5 Efficient Test Case Design Techniques to Increase Coverage

TechRounder PDF Edition

Live article: https://www.techrounder.com/development/efficient-test-case-design-techniques-to-increase-coverage/

By Vipin PG | Published September 7, 2023 | Updated January 4, 2026 | Format: Article | 5 min read

In brief

Software testing is a cornerstone of the software development lifecycle. It ensures that the end product is reliable and meets its intended purpose.

Software testing is a cornerstone of the software development lifecycle. It ensures that the end product is reliable and meets its intended purpose. The process of test case design plays a crucial role in effective testing. In this article, together with Zebrunner, we delve into the significance of test case design and explore essential principles and techniques that can efficiently boost test coverage.

Role of test case design in testing

Test case design, in simple words, is the process of creating a detailed plan or set of instructions that outlines how a specific aspect of a software application should be tested. It involves defining what needs to be tested, what steps to follow, what data to use, and what the expected outcomes should be.

Test case design ensures that testing is systematic, organized, and thorough, helping testers identify and report any issues or defects in the software accurately. It’s like creating a recipe for testing, specifying exactly what needs to be done to check if the software works as intended.

Essential test case design principles

Below, you can find 4 main test case design principles. Following common rules when developing test cases speeds up your quality assurance process. This ensures that your testing team is in a shared knowledge field, where every team member contributes, resulting in faster releases. Also, don’t forget to use good test case management software to make your work easier and better organized.

1. Clear identification of test objectives and goals. The foundation of effective test case design lies in understanding the objectives and goals of the testing effort. This involves determining what needs to be tested, the scope of testing, and the intended outcomes. A clear understanding of objectives guides the creation of focused and purpose-driven test cases.

2. Proper understanding of requirements and user scenarios. A deep comprehension of software requirements and user scenarios is crucial for crafting relevant test cases. Each requirement and scenario needs to be translated into specific test cases that address the functional and non-functional aspects of the application.

3. Isolation of test cases for independent validation. Isolating test cases ensures that each test case independently verifies a specific functionality or scenario. This isolation prevents dependency issues and allows for easier identification of defects when they arise.

4. Incorporation of boundary and edge cases for robust testing. Boundary and edge cases often expose vulnerabilities that standard test cases might miss. You can ensure the software’s robustness by designing test cases that push the limits of input values, data ranges, and system states.

Techniques to enhance test case efficiency

1. Equivalence partitioning

This is a technique that streamlines the design of test cases by grouping similar inputs into equivalence classes. This method efficiently covers a range of scenarios within each class, making the testing process more effective.

Example Imagine testing a login form that requires a username and a password. Instead of testing every possible username and password combination, equivalence partitioning allows you to group inputs into categories. For instance:

- Equivalence class 1: Valid usernames (e.g., "user123", "john.doe")
- Equivalence class 2: Invalid usernames (e.g., "admin", "guest")
- Equivalence class 3: Valid passwords (e.g., "P@ssw0rd", "SecurePwd123")
- Equivalence class 4: Invalid passwords (e.g., "password", "123456")

By designing test cases to cover each equivalence class, you efficiently test a representative subset of values within that category. This technique ensures that various inputs are tested without exhaustive testing of every possible combination.

2. Boundary value analysis

This method focuses on testing values at the edges of input ranges. This technique aims to identify potential issues that might emerge due to boundary conditions. Doing so ensures that the software effectively handles extreme inputs and maintains its integrity.

Example Consider testing a web application that processes age inputs for user registration. The acceptable age range might be between 18 and 65 years.

- The lower boundary: Test with an age value of 18. This checks if the software correctly handles the minimum valid age.
- The upper boundary: Test with an age value of 65. This verifies if the software properly handles the maximum valid age.
- Values just inside the boundaries: Test with ages 19 and 64 to ensure the software behaves as expected near the edges.
- Values just outside the boundaries: Test with ages 17 and 66 to uncover any issues that might occur at the edges of the acceptable range.

As we see, boundary value analysis helps pinpoint potential bugs or vulnerabilities related to input limits, contributing to a more robust application.

3. Decision table testing

Decision table testing offers an effective approach when dealing with complicated business logic that demands thorough testing. This technique simplifies the testing process by organizing a matrix of input conditions and their expected outcomes. It ensures a comprehensive examination of diverse scenarios.

Example Suppose you’re testing a flight booking system with complex rules for determining ticket prices. These rules might involve travel class, booking date, and frequent flyer status. Decision Table Testing allows you to define combinations of these factors and their respective outcomes. For instance:

- Travel Class: Economy, Business, First Class
- Booking Date: Peak Season, Off-Peak Season

- Frequent Flyer Status: None, Silver, Gold

By systematically testing these various scenarios as defined in the decision table, you can ensure that the pricing logic responds correctly to different combinations of inputs. This method is particularly beneficial when the interactions between conditions become intricate, enhancing your ability to identify defects accurately.

4. State transition testing

State transition testing is a test case design method that holds particular value for systems characterized by distinct states or modes. This technique validates the transitions between these states, ensuring the correct behavior at every system operation step.

Example Consider a simple traffic light system with three states: red, yellow, and green. The transitions between these states follow a defined sequence. State transition testing for this scenario would involve validating the correct behavior during each state transition:

- From red to green: The light should smoothly transition from red to green after a specific time interval.
- From green to yellow: There should be a brief yellow interval before switching from green to red.
- From yellow to red: The light should transition seamlessly from yellow to red.

By meticulously testing these state transitions, you ensure the traffic light system functions as expected, adhering to the predefined sequence and timing. State transition testing is beneficial when the system’s behavior varies significantly based on its operational state.

This technique enhances your ability to identify defects related to state transitions, leading to a more reliable and well-behaved software system.

5. Pairwise testing

This is a test case design method that optimizes test coverage by generating test cases covering various combinations of input parameters. This technique minimizes the required tests while ensuring comprehensive coverage of possible scenarios.

Example Imagine you’re testing a form with multiple input fields: Name, Email, Phone Number, and Address. Instead of exhaustively testing every combination, pairwise testing allows you to focus on relevant interactions. For instance:

- Test case 1: Name + Email
- Test case 2: Name + Phone Number
- Test case 3: Name + Address
- Test case 4: Email + Phone Number
- Test case 5: Email + Address
- Test case 6: Phone Number + Address

By generating test cases in pairs, you effectively cover the interactions between two parameters simultaneously. This approach significantly reduces the required test cases while addressing potential issues arising from parameter interactions.

References

1. zebrunner.com - test-case-management - https://zebrunner.com/test-case-management?utm_source=referral&utm_medium=techrounder&utm_campaign=seo