White Box Testing with Object Oriented programming

Mrs.Randive R.B¹, Mrs.Bansode S.Y²
¹²TPCT’S College Of Engineering, Osmanabad

Abstract: Software testing is one of the best means to affirm the quality of software and deliver an error-free application. Software testing is used to evaluate the correctness, completeness and quality of developed computer software. Software has matured into a separate discipline giving way to several different testing techniques that have been introduced, analyzed and studied in this area. Black box testing and white box testing are two such testing approaches that are quite commonly used by software testers. White-box testing method is used for logical and analytic test in unit test level.

Keywords: Statement Tests Decision Tests Branch Condition Tests Decision/Condition Tests Data Flow Tests Multiple Condition Tests

I. INTRODUCTION

In white box testing methodology, the tester has the knowledge of the internals of a system and knows how the system is implemented. The tester uses this knowledge to develop test cases that will examine the control flow, information flow, data flow, exception and error handling as well as coding practices of the system.

A. How to write Test Cases for White Box Testing?

- The tester analyzes and understands the structure of the system by examining its code.
- The tester understands the weak spots within the code that is most prone to defects.
- The tester develops test cases to cover individual data/information/control flows and branches within the code.
- The tester also develops test cases to test proper working of all the functionalities and error handling of the system.

B. Techniques of White Box Testing

When it comes to white box testing, the knowledge that the tester possesses about the system is the driving factor, which helps the tester to devise test cases aimed at discovering defects with the internal working of the system.

- Statement Tests: All the statements within the code must have a test case associated with it such that each statement must be executed at least once during the testing cycle.
- Decision Tests: All the decision directions must be executed at least once during the testing life cycle.
• **Branch Condition Tests**: All the conditions in a specific decision must be tested for proper working at least once.

• **Decision/Condition Tests**: All the combination of the possible conditions within a specific decision for all the decisions is to be tested.

• **Data Flow Tests**: This will ensure that all the variables and data that are used within the system are tested by passing the specific variables through each possible calculation.

• **Multiple Condition Tests**: This will ensure that each point of entry with in the code is tested at least once during the testing life cycle.

**II. OBJECTIVE**

The objective of this paper is: design and development of test cases using white box testing methods for object-oriented software. The aim of this paper is to study various established as well as emerging techniques, with special focus on those for object-oriented software using Statement Tests, Decision Tests, Branch Condition Tests, Decision/Condition Tests, Data Flow Tests, and Multiple Condition Tests.

**III. EXAMPLES FOR OBJECT ORIENTED SYSTEM USING TEST CASES**

Examples: - The Pseudocode is like
1. INPUT C & D
2. E=C+D
3. IF E>50
4. PRINT”IT IS DONE”

For **Statement Coverage** – we would need only one test case to check all the lines of code.

That means:
If I consider Test Case_01 to be (A=30 and B=40), then all the lines of code will be executed

Now the question arises:
Is that sufficient?

What if I consider my Test case as A=22 and B=29?
Because Statement coverage will only cover the true side, for the pseudo code, only one test case would NOT be sufficient to test it. As a tester, we have to consider the negative cases as well.

Hence for maximum coverage, we need to consider “Branch Coverage”, which will evaluate the “FALSE” conditions.

In the real world, you may add appropriate statements when the condition fails.

So now the pseudocode becomes:
1. INPUT A & B
2. C=A+B
3. IF C>50
4. PRINT”IT IS DONE”
5. ELSE
6. PRINT “IT IS PENDING”

Since Statement coverage is not sufficient to test the entire pseudo code, we would require Branch coverage to ensure maximum coverage.

So for Branch coverage, we would require two test cases to complete testing of this pseudo code.

**Test Case_01**: A=37, B=10
**Test Case_02**: A=25, B=30

With this, we can see that each and every line of code is executed at least once.
Here are the conclusions so far:

- Branch Coverage ensures more coverage than Statement coverage.
- Branch coverage is more powerful than Statement coverage.
- 100% Branch coverage itself means 100% statement coverage.
- 100% statement coverage does not guarantee 100% branch coverage.

Now let’s move on to the **Path Coverage**:

As said earlier, Path coverage is used to test the complex code snippets, which basically involves loop statements or combination of loops and decision statements.

**Consider this pseudocode:**

1. INPUT A & B
2. C = A + B
3. IF C>50
4. PRINT “IT’S DONE”
5. END IF
6. IF A>25
7. PRINT “IT’S PENDING”
8. END IF

Now to ensure maximum coverage, we would require 4 test cases. How?

Simply – there are 2 decision statements, so for each decision statement, we would need to branches to test. One for true and other for the false condition. So for 2 decision statements, we would require 2 test cases to test the true side and 2 test cases to test the false side, which makes a total of 4 test cases.

**To simplify these let’s consider below flowchart of the pseudo code we have:**

![Flowchart](image-url)
So, in order to have the full coverage, we would need following test cases:

**TestCase_01:** A=50, B=20  
**TestCase_02:** A=25, B=30  
**TestCase_03:** A=40, B=65  
**TestCase_04:** A=30, B=30

So the path covered will be:

Red Line = TestCase_01 = (A=50, B=20)  
Blue Line = TestCase_02 = (A=25, B=30)  
Orange Line = TestCase_03 = (A=40, B=65)  
Green Line = TestCase_04 = (A=30, B=30)

**IV. CONCLUSION**

Note that the statement, branch or path coverage does not identify any bug or defect that needs to be fixed. It only identifies those lines of code which are either never executed or remains untouched. Based on this further testing can be focused on.

Relying only on black box testing is not sufficient for maximum test coverage. We need to have combination of both black box and white box testing techniques to cover maximum defects.

If done properly, White box testing will certainly contribute to the software quality. It’s also good for testers to participate in this testing as it can provide the most “unbiased” opinion about the code.

**V. ADVANTAGES**

White-box testing is one of the two biggest testing methodologies used today. It has several major advantages:

1. Side effects of having the knowledge of the source code is beneficial to thorough testing.
2. Optimization of code becomes easy as inconspicuous bottlenecks are exposed.
3. Gives the programmer introspection because developers carefully describe any new implementation.
4. Provides traceability of tests from the source, thereby allowing future changes to the source to be easily captured in the newly added or modified tests.
5. Easy to automate.
6. Provides clear, engineering-based rules for when to stop testing.

VI. DISADVANTAGES

Although white-box testing has great advantages, it is not perfect and contains some disadvantages:
1. White-box testing brings complexity to testing because the tester must have knowledge of the program, including being a programmer. White-box testing requires a programmer with a high level of knowledge due to the complexity of the level of testing that needs to be done.
2. On some occasions, it is not realistic to be able to test every single existing condition of the application and some conditions will be untested.

The tests focus on the software as it exists, and missing functionality may not be discovered.

VII. FUTURE WORK:

Future work would include extending the tool to incorporate more functionality. Both testing and maintenance components can be added. Some additions can be:
1. Incorporating a fully functional Method Basis path generator module.
2. Providing both test case Generation as well as Execution. The user would be able to provide test data; and the test cases generated would be executed using the test data as input.
3. Reporting code coverage achieved after test set has been executed. Various test adequacy criteria like statement coverage, branch coverage, and path coverage can be reported upon.
4. Metrics: certain program metrics like lines of code (LOC), functions points, interfaces, etc., can be reported upon.

REFERENCES

III. Mary Jean Harrold , Gregg Rothermel “Performing data flow testing on classes ” December 1994 ACM SIGSOFT software Engineering Notes, PROCEEDINGS of the 2nd ACM SIGSOFT symposium on foundations of software engineering, volume 19 Issue 5