Shravan

CT14963

Black Box Testing

In black box testing, the testing team analyses the workings of an application without first having an extensive understanding of its internal structure and design. During testing, the input value is simply compared with the output value. Due to its nature, black box testing is sometimes called specification-based testing, closed box testing, or opaque box testing.

Black Box Testing Methods:

1. Equivalence Partitioning:

Also known as equivalence class partitioning (ECP), this black box testing technique calls for the input values for the application or system to be classified based on outcome similarity. This enables testing teams to use only one value from within the class or group for analyzing the outcome rather than having to review all the relevant input values from the group. This technique maintains test coverage, and the amount of rework required and time spent are minimized.

For instance, let’s assume a text field that only accepts numerical inputs between 1950 and 2000 to be tested. Using equivalence partitioning, we can create three sets of groups for testing this field: any number less than 1950, any number between 1950 and 2000, and any number greater than 2000.

The valid class will be any one number between 1950 and 2000, while the other two groups will be invalid classes.

In this example, the testing team has reduced the cases to only three, allowing all possible scenarios to be tested within a few moments.

2. Boundary Value Analysis:

In this technique, the tester focuses on analyzing both valid and invalid values at the boundaries–or the limits at which the system behavior changes. This technique is popular as many applications exhibit bugs when processing boundary values.

Continuing the above example, let’s say a field that accepts values between 1950 and 2000 is being tested. The tester, using boundary value analysis, will enter the following values during testing: 1949 (1950-1), 1950, 1951 (1950+1), 1999 (2000-1), 2000, and 2001 (2000+1).

3. Decision Table Testing:

With this method, the tester detects two outputs for two conditions that are defined by a logical connection, such as:

If

{

(Condition = True)

then output1 ;

}

else output2; /\*(condition = False)\*/

Based on the above example, the tester will create a decision table with all the probable outcomes.

4. State transition testing:

In this technique, the tester analyzes the various states of the application, which change as per events or conditions that the application is subjected to. Each event triggers a state that is then treated as a scenario to be tested.

This technique is handy for simple scenarios, while more complex scenarios will create complicated transition diagrams and make the technique less effective.

5. Experience-based testing:

Guessing the errors that might arise in the application is a key example of this technique. Here, the testing team leverages its experience around the application’s behavior and its functionalities to find the areas prone to errors.

Examples of guessing based on experience include searching for common errors, such as dividing by zero, handling null values in text fields, and accepting blank inputs where they should not be allowed.

6. Graph-based testing:

All applications are created using objects. This method identifies the objects, and an ‘object graph’ is created. This graph is used to identify all object relationships, write test cases, and test for error discovery.

7. Comparison testing:

Finally, in this technique, independent versions of one application are compared to each other during the testing process.

White Box Testing

The software application’s internal coding, design, and structure are examined in white box testing to verify data flow from input to output. White box testing is leveraged to improve design, usability, and application security. The other names for this methodology include code-based testing, glass box testing, open box testing, clear box testing, and transparent box testing.

Unlike black box testing, which focuses on ensuring a smooth user experience, white box testing is intensive. Combined with other bug-quashing techniques, it is a robust quality assurance tool. This methodology is designed to conduct in-depth simulations of all the scenarios the application might encounter at the code level.

White Box Testing Methods:

1. Statement coverage:

All statements are at least once executed at the source code level in this white box testing approach. The number of statements executed is recorded.

This technique allows the testing team to analyze the source code and set expectations around what it can and cannot do. It is also reliable for checking code quality and verifying path flow. However, one cannot use it to test false conditions.

This method tests the internal coding and infrastructure of the software. It can be carried out by programmers and is suitable for Drupal specialists.

2. Decision coverage:

In this technique, decision coverage is given to Boolean values, and Boolean expressions’ true and false results are reported.

Control flow statements that create a possibility of two or more outcomes, such as ‘do while’, ‘if’, and ‘case’, are considered as ‘decision points’ in this technique as they can lead to two outcomes: either true or false.

This technique covers all the possible results of every Boolean condition within the code through control flow charts and graphs. However, it can sometimes be difficult to achieve complete coverage due to the existence of complicated expressions.

3. Branch coverage:

This technique covers all branches of the control flow graph. Each decision point condition’s possible true and false outcomes are covered at least once. This method guarantees that the several branches of each decision point are executed successfully.

Numerous methods are available for calculating branch coverage, the most common of which is pathfinding. This method uses the number of executed branch paths to calculate branch coverage. Developers can substitute decision coverage with branch coverage.

4. Condition coverage:

In this technique, the testing team analyzes all the conditional expressions in an application for every possible outcome. All the conditions are tested independently, and all the possible results of every condition are tested at least once. Predicate coverage is another name for this technique.

5. Multiple condition coverage:

In this technique, all the possible permutations of condition outcomes in every decision, as well as all entry points, are tested at least once. Full multiple condition coverage generally requires a high number of test cases.

6. Path coverage:

All program paths are tested in this comprehensive technique. It is leveraged to ensure that every application path is utilized at least once. This technique is generally useful for testing complex programs.

7. Control flow testing:

This technique determines the sequence in which an application’s statements or instructions are executed via a control structure. The test cases of the analyzed application are developed using the control structure. Control flow testing is generally used during unit testing.

Testers, using this technique, select a specific part of the application for setting the testing path. The control flow graph is created from the edge, node, junction node, and decision node to outline all the possible execution paths.

8. Data flow testing:

Finally, white box testers use this technique to analyze the control flow of programs to explore the variable sequence based on the event sequence. During testing, the focus lies on two points: where values are assigned to the variables and where these values are used.

In this technique, the control flow graph is used to detect logical inconsistencies that disrupt data flow. Reasons for anomaly detection include variables being used without initialization and initialized variables not being used.