SEMESTER PROJECT

Subject: Software Quality Engineering

Submitted to: Dr. Sumaira Nazir

Section: BSSE-6th (Evening)

Dated: 14th December, 2023

Submitted by:

Fayyaz Ahmad (SP21116)

Uzair Sajid (SP21133)

Fatima Wahid (SP21135)

Department of Software Engineering, National University of Modern Languages, Islamabad

Fall, 2023

Contents

BLACK BOX TESTING

What is Black Box Testing?

Black-box testing is a software testing method that focuses on the functionality of a software application without requiring knowledge of the internal code or logic.

Importance of Black-box Testing:

Ensures External Compliance:

- Validates that the software meets specified requirements.

User Perspective:

- Mimics user interactions, ensuring a user-friendly experience.

Objective Assessment:

- Provides an unbiased evaluation of the system's functionality.

Identifying Defects:

- Aids in discovering errors and defects in the software.

White Box vs. Black Box Testing

Black Box Testing	White Box Testing
• It involves testing from an external or end-user perspective.	• It is based on the inner workings of an application and revolves around internal testing.
• The Black Box name symbolizes not being able to see the inner working of the software so that only the end-user experience can be tested.	• White Box name symbolizes the ability to see through the software's outer shell into its inner working.

What do you verify in Black Box Testing?

Black box testing focuses on verifying the functionality of a system without requiring internal knowledge of its code or structure. In this method:

Input/Output Validation:

Testers assess whether the inputs produce the expected outputs. This involves providing various inputs and analyzing the corresponding outputs to ensure they align with expected behaviors.

Functional Testing:

This involves testing the functionalities or features of the system. Testers examine if the system performs as intended according to the specifications.

Boundary Testing:

This verifies the behavior of the system at the boundaries of allowed inputs. Testers check how the system handles maximum and minimum input values or conditions.

Error Handling:

Evaluating how the system handles unexpected or erroneous inputs. Testers intentionally input incorrect data to verify if the system detects and responds appropriately.

Usability Testing:

Assessing the user interface and overall user experience without delving into the internal workings. This involves testing navigation, ease of use, and intuitiveness.

Integration Testing:

Verifying the interactions between various system components or modules to ensure they work together as expected.

Why Black Box Testing:

- A benefit of a specification-based approach is that the tests are looking at what the software should do, rather than what it does do.
- Expected outcomes can be generated if they are stored in the specification, assuming that the stored specification is actually correct.
- The designer and the tester are independent of each other.
- The testing is done from the point of view of the user.
- Test cases can be designed after requirements are clear

When Not to Test Software Using Black Box:

- Test cases are challenging to design without having clear functional specifications.
- Inefficient testing because tester only has limited knowledge about an application.
- A good test case is one that has a reasonable probability of finding an error.
- Keeping in view the fact that an exhaustive-input testing of a program is impossible
- Limited to trying a small subset of all possible inputs Of course, then, we want to select the right subset.
- The subset with the highest probability of finding the most errors

Subset in Black Box testing:

One way of locating the subset is to realize that a well-selected test case also should have two other properties:

1. It reduces, by more than a count of one, the number of other test cases that must be developed to achieve some predefined goal of "reasonable" testing.

2. It covers a large set of other possible test cases

Black Box Testing Techniques

Following techniques are used in Black Box Testing:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing

Equivalence Partitioning

Equivalence Partitioning is a software testing technique that involves dividing the input data of a software application into partitions or groups, where each group is expected to exhibit similar behavior. The goal of Equivalence Partitioning is to reduce the number of test cases while still ensuring comprehensive test coverage.

Example: User Age Input in a Registration Form

Scenario:

Consider a registration form for an online service that requires users to input their age. The acceptable age range is 18 to 99 years.

Equivalence Classes:

Valid age range: 18 to 99 (inclusive)

Invalid age range: < 18 and > 99

Test Cases:

- Input an age within the valid range (e.g., 25) - Expected outcome: Registration is successful.
- Input the minimum valid age (18) - Expected outcome: Registration is successful.
- Input the maximum valid age (99) - Expected outcome: Registration is successful.
- Input an age below the valid range (16) - Expected outcome: Registration fails with an error message.
- Input an age above the valid range (105) - Expected outcome: Registration fails with an error message.

Example: Credit Card Transaction Amount Limit

Scenario:

A banking application allows users to perform credit card transactions. There is a limit on the transaction amount for different types of users: regular users and premium users.

Equivalence Classes:

Regular user transaction limit: $1 to $500

- Premium user transaction limit: $1 to $2000

Test Cases:

- Regular user processes a transaction within the limit ($300) - Expected outcome: Transaction is successful.
- Premium user processes a transaction within the limit ($1500) - Expected outcome: Transaction is successful.
- Regular user attempts to process a transaction exceeding the limit ($700) - Expected outcome: Transaction fails with an error message.
- Premium user attempts to process a transaction exceeding the limit ($2500) - Expected outcome: Transaction fails with an error message.
- Regular user attempts to process a transaction with the minimum allowed amount ($1) - Expected outcome: Transaction is successful.
- Premium user attempts to process a transaction with the minimum allowed amount ($1) - Expected outcome: Transaction is successful.

Boundary Value Analysis

Boundary Value Analysis (BVA) is a software testing technique that focuses on testing values at the boundaries of input domains. The idea behind BVA is that errors are often concentrated at the edges or boundaries of input ranges rather than in the middle of the input domain. By testing values at the boundaries, testers aim to identify potential issues related to boundary conditions.

Example: Validating a Range of Values for a Text Field:

Consider a scenario where a system has a text field that accepts input with a minimum length of 5 characters and a maximum length of 15 characters.

Boundary Values:

- Minimum Length: 5 characters
- Maximum Length: 15 characters

Boundary Value Analysis Testing:

- Test Case 1: Enter a value with 4 characters (below the minimum). The system should reject the input.
- Test Case 2: Enter a value with exactly 5 characters (minimum). The system should accept the input.
- Test Case 3: Enter a value with exactly 15 characters (maximum). The system should accept the input.
- Test Case 4: Enter a value with 16 characters (above the maximum). The system should reject the input.

Scenario: Validating Numeric Input for a Range:

Imagine a scenario where a system accepts numeric input for a field that must be between 10 and 100 (inclusive).

Boundary Values:

- Minimum Value: 10
- Maximum Value: 100

Boundary Value Analysis Testing:

- Test Case 1: Enter a value of 9 (below the minimum). The system should reject the input.
- Test Case 2: Enter a value of 10 (minimum). The system should accept the input.
- Test Case 3: Enter a value of 50 (within the valid range). The system should accept the input.
- Test Case 4: Enter a value of 100 (maximum). The system should accept the input.
- Test Case 5: Enter a value of 101 (above the maximum). The system should reject the input.

Decision Table Testing

Decision table testing is a black-box testing technique used to test the various combinations of input conditions and their corresponding outputs. It is particularly useful when there are multiple input conditions and the system's behavior depends on the combinations of these conditions.

A decision table consists of rows and columns, where each column represents a different input condition or combination of conditions, and each row represents a unique test case with corresponding expected outputs.

Example 1: Online Shopping Cart Checkout

Assume we have an online shopping cart system, and we want to test the checkout process based on various conditions such as payment method, shipping method, and order total.

Condition 1	Condition 2	Condition 3	Output
PaymentMethod	ShippingMethod	OrderTotal	CheckoutResult
Credit Card	Express Shipping	$100 or more	Successful
Credit Card	Standard Shipping	Less than $100	Successful
PayPal	Express Shipping	Less than $100	Successful
PayPal	Standard Shipping	$100 or more	Successful
Other	Any	Any	Unsuccessful

In this example, we have three input conditions: PaymentMethod, ShippingMethod, and OrderTotal. The Output column represents the expected result of the checkout process.

Example 2: Student Enrollment System

Consider a student enrollment system where students can enroll in courses based on their eligibility and available seats.

Condition 1	Condition 2	Output
StudentStatus	AvailableSeats	EnrollmentStatus
New Student	Seats available	Enroll
New Student	No seats available	Waitlist
Returning Student	Seats available	Enroll
Returning Student	No seats available	Not Enroll

In this example, the input conditions are StudentStatus and AvailableSeats. The Output column represents the expected EnrollmentStatus for a student under different conditions.

State Transition Testing

State Transition Testing is a type of software testing that focuses on the transitions between different states of a system. In many applications, the behavior of the system changes based on its current state and the input it receives.

ATM Withdrawal System:

States:

- Initial State: ATM ready/idle.
- State 1: User inserts the card.
- State 2: User enters the PIN correctly.
- State 3: User selects an account type (e.g., checking or savings).
- State 4: User enters the withdrawal amount.
- State 5: Sufficient funds are available.
- State 6: Dispensing cash.
- State 7: Transaction complete.

Transitions:

- Transition 1: From Initial State to State 1 (Card insertion).
- Transition 2: From State 1 to State 2 (Correct PIN entry).
- Transition 3: From State 2 to State 3 (Account type selection).
- Transition 4: From State 3 to State 4 (Entering withdrawal amount).
- Transition 5: From State 4 to State 5 (Sufficient funds available).
- Transition 6: From State 5 to State 6 (Dispensing cash).
- Transition 7: From State 6 to State 7 (Transaction complete).

Test Cases:

- Verify that the system transitions from the Initial State to State 7 for a valid withdrawal sequence.
- Test if the system rejects the transaction when an incorrect PIN is entered (transition from State 2 to an error state).
- Confirm that the system prompts the user to select an account type (transition from State 2 to State 3).

E-commerce Shopping Cart:

States:

- Initial State: Shopping cart is empty.
- State 1: Product added to the cart.
- State 2: User proceeds to checkout.
- State 3: User provides shipping information.
- State 4: User makes a payment.
- State 5: Order confirmation.

Transitions:

- Transition 1: From Initial State to State 1 (Adding a product to the cart).
- Transition 2: From State 1 to State 2 (Proceeding to checkout).
- Transition 3: From State 2 to State 3 (Providing shipping information).
- Transition 4: From State 3 to State 4 (Making a payment).
- Transition 5: From State 4 to State 5 (Order confirmation).

Test Cases:

- Validate that the system allows adding products to the cart and transitions from the Initial State to State 1.
- Test if the system prompts the user to provide shipping information when attempting to proceed to checkout (transition from State 1 to State 3).
- Ensure that the system displays an order confirmation after successful payment (transition from State 4 to State 5).

Advantages of Black Box Testing

1. Objective Testing
2. Encapsulation
3. External Quality Assessment
4. Effective Error Identification

Objective Testing

Black box testing allows for an objective assessment of software functionality by focusing solely on inputs and expected outputs without knowledge of internal code structures. Testers evaluate the system based on specified requirements, ensuring that the software functions as intended from an end-user perspective. This objectivity enhances the reliability of the testing process and facilitates unbiased evaluation of the software's performance.

Encapsulation

Black box testing promotes encapsulation by treating the software as a closed box, emphasizing the external behavior while disregarding the internal implementation details. This encapsulation allows testers to concentrate on the functional aspects of the software without being influenced by its internal complexities. It also supports modularity and abstraction, making it easier to identify and address issues without delving into the intricacies of the code.

External Quality Assessment

Black box testing provides an external perspective on software quality, evaluating the system based on user requirements and expected outcomes. This approach assesses the software's functionality, reliability, and usability without requiring knowledge of its internal design. External quality assessment is crucial for ensuring that the software meets user expectations and performs effectively in real-world scenarios.

Effective Error Identification

Black box testing is effective in identifying errors or defects in the early stages of the software development life cycle. By focusing on inputs and outputs, testers can uncover discrepancies between expected and actual results, helping to pinpoint issues such as incorrect calculations, improper data handling, or functionality deviations. This early error identification contributes to overall software quality and reduces the cost of fixing defects later in the development process.

Disadvantages of White Box Testing

1. Incomplete Testing
2. Limited Code Coverage
3. Dependency on Specifications
4. Difficulty in Error Localization

Incomplete Testing

Black box testing may result in incomplete test coverage as it primarily focuses on inputs and outputs, potentially overlooking certain code paths or internal logic. Without insight into the internal workings of the software, testers may miss critical scenarios, leading to gaps in test coverage and a higher likelihood of undetected defects.

Limited Code Coverage

Black box testing often provides limited coverage of code paths, especially when compared to white box testing. Testers have no visibility into the internal code structure, making it challenging to verify every possible execution path. Consequently, certain parts of the code may remain untested, leaving potential defects undiscovered.

Dependency on Specifications

Black box testing heavily relies on the accuracy and completeness of the specifications and requirements provided. If the specifications are unclear, ambiguous, or incomplete, it can lead to ineffective testing. Testers may misinterpret requirements, resulting in inadequate test cases and potentially missing critical aspects of the software's intended functionality.

Difficulty in Error Localization

When defects are identified during black box testing, pinpointing the exact location and cause of the error within the code can be challenging. Testers, lacking visibility into the internal structure, may struggle to localize and diagnose the root cause of issues. This difficulty in error localization can lead to longer debugging cycles and delays in resolving identified defects.

Conclusion

Black box testing serves as a crucial validation method, focusing on the functionality and behavior of a system without requiring knowledge of its internal structure. By testing inputs and analyzing outputs, assessing functionalities, examining error handling, and ensuring usability, it validates whether the system operates in accordance with its specified requirements. This approach is vital for ensuring that the end-user experience aligns with expectations and that the system functions correctly in real-world scenarios, contributing significantly to the overall quality and reliability of the software or application.