtering and sorting features.

3.Session-Based Testing

Session-Based Testing (SBT) is a structured yet exploratory testing approach that divides the testing process into focused, time-boxed sessions. This method aims to combine the flexibility of exploratory testing with a clear framework for tracking and reporting results.

Key Features of Session-Based Testing

Time-Boxed Sessions:

Testing is performed in predefined blocks of time, typically lasting 60–120 minutes.

Charters:

Each session is guided by a test charter, which outlines the specific focus area or goal for the session.

Simultaneous Test Design and Execution:

Testers create and execute test cases in real-time based on the application’s behavior and the defined charter.

Session Reports:

Detailed notes and observations are recorded during the session to document the outcomes, issues, and next steps.

Accountability:

Sessions are reviewed by leads or managers to ensure alignment with project goals and priorities.

Example of a Session-Based Test Charter

Charter:

“Test the checkout functionality of an e-commerce application.”

Session Details:

Duration: 90 minutes

Focus Areas:

Valid and invalid payment methods.

Coupon code application.

Address validation.

---

4. Ad-Hoc Testing

Ad-Hoc Testing is an informal and unstructured testing method where the tester explores the application without predefined test cases or documentation. The goal is to identify defects by exploring the software randomly, leveraging the tester's intuition, creativity, and domain knowledge.

Key Features of Ad-Hoc Testing

No Predefined Plan:

The testing process is unscripted, with no prior test case preparation.

Exploratory in Nature:

Relies on the tester's knowledge, experience, and understanding of the system to identify defects.

Random and Unpredictable:

Testers intentionally attempt unusual or extreme scenarios to uncover bugs.

Examples of Ad-Hoc Testing Scenarios

Scenario: Testing an e-commerce website

Try adding the same item to the cart repeatedly to see if the cart updates correctly.

Enter special characters in the search bar to check how the system handles unexpected inputs.

Test checkout with an expired credit card.

Scenario: Testing a login module

Enter multiple incorrect passwords to check for account lockout behavior.

Leave mandatory fields blank and submit the form.