A Comparative analysis on Black Box Testing Strategies

Pramod Mathew Jacob
Research Scholar
School of Information Technology & Engineering
VIT, Vellore
pramod3mj@gmail.com

Dr. M. Prasanna
Associate Professor
School of Information Technology & Engineering
VIT, Vellore
prasanna.m@vit.ac.in

***Abstract*—Software rules in almost all fields of science and technology. Now a days the user classes of a particular software are comparatively high based on their acceptance and quality. So it is the onus of the software developer to ensure that the software is bug free and it will perform the desired functionalities without fail. Software testing is an important activity in Software Development Life Cycle (SDLC) phases. In traditional Waterfall model, testing is the life cycle activity that is performed after the design and coding phase. The various research analytics states that nearly 30% effort of entire software development is used for performing testing activities. So testing activities plays a vital role in developing a quality software. Software testing mainly consists of three categories: Unit testing, Integration testing, System testing. Unit testing basically follows two approaches, Black box testing and White box testing.This paper analyses the various Black box testing methodologies and the issues related to the same.**

Keywords— Software Testing; Software Development Life Cycle; Unit Testing; Integration Testing; System Testing; Boundary Value Analysis; Equivalence Class Partitioning;

I. INTRODUCTION

Software testing is a technical activity which involves the considerations of economics and human psychology. The myth among some testers regarding testing is that "*Testing is an SDLC activity to ensure that the software is completely bug free*" or "*Testing is the process of ensuring that the developed software will serve all its functional and non-functional requirements*"[1]. Though these two statements look theoretically perfect, a good tester should keep the following definitionwhile testing. "*Software testing is the process of evaluating a software or program with the intention to identify bugs*"[2]. This definition implies the following.

- It's not about proving the fact that the software is error free.
- A tester should assume that the program contains errors before testing.

Software testing can also be stated as the process of validating and verifying that a software or application meets the business and technical-functional requirements that guided in its design and development. Validating and Verifying (V&V)[3] is the process of ensuring that a software meets the requirements mentioned in Software Requirement Specification (SRS) document and that it fulfills its intended functionality. It can be considered asa methodology to ensure software quality control. The terms can be defined as follows:

Verification: It is the process of ensuring whether the software product is built in the right manner.

Validation: It is the process of ensuring whether the developed software product is what they expected.

In order to analyze various testing techniques, first we have to know the basic terminologies used in software testing:

Test case: A test case is a set of test data that is to be inputted, expected results for each test case and resultant conditions, designed for a particular test scenario in order to ensure compliance against a specific functional requirements [4].

Consider the example test cases for the program to find the largest among two input integers. Then it can have the following test cases.

Integer 1= 3 and Integer 2= 4

Integer 1= 4 and Integer 2= 3

Integer 1= -3 and Integer 2= -1

Integer 1= -1 and Integer 2= 0

Integer 1= 3 and Integer 2= 3

Test suite: It is mathematically a set which consist of all the test cases as set elements. A test suite may have infinite number of test cases. A good test suite should have minimal test cases which covers most errors. For the above mentioned program to compute largest of two integers, the test suite can be defined as follows:

{(3, 4), (4, 3), (-3, -1), (-1, 0) (3, 3)}

Error: Error is the degree of mismatch from actual result and expected result.It represents mistake made by code developers [5].Though error and bug sounds synonyms, they are not exactly the same. Error is the mistake found by tester. When developer accepts this mistake, then it is called as a bug.

978-1-5090-1987-8/16/$31.00 ©2016 IEEE

Fault: Fault is an incorrect instruction, function or data interpretation in a computer program which makes the program to generate an unexpected result. Fault leads to error.

Bug: Bug is a fault in the code block which generates an unintended result. It is normally the mistake accepted by the developer.

Failure: Failure is the inability of a software system to perform its expected functional and non-functional requirements. Execution of a fault leads to failure.

Defect: A defect is a mistake committed by programmer in coding or logic that causes a program to generate incorrect or deviated results.Normally a defect is detected when a failure occurs.

II. SOFTWARE TESTING METHODOLOGIES

The software testing activities includes the following phases: [6]

- Design an appropriate test suite that executes all the Lines of Code (LOC).
- Execute each test case to find the errors or mistakes in the code block.
- Identify the source of error by examining the test result.
- Modify the program instructions to fix the error.

A software product normally undergoes three levels of testing.

- Unit testing or Modular testing
- Integration testing
- System testing

Software industry follows modular programming. Modular programming is a software design strategy in which the entire software is broken into sub programs or modules based on their functional attributes, so that each module performs a unique functionality. The various modules are designed in such a way that they are loosely coupled and highly cohesive.Coupling is defined as the degree of interaction between two modules. If two modules are interacted with larger data, then they are tightly coupled, else loosely coupled. Cohesion can be defined as the functional strength of a module [7]. If all the functions in module work together to achieve a common objective, then the module possess high cohesion. A module which possess high cohesion and low coupled is called as functionally independent module. Thus these various individual sub programs or modules are to be tested separately. This process of testing individual modules is called Unit testing [6].

The individually unit tested modules are combined or integrated together to form the software. Testing performed at this level of integrating modules is termed as Integration testing. Integrating modules can be in either bottom up, top down or mixed. Based on the hierarchy in which the modules are integrated, Integration testing is classified into four as Top down integration, Bottom up integration, Mixed / Sand-witch integration and Big bang integration.After integration the resultant software should perform all the desired functionalities. This can be ensured by performing System testing.

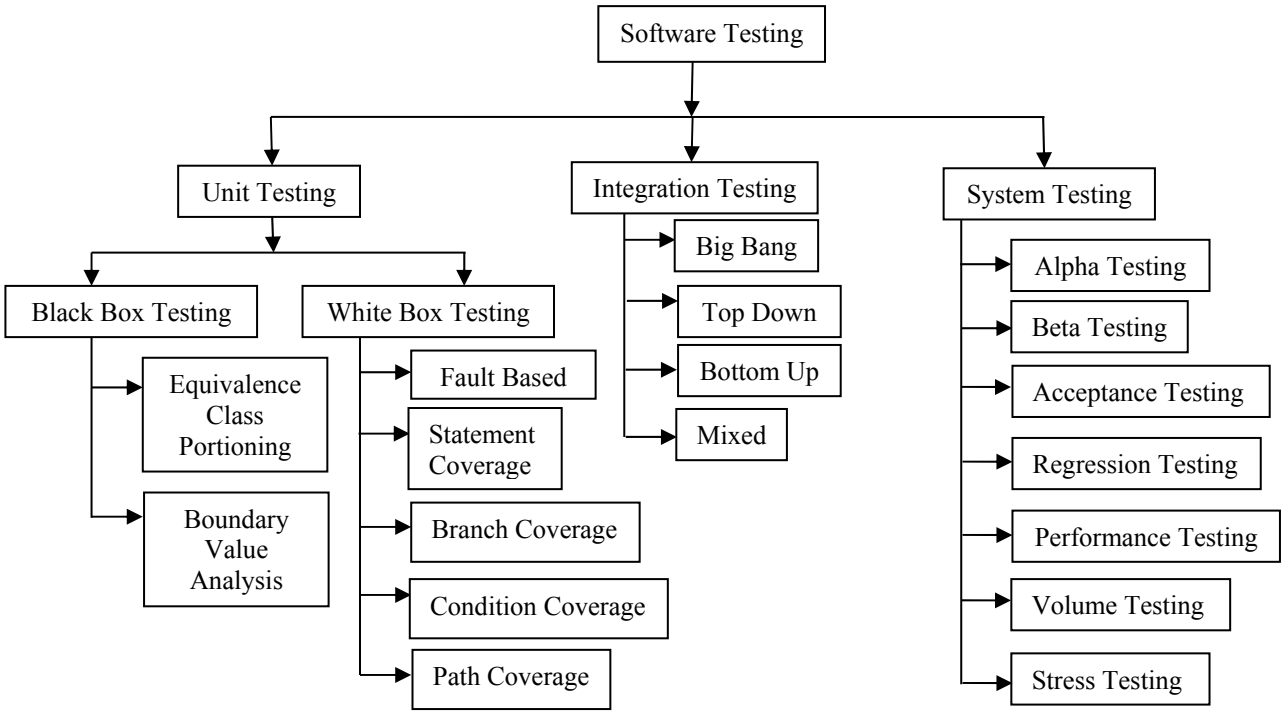

Fig. 1. Classification of Software Testing

III. UNIT TESTING

We analyzed the various modular testing methodologies by using suitable sample program code. Consider the Sign up page of a particular website with the following fields:

Name: char [50]

Gender: char

Date of Birth: DD/MM/YYYY

Username: string

This sign up page can be considered as a module of this software. Thus after developing this module we have to ensure that this unit executes successfully by performing its intended functionality. Unit testing is applied first, which consist of Black box testing and White box testing. Black box testing checks the functional aspects of a software system whereas White box testing verifies the logical aspects of the software system.

Black box tester need not know about the internal logic or program structure, because the internal logical aspects are unknown to the tester. It's just like the internal structure is covered up from the tester by using a black box.The testers have no knowledge of how the system or component is structured inside the box. In black-box testing the tester is concentrating on what the software does. They won't focus on how the software does it [7]. The comparison between Black box and white box testing strategies are shown in Table I.

TABLE I. COMPARISON OF BLACK BOX AND WHITE BOX TESTING

Parameter	Black box testing	White box testing
Focuses on	Functional requirements of the software	Internal logic of the program
Applying stage	Later or final stages of testing	Early stages of testing
Usage of control structure of procedural design	Not used	Used
Application area	Bigger monolithic programs	Testing small program components or modules
Performed by	Independent software testers	Experienced or trained Software developers
Programming knowledge	Not required	Required
Primary aim	To uncover errors and validate software	Exercise specific set of conditions ,loops or path
Applicable levels	Acceptance testing, system testing	Unit testing, Integration testing

We analyzed our test module by applying the following two black box testing methodologies

- Equivalence Class Partitioning (ECP)
- Boundary Value Analysis (BVA)

A. Equivalence class partitioning

A black-box testing method that divides the input domain of a program into classes of data from which test cases are derived [8].The challenge for the tester is to design a minimal test suite. In this methodology the entire input set is partitioned into various subsets or classes. Each class represents a set of test inputs with similar features and specifications. Each subset or classes may be valid or invalid based on the input constraints.From each subset, test cases are selected so that the largest numbers of attributes of an equivalence class are exercise at once. The concept behind this testing methodology is to divide a set of test conditions into groups or classes that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'. Equivalence partitions are also called as equivalence classes. In Equivalence Class Partitioning (ECP) technique we need to test only one condition from each subset or partitioned class. This is because we are assuming that all the conditions in one class will be treated in the same way by the software. If one condition in a partitioned class works, we assume all of the conditions in that partition will work, and if one of the conditions in a partitioned class does not work, then we assume that none of the conditions in that partition will work.

Consider an application program which accepts only two digit integers. Then the valid equivalence class is integers from 10 to 99. The invalid partitions in this case are decimal numbers, integers less than 10, integers greater than 99, non-numeric characters.

The main challenge for the tester to perform ECP is to define the various partitions for the input data set. Partitions should be carefully designed, since we exercise a single member from each partition [9]. In order to performequivalencepartitioning the tester should followtheguidelinesmentionedin Table II.

TABLE II. EQUIVALENCE CLASS COUNT FOR VARIOUS INPUT CASES

Input condition specification	Number of valid classes	Number of invalid classes
Range of values	1	2
Specific value	1	2
Element of a set	1	1
Boolean	1	1

Based on the above guidelines, the sign up page fields can be analyzed as follows.

1) Field Name: Name - char [50]

The character set denotes a set of alphabets and white space. Let the character set

C={A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z,' '}

Then there will be one valid equivalent class which accepts all the elements of set *C* and an invalid equivalence class is the complement set *C*'. So we can choose any random element from both partitions to design the test suite.

Test suite= {C, 4}

Where *C* is an element from valid partition and 4 is an element from invalid partition.

2) Field name: Date of Birth -month field

The month field of a DOB field can take the values ranges from 1 to 12. Though it represents range of values there will be one valid partition and two invalid partitions. Let *'m'* denotes the month value. Then

Valid class: $1 \leq m \leq 12$ (Values range from 1 to 12)

Invalid class 1: $m<1$ (Values less than 1)

Invalid class 2: $m>1$ (Values greater than 12)

So the minimal test suite consists of 3 test cases representing each partition.

Test suite= {-4,6,15}

The same can be applied for date field in various combinations based on the month value.

3) Gender : char

The gender field has three possible inputs. Male (M), Female (F) or Other (O). Let set G= {M,F,O} represents the valid class, then G' represents the invalid equivalence class.

B. BOUNDARY VALUE ANALYSIS

Most of the program developers make errors at the boundary values while writing conditional statements [10]. Boundary Value Analysis(BVA) is based on testing at the boundaries between various sub classes or partitions. Here we have both valid boundaries (in the valid class) and invalid boundaries (in the invalid class).A number of errors occur at the boundaries of the input domain is comparatively more than that in the "middle". It selects test cases at the edges of each subclass. The tester should keep in mind the following guidelines while performing BVA [11].

- If an input condition specifies a range of values between *m* and *n*, test cases should be designed with values *m* and *n* as well as values just above and just below *m* and *n.*

  Test suite = {m, n, m-1, n+1}

- If an input condition specifies a number of values, test case should be developed that exercise the lowest and highest numbers. Numbers just above and just below the least and highest are also tested.

If internal program data structures have prescribed boundaries (e.g., an array), design a test case to exercise the data structure at its minimum and maximum boundaries.

Consider the case of input field (Date of BIRTH) in which Month as an input data. Then the valid input should be any integer from 1 to 12, since there are only 12 months in a calendar year. So if the user enters the data as '25', then the system shouldn't accept it. Thus the various partitions in this input case are

Valid partition= Set of numbers from 1 to 12

First invalid partition= Set of numbers less than 1

Second invalid partition= Set of numbers greater than 12

In the above case, in BVA the boundary values are 1 and 12. We have to choose a number just below the minimal condition value and a value just above the maximum condition value.Hence we should select the number just below 1 and just above 12 along with 1 and 12.

Test Suite = {0,1, 5, 12,13}

IV. TESTING ANALYSIS

We can analyze this unit testing methodologies with the following sample program in C to validate the month field of Date of Birth data.

```
void main()
{
int month;
printf("Enter month");
scanf("%d", &month)
if((month>=1)&&(month< 12))printf("Valid month");
elseprintf("Invalid month");
}
```

Fig. 2. C program to validate month field of DOB

In the above case if we apply the equivalence partitioning test suite, we got the following test status report.

TABLE III. TEST STATUS REPORT FOR EQUIVALENCE PARTITIONING

Module under test: Month field of DOB			
Test case	**Expected Result**	**Actual Result**	**Test result**
-4	Invalid month	Invalid month	Successful
6	Valid month	Valid month	Successful
15	Invalid month	Invalid month	Successful

Thus the above program code works perfect for equivalence partitioning test suite. Now we analyze the same program code using Boundary value analysis test suite. The resultant test status report is shown below.

TABLE IV. TEST STATUS REPORT FOR BVA

Module under test: Month field of DOB			
Test case	**Expected Result**	**Actual Result**	**Test result**
0	Invalid month	Invalid month	Successful
1	Valid month	Valid month	Successful
12	Valid month	Invalid month	Unsuccessful
13	Invalid month	Invalid month	Successful
5	Valid month	Valid month	Successful

BVA identifies the bug which cannot be directly identified from an equivalence class partitioning methodology.

Now consider the example test case of Gender field in the Sign up page. Boundary value analysis cannot be applied in this case. The gender field represents a set of valid inputs 'M' and 'F'. In this case the input variable represents a set, not a

range of values. Here BVA fails miserably whereas Equivalence partitioning can be performed [12].

In order to unit test a module, a good tester should follow both equivalence class partitioning and boundary value analysis to ensure the reliability of a module.Both these black box testing methodologies ensure that the system should meet its specified functionalities [13]. Though internal logic or program structure is not considered, experienced testers are not required to perform Black box testing.

V. RESULTS

From the analytics it is possible to derive the following relationship between input value and the number of test cases. In BVA if there are *'N'* input variables in a software then there will be *4N+1* test cases in the minimal test suite. This can be proved by an example test input with one variable. Then the test suite may be as follows:

Test suite= {lower, lower-1, upper, upper+1}

For a single variable input there will be the above said four test cases. In order to design a minimal test suite we should include a value in the *mid –range* too. So there will be 5 test cases in the minimal test suite as shown below.

Minimal test suite= {lower, lower-1, normal, upper, upper+1}

In Equivalence class if there are *'N'* partitions for the input set, then there will be *'N'* test cases in the minimal test suite.

Thus the analytical study on black box testing strategies proves that the test cases are comparatively more in BVA than that of Equivalence Partitioning.

We analyzed our DOB module with the standard testing metrics [15].

A) Percentage of test cases passed

This value denotes the pass percentage of the executed tests.

$$Test\ case\ pass\% = \left(\frac{No.of\ test\ cases\ passed}{Total\ no.of\ test\ cases} \right) \times 100 \quad (1)$$

In ECP, Test case pass % = (3/3)× 100 = 100%

BVA, Test case pass % = (4/5)× 100 = 80%

B) Defect Leakage

Defect leakage is the number of defects left uncovered after the test passes to next phase.

Defect leakage in ECP = 1

Defect leakage in BVA = 0

C) Minimal Test suite size

The number of test cases in a test suite.

For our DOB module,

Minimal test suite size of ECP = 3

Minimal test suite size of BVA = 5

D) Time for testing

Testing time is estimated based on the test suite size and coverage area.Though the number of test cases is more in BVA, the time for testing will also be more. Hence testing effort is also more for BVA than that of EP.

E) Cost of finding defect in testing

$$Cost\ of\ finding\ defect = \frac{Total\ effort\ spent\ on\ testing}{Defects\ found\ on\ testing} \quad (2)$$

$$Total\ effort = Total\ Function\ points \times Estimate\ defined\ per\ function\ point\ based\ on\ task\ weightage \quad (3)$$

Consider that the effort for executing each test case is 1.

Then in Boundary value analysis,

Cost of finding defect = 5/ 1 = 5 whereas in ECP,

Cost of finding defect = 3/0 = 0

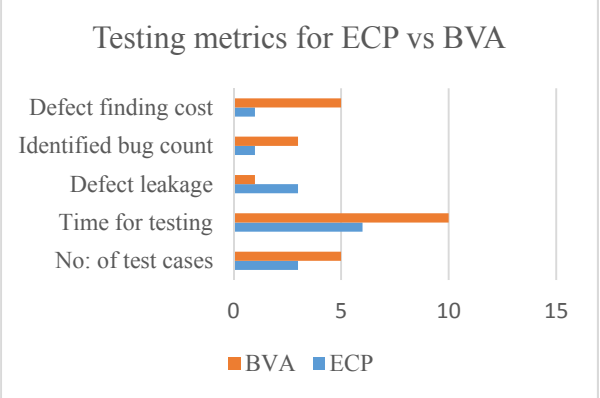

Fig. 3. BVA vs Equivalence Partitoning

Though the graph highlights BVA in most parameters, there are cases where Equivalence class partitioning succeeds and BVA fails. But the analytical study shows that increased bug count is in BVA. The increased test cases in BVA improves the bug identification rate, but is restricted for range of values input [14]. So ECP succeeds if the input value is boolean or set elements or a particular value. BCP has more defect leakage compared to ECP.

The results can be tabularized as shown below.

TABLE V. COMPARISON OF ECP AND BVA

No.	Parameter	Equivalence Class Partitioning	Boundary Value Analysis
1	Number of Test cases	Less	More
2	Quality of testing	Low	High
3	Identified bug count	Low	High
4	Testing time	Low	Slightly High
5	Cost effectiveness	Less	More
6	Testing Effort	Less	More
7	Defect leakage	More	Less

VI. CONCLUSIONS

In modular programming methodology, Black box testing plays a vital role. Equivalence class partitioning and Boundary value analysis methodologies are not alternative testing methods. Instead BVA is a test case design methodology that complements the ECP. Each methodology has its own pros and cons. The main limitation of BVA is it can be applied for the test case which accept an input range whereas the ECP can be applied on various sets or range of input values. Though BVA is time consuming, its bug identification rate is comparatively higher. The application of both testing strategies improves the reliability as well as the quality of the software which may lead to the overall customer satisfaction.

REFERENCES

[1] B. Beizer, "Software Testing Techniques". London: *International Thompson Computer Press*, 1990.

[2] Glenford J. Myers, "The Art of Software testing", Second Edition, Wiley India Edition.

[3] B. Beizer, "Black Box Testing", New York: John Wiley & Sons, Inc., 1995.

[4] D. Galin, "Software Quality Assurance", Harlow, England: Pearson, Addison,Wesley, 2004.

[5] SoftwareTestingOverview,available:http://www.tutorialspoint.com/software_testing/software_testing_quick_guide.htm

[6] SriivasanDesikan, Gopalaswamy Ramesh, "Software Testing: Principles and Practices", Pearson.

[7] Rajib Mall, "Fundamentals of Software Engineering",Third edition, PHI

[8] Ian Sommeriele, "Software Engineering", Addison Wesley.

[9] Pressman, "Software Engineering –A Practitioner's Approach".

[10] Pankaj Jalote , "An Integrated Approach to Software engineering", Narosa Publication.

[11] T.H.Shivkumar, "Software Testing Techniques", Volume 2, Issue 10,ISSN:2277 128X.

[12] Jovanovi,ć, Irena,"Software Testing Methods and Techniques", Page No-30-41.

[13] Software Engineering Economics, Prentice-Hall, 1981.

[14] Van Vleck, T., "Three Questions About Each Bug You Find", *ACM Software Engineering Notes,* vol. 14, no. 5, July 1989.

[15] http://www.softwaretestinghelp.com/software-test-metrics-and measurements/