White-Box and BlackBox

By

Mansoor Alruithe

Alru0011 - 2090688

June 2014

Table of Contents

Comparison between White-box and Black-box Test

Introduction

Software testing is aimed at assessing the differences that may exist between required conditions and existing conditions(Myers et al. 2011). The information gained from the assessment can be used to gauge the quality of the software under test. The assessment can also provide an overview of the risks of implementing the software. To achieve the testing objectives, both black-box testing and white-box testing can be used. This paper contains a deep comparison between black box and white box testing; it explains the use and meaning of each type of testing, lists the advantages and disadvantages of each type, and also provides some examples for each type of testing to highlight the differences between the two types of testing.

Black box testing

Overview

Black-box testing is a functional and behavioral testing technique which is used in determination of whether or not a program performs what it is supposed to, functionally(Beizer 1995). Black-box testing concentrates on the results of the software by providing input and examining the output without knowing how this software has been implemented (Stepp, nd).. The idea of the software product being tested without the knowledge of implementation – black-box – gives it the name black-box testing(Beizer 1995). Although the tester might have had a look at the code before testing, he/she has no access to the source code while executing the black-box test; tester only deals with the application user interface and executes a pre-designed test cases(Beizer 1995). Black-box testing ensures that application functionality is working as expected, it is also known as functional testing or data driven testing. Figure 1 demonstrates the general idea of black-box testing.

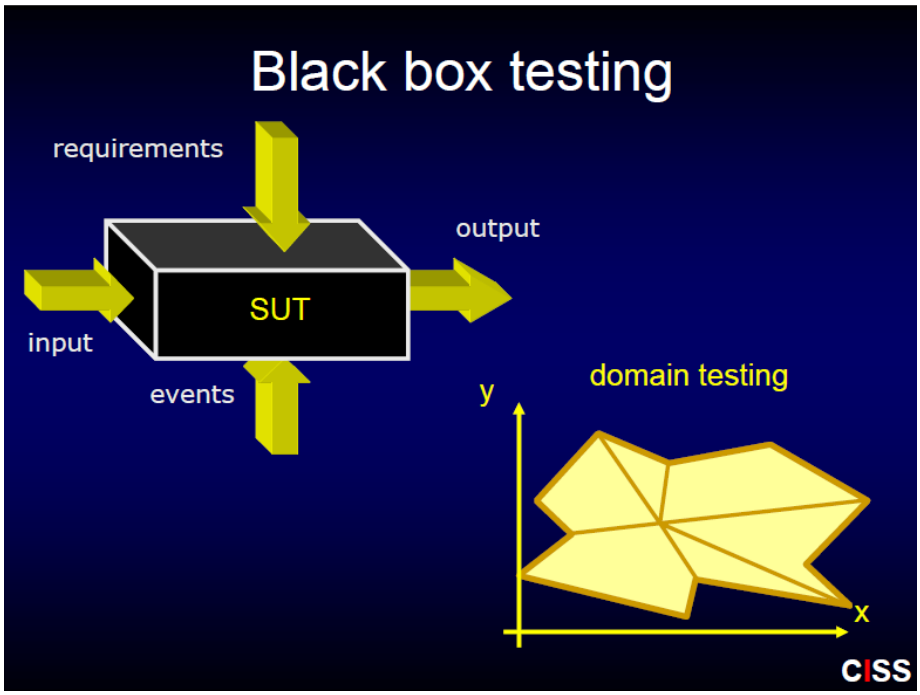

Figure 1: Black-box Testing

Figure one depicts that the three major approaches in black-box testing include the analysis of input and output to a software application domain of the program; the analysis of observable black-box behavior; and heuristics analysis(Beizer 1995).

Advantages of black box testing

To start with, testing a huge software application with enormous amount of code can be tedious and time consuming. Although it is important, the efficiency with which black-box testing bring to the tester allows him or her to identify efficiently the main areas where debugging is required. In addition, this can be done is a very short time(Schroeder and Korel 2000). Further, the tester does not require the help of code to do the tests. However, the tester must access a functional user interface of the software application on which the test has to be done(Schroeder & Korel 2000). Although the tester might be a developer, he/she must behave like a user – for this reason, black-box testing separates the developers' perspective from the user's perspective(Schroeder & Korel 2000). In addition, non-technical testers can be used in testing as long as they are given an outline on how the software application is supposed to function(Schroeder & Korel 2000). Despite this, the tester is not required to have a full functional

knowledge of the software application. It is imperative to note that black-box testing helps the developers to identify vagueness and contradictions in functional specifications of software applications(Schroeder & Korel 2000). Test cases can be designed as soon as the functional implementation of the software system has been completed.

These advantages are summarized below:

- It is efficient for large code.
- It is consumes less time and effort comparing to white box testing.
- It doesn't require code access
- It clearly separates developer perspective from the user perspective
- Low skilled testers can test the system without being aware of how it is implemented
- The testers do not require to have a full functional knowledge of the system
- Black-box testing helps the developers identify contradictions in functional specifications

Disadvantages of black box testing

Beside the advantages that black-box testing brings to software testing, it is inevitable for the testing method to have its flows. First, "tester is not required to have a full functional requirement" brings a challenge to designing test cases(Schroeder & Korel 2000). This is because the user might test some functions and leave out other functions. It is also possible that the user cannot identify all the requirements(Schroeder & Korel 2000). Software specifications are important in building any test cases. This type of testing does not guarantee the knowledge of specifications which offers a big challenge and can lead to building vague test cases(Schroeder & Korel 2000). This is complimented by the difficulty to point out all inputs and their required outputs; pointing out these important parameters can make the process slow and difficult especially for a non-technical user(Beizer 1995;Schroeder & Korel 2000). Consequently, there is high probability of leaving out testing paths in the process. It is important to

point out that this testing method is not suitable in some cases – for instance, testing of algorithms(Beizer 1995;Schroeder & Korel 2000).

- It has a limited coverage, as it only execute a selected number of test cases.
- Testers have limited knowledge about the system being under test.
- It blindly covers the system features, since the tester has no access to source code.
- Test cases are difficult to be designed.
- It is not suitable in some cases like testing algorithms.
- The tester might repeat tests done by the developer
- Some test paths might be skipped during the process

Examples of black box testing for my version of Game 2048

Unit Test 1 – Application randomly populates tow tiles at game start

Prerequisites and conditions

1- Game is started.

Scenario

1- User starts the game.
2- Board is displayed to user.

Scope of execution

This test case insures that application randomly populates tow tiles at game start.

Expected result

Tow tiles to be randomly populated by 2 or 4 values “randomly”.

Success criteria

Game board should be displayed showing tow random tiles have random values 2 or 4.

Actual result

Game board is displayed showing tow random tiles with random values 2 or 4.

Status

Passed

Unit test 2 – Application randomly populates one tile after each user move

Prerequisites and conditions

1- Game is started.

Scenario

1- User starts the game.
2- Board is displayed to user.
3- User makes move to left, right, top, or down.

Scope of execution

This test case insures that application randomly populates one tile after user moves.

Expected result

One free tile to be randomly populated by 2 or 4 values “randomly”.

Success criteria

Game board should be displayed showing new random tile with random value 2 or 4.

Actual result
Game board is displayed showing new random tile with random value 2 or 4.

Status
Passed

Unit test 3 – Application exits when board is full

Prerequisites and conditions
1- Game is started.

Scenario
1- User starts the game.
2- Board is displayed to user.
3- User makes move to left, right, top, or down.
4- New tile is populated with random value 2 or 4
5- Repeat steps from 2 to 4 until all tiles are populated, and board is full.

Scope of execution
This test case insures that application exits when board is full.

Expected result
Application should exit and display “Game over” message to user.

Success criteria
Application exits and displays “Game over” message to user.

Actual result
Application exited and displayed “Game over” message to user.

Status
Passed

White-box testing

Overview

White-box testing is used by software developers to verify that a specified piece of code works as per required specifications(Mathur 1991). It is a detailed investigation process of the code structure and internal logic of the system. Before a developer can test the code, knowledge of the implementation code is required (Stepp, nd). The code can then be divided into sub-blocks and tested to identify flaws in the different sections of the software system(Mathur 1991). White-box originates from the ability of the tester to see the implementation of the system being tested. White-box testing is also referred to as "glass" testing, and structural testing.

Advantages of white box testing

As aforementioned, white-box testing is done by a developer who has knowledge of the implementation of the code being tested. This encourages the developer to carefully reason about the implementation of the piece of code and its effect on functionality and performance(Mathur 1991). The negative effects can be identified by uncovering the hidden errors in the implementation. The knowledge of the implementation of data can help the user developer know all possible inputs and required outputs which can be used in functional testing of the software system(Mathur 1991). By the developer going through the code, he/she can optimize it for better performance(Mathur 1991). Furthermore, unrequired extra lines of code can be eliminated from the system. Finally, this testing method can be used to test complex logic algorithms.

- It can be used to optimize the code, not only to find out defects.
- Encourages careful reasoning about the system implementation
- Hidden errors in the code can be identified
- All possible inputs and outputs can be tested
- It provides maximum coverage of the system, as tester is aware of how the system is internally implemented and designed.
- It is suitable to test complex logic algorithms.

Disadvantages of white-box testing

Although white-box testing has many advantages, its disadvantages mainly lie in its costs and time efforts – the test requires more time to assess the code line by line(Lewis 2004). As a result, it becomes a tedious task in the event that a large

software system is being tested(Lewis 2004). The test requires experienced developers to be involved. Since the developer of the system cannot point out all his/her errors, another experienced developed has to be used to identify flaws in the system. This makes the process very expensive(Lewis 2004). The test script must be developed based on the current state of the code. In the event that the code is changed, another test script must be developed to cater for the changes(Lewis 2004). These disadvantages are summarized below:

- It requires more time and effort to be performed
- It is expensive.
- Skilled testers are needed to perform white-box test, which leads to extra cost
- In case of large code, white-box testing is exhaustive
- In case of changing source code, test script is required to be updated often
- It is impossible to look into every line of code thus providing a challenge in uncovering all errors

Examples of white-box testing for my version of Game 2048

Test 1- Boundary testing

Block of code being tested

```
if(data[r][c]==0){

        return true;

}

else{

        return false;

}
```

Test data

To test this block of code, we need 3 data sets to test both paths/flows, first that array element data[r][c]=0; and second is that array element data[r][c] equals 2 "any positive value", and third one is that array element data[r][c] equals -2 "any negative value".

Expected behavior

1- In case that data[r][c]=0 this code block should returns true.
2- In case that data[r][c]=2 this code block should returns false.
3- In case that data[r][c]=-2 this code block should returns false.

Case	data[r][c]	Output
1	0	true
2	2	false
3	-2	false

Test 2 – Decision paths testing

Block of code being tested

```
if(values[r][c]<10){

        System.out.print("[   "+values[r][c]+"]");

} else if(values[r][c]<100){

        System.out.print("[  "+values[r][c]+"]");

}

else if(values[r][c]<1000){
```

```
        System.out.print("[ "+values[r][c]+"]");
}
else{
        System.out.print("["+values[r][c]+"]");
}
```

Test data

For this block of code, we need four test data sets, first one is a value = 0 “any value less than 10”, second one is value = 32 “any value greater than 10 and less than 100”, third one is value = 256 “any value greater than 100 and less than 1000”, and last one is 1024 “any value greater than 1000”.

Expected behavior

1- In case that data[r][c]=0 this code block should prints “[0]”.
2- In case that data[r][c]=32 this code block should prints “[32]”.
3- In case that data[r][c]=256 this code block should prints “[256]”.
4- In case that data[r][c]=1024 this code block should prints “[1024]”.

Case	data[r][c]	Output
1	0	[0]
2	32	[32]
3	256	[256]
4	1024	[1024]

Summary of the comparison between Black-box testing and White-box testing

Black-box Testing	White-box Testing
It focuses on the functional requirements of the system under test	I focuses on the internal logic of the system
The test can be carried only after the completion of the functional implementation of the system	It can start as soon as the implementation starts
Attempts to point out flaws in: ➢ Incorrect/missing functions ➢ Performance ➢ User Interface ➢ Data structures ➢ Database access	Attempts to point out flaws in: ➢ Internal logic ➢ Program status
Assumes control structure of procedural design	Obtains test cases from the control structures
The tests are done at the user interface	Tests are performed in the code based on procedural detail
Errors are pointed out	Flaws in logical paths are pointed out
Demonstrates that functions are operational	Logical paths are examined to ensure the code is functional

Conclusion

To achieve the objective of software testing, “white-box and black-box testing” techniques are important and are meant to find different types of defects in a software application. This means that they complement rather than substitute one another. Consequently, successful software testing must take into account the two testing methods. The advantages and disadvantages of each of the methods have been pointed out. It is imperative to note that carrying out both tests on a single software application can reduce the disadvantages tremendously. Generally, black-box testing focuses on the functional requirements of the application while white-box testing focuses on the internal logic that ensures the functional units perform their functions successfully and efficiently.

References

Beizer, B. 1995. *Black-box testing: techniques for functional testing of software and systems* John Wiley & Sons, Inc.

Lewis, W.E. 2004. *Software testing and continuous quality improvement* CRC press.

Mathur, A. P. Performance, effectiveness, and reliability issues in software testing, IEEE, pp. 604-605.

Myers, G.J., Sandler, C., & Badgett, T. 2011. *The art of software testing* John Wiley & Sons.

Schroeder, P.J. & Korel, B. 2000. *Black-box test reduction using input-output analysis*, 25 ed. ACM.

Stepp, M. nd, Black/White-Box Testing, CSE 403 Lecture 13, Computer Science and Engineering, University of Washington. available at <<https://courses.cs.washington.edu/courses/cse403/13sp/lectures/13-blackwhiteboxtesting.pdf>>.