A Comparative Analysis On Equivalence class partitioning And Boundary value analysis

Akshatha .V¹, V. Illango²
¹PG Scholar, New Horizon College Of Engineering
²Professor, New Horizon College Of Engineering

Abstract - The purpose of this paper is to carry out a detailed review on the large amount of information available on two testing techniques which fall under black box testing methodology. A detailed analysis of equivalence class partitioning and boundary value analysis has been carried out. These techniques have been comprehensively unfolded and also the strengths and weaknesses have been highlighted. This paper can be studied before carrying out any empirical study regarding the efficiency of these two testing techniques. Then, it’s only after these analytical and empirical investigations that we can come up with some sort of solid framework to effectively compare these two testing techniques with each other and also with other testing techniques.

Keywords - Software Testing, Unit Testing, Integration Testing; System Testing, Boundary Value Analysis, Equivalence Class Partitioning.

I. INTRODUCTION

Software testing is a mechanism which helps us expose errors in a software and forms an important part of the software development process. Although it’s widely accepted that it’s impossible to deliver a fault free software, that does not mean that we can deliver an unsatisfactorily tested software. In fact testing should be done until the product is valid and verified. Verification checks whether the system in entirety works as per the specification and validation checks whether the software works according to the customers’ requirements [1]. Testing obviously takes a share in the time and resources that are available but it is only because of proper testing that a good product can be developed.

The main aim of testing should be to break the software and to find errors in it rather than to try and pass the software as fault free. Now there are various types of testing approaches and in order to understand which testing technique to use and where in software development lifecycle to use it we need to analyze them before trying to bring them on a hierarchy. Before using any software testing technique it is of given importance to have a detailed theoretical knowledge of that testing technique[1].

Software testing is broadly classified into two basic methodologies they are functional testing and structural testing[23]. Functional testing methodology uses the black box approach of designing test cases wherein test cases are designed on the basis of specifications only and as the word black box signifies, the tester has no concern with the internal structure of the program under consideration., whereas structural testing methodology uses the white box approach to design test cases and the internal structure of the program[24]. Both the methodologies are imperative to software testing. Also both these methodologies have many different approaches to do the work they are meant to do.
In this paper we have analyzed two of the approaches which fall under black box testing[19][20]. We have studied equivalence class partitioning and boundary value analysis together as they are closely related and complement to each other with the help of thorough literature survey[3]. Along with decision table testing they form the basis of functional testing methodology. In the first section of the paper we have fully analyzed the boundary value analysis and equivalence class partitioning have been comprehensively unfolded. Moreover, the steps to generate the test cases have been mentioned as well. These techniques are crucial to testing and it can be seen why they are still widely used by the testing teams. A review of equivalence class partitioning and boundary value analysis techniques was done by studying the vast literature available on it. We are using grounded theory type of research based upon carrying out review of literature.

There are various test case design techniques as far as black box testing is concerned mentioned below, but given the scope of this paper we will be studying equivalence class partitioning and boundary value analysis. Also, test cases are developed using various software testing techniques so that we can save ourselves from the nuisance of exhaustive testing which is quite impractical. Now, equivalence class partitioning and boundary value analysis are studied together because they are hugely related to each other although they do differ from one another in many ways.

![Software Testing Models](image)

**Fig. 1. Software Testing Models**

### 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[18]. The comparison between Black box and white box testing strategies are shown in Table I.

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

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[2]. 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[21]. 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, nonnumeric 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[3]. In order to perform equivalence partitioning the tester should follow the guidelines mentioned in Table II.

### TABLE II. EQUIVALENCE CLASS COUNT FOR VARIOUS INPUT CASES

<table>
<thead>
<tr>
<th>Input condition specification</th>
<th>Number of valid classes</th>
<th>Number of invalid classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range of values</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Specific value</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Element of a set</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Boolean</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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:** $m>12$ (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.

**Steps to generate test case for equivalence class partitioning :**

i. Firstly, identify the input domain of the program, after properly studying the requirements, which helps identify the input and output variables, their types, and any conditions associated with their use.

ii. Secondly, divide the input domain into various categories where the participants of every category have some sort of relation with each other.

iii. Thirdly, we can expect that every test case from a category displays the same behavior. Every category is thus, a separate equivalence class and we partition the set of values of each variable into disjoint subsets which cover the entire domain, based on the expected behavior.

iv. Fourthly, sometimes equivalence classes are combined using the multi-dimensional partitioning approach but usually this step is omitted. Nonetheless, if we do combine the equivalence classes its helps create useful tests.

v. Lastly, infeasible equivalence classes, which contain a combination of input data that cannot be generated during test need to be identified. Constraints in the requirements render certain equivalence classes infeasible[4][5][6].

**Types of equivalence class testing:**

- **Weak normal equivalence class testing:** It is based on single fault assumption which means that an error is rarely caused due to two or more faults occurring simultaneously meaning that the variables are independent. Only one variable is taken from each equivalence class[7].

  ![Diagram](image.png)

  *Fig. 2. One variable is taken and that too only from valid equivalence classes.*

In the figure above one variable is taken from each equivalence class. With respect to the figure the square boxes represent the test cases and the test data’s are indicated by each dot. Each class consists of one dot which means that there is one element for each test case. Hence, the figure shows that we will have the same number of weak normal equivalence class test cases as the classes in the partition.

- **Strong normal equivalence class testing:** Here it is assumed that errors result in a combination of faults i.e. multiple fault assumption, where the variables are not independent of each other. So, we test every combination of elements formed as a result of the Cartesian product of the equivalence relation [7].

In the figure above, each test data is equally distributed among all the test cases. With respect to the figure the square boxes represent the test cases and the test data’s are indicated by each dot. Each class consists of one dot which means that there is one element for each test case. These test cases work as same as truth tables in digital logic’s making sure that we have got the similarities between these truth tables and our pattern of test cases. The Cartesian product represents the ‘completeness’ in the following ways:
1) Equivalence classes are covered.
2) We have possible combination of inputs at least one from each.

**Advantages of equivalence class partitioning:**
Equivalence class partitioning reduces redundancy and ensures completeness of testing. It reduces effort as compared to exhaustive testing and gives if at better results in lesser time.

**Disadvantages of equivalence class partitioning:**
Equivalence class partitioning makes an assumption that the data in the identical equivalence class is processed in the similar way be the system. It needs to be supplemented by boundary value analysis[8][9]. It is more concerned with data dependencies and testing similar inputs and outputs the same way by grouping them in classes. This reduces the test cases but at the cost of an increased effort to group them. It is a sophisticated method as it is concerned with the values inside the domain.

**B. Boundary value analysis**
Most of the program developers make errors at the boundary values while writing conditional statements[4]. 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"[22]. It selects test cases at the edges of each subclass. The tester should keep in mind the following guidelines while performing BVA [16].

- 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\}.

**Fig. 4. Boundary value analysis.**

**Steps to generate test case for boundary value analysis:**

1. Firstly, find out values for each of the variables that should be exercised during testing from the specification meaning we divide the input domain using uni-dimensional partitioning. This leads to as many partitions as there are input variables. A different method is creating a single partition of an input sphere using multi-dimensional partitioning. Also several sub domains are also formed during this step.
2. Secondly, identify the boundaries and select values on or close to the boundary i.e. minimum value, just above minimum value, maximum value, just below maximum value nominal (average) value.
3. Thirdly, select the test data such that every boundary value occurs in at least one test input[4][5][6].

**Types of boundary value analysis:**

- **Robustness testing:** Here invalid values are selected and the response of the program is checked. We check the values minimum, just above minimum, just below minimum, maximum, just above maximum, just below maximum and nominal (average) value. Robustness testing focuses on exception handling and is highly advocated when it needs to be checked[12].

- **Worst case testing:** Here single fault assumption theory is not considered. Single fault assumption theory of reliability is that failures are rarely a result of simultaneous occurrence of two or
more faults. But here we consider that failure is due to the occurrence of more than one fault simultaneously. Input values are minimum, just above minimum, average, just below maximum and maximum. But the most important fact here is that the restriction of one input value at any of the mentioned values and other input values must be at nominal, is not valid in worst case testing. It is generally used for situations where a high degree of testing is required[12].

Robust- worst case testing: It is also one of the special cases of boundary value analysis and here the states are the same as in robustness test. The difference lies in the number of test cases generated. It produces the largest set of test cases so far and requires a lot of effort. The difference between these various extensions of boundary value analysis is actually in the number of test cases that they generate [12].

Advantages of boundary value analysis :
Boundary value analysis can be used at unit, integration, system and acceptance test levels. Experience has shown that such test cases have a high probability of detecting a fault in the software, where test cases have input values close on the boundary. It is not concerned with the data or logical dependencies as it is a domain based testing technique [13]. It is computationally less costly as far as the creation of test cases are concerned [12].

Disadvantages of boundary value analysis :
In boundary value analysis inputs should be independent, this is what restricts it’s the applicability making no sense for boolean variables and no choice for nominal value, just above and below boundary values. So, it’s suitable only for ranges, sets and not for logical variables. The number of test cases generated is higher as compared to other functional methods. It doesn’t consider the nature of the function or the dependencies and the variables [12]. It’s performing cost is lesser but higher test cases required than equivalence class portioning [17].

IV. TESTING ANALYSIS
We can analyze this unit testing methodologies with the following sample program in C to validate the month field of DateOfBirth data.

```c
void main()
{
    int month;
    printf("Enter month");

    scanf("%d", &month) if((month>=1)&&(month< 12))printf("Valid month");  else printf("Invalid month");
}
```

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**

<table>
<thead>
<tr>
<th>Test case</th>
<th>Expected Result</th>
<th>Actual Result</th>
<th>Test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>Invalid month</td>
<td>Invalid month</td>
<td>Successful</td>
</tr>
<tr>
<td>6</td>
<td>Valid month</td>
<td>Valid month</td>
<td>Successful</td>
</tr>
<tr>
<td>15</td>
<td>Invalid month</td>
<td>Invalid month</td>
<td>Successful</td>
</tr>
</tbody>
</table>

In the above table, random input values have been taken and output value is compared with the expected result and actual result. If the expected result and the actual result are same then the test result will be successful and if the expected result and actual result are not same then the test result will be unsuccessful. For example, let us take the first test case as the input number which is -4. When the input is taken in the above code in place of month, it will not satisfy the if statement and prints invalid month i.e.

The first condition if (month >= 1) which is false because -4 is not greater than 1.
The second condition if (month < 12) which is true because -4 is less than 12.

But, if you observe the code an if statement has AND operator which means both the conditions has to be true. In our test case where we have taken -4 as the input the first condition is false and the pointer does not go to the second condition as the first condition is false because it is AND operator.

Likewise all the other test inputs has to be checked with both the conditions and has to compare with the expected result and actual result.

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 boundary value analysis**

<table>
<thead>
<tr>
<th>Test case</th>
<th>Expected Result</th>
<th>Actual Result</th>
<th>Test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalid month</td>
<td>Invalid month</td>
<td>Successful</td>
</tr>
<tr>
<td>1</td>
<td>Valid month</td>
<td>Valid month</td>
<td>Successful</td>
</tr>
<tr>
<td>12</td>
<td>Valid month</td>
<td>Invalid month</td>
<td>Unsuccessful</td>
</tr>
<tr>
<td>13</td>
<td>Invalid month</td>
<td>Invalid month</td>
<td>Successful</td>
</tr>
<tr>
<td>5</td>
<td>Valid month</td>
<td>Valid month</td>
<td>Successful</td>
</tr>
</tbody>
</table>

In the above table, random input values has been taken and output value is compared with the expected result and actual result. If the expected result and the actual result are same then the test result will be successful and if the expected result and actual result are not same then the test result will be unsuccessful. For example, let us take the first test case as the input number which is 0. When the input is taken in the above code in place of month, it will not satisfy the if statement and prints invalid month i.e.
The first condition if(month >= 1) which is false because 0 is not greater than 1.
The second condition if(month < 12) which is true because 0 is less than 12.

But, if you observe the code an if statement has AND operator which means both the conditions has to be true. In our test case where we have taken 0 as the input the first condition is false and the pointer does not go to the second condition as the first condition is false because it is AND operator.

Likewise all the other test inputs has to be checked with both the conditions and has to compare with the expected result and actual result.

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 is 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[10]. 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[11]. 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.

- Percentage of test cases passed - This value denotes the pass percentage of the executed tests.

\[
\text{Test case pass} \% = \left( \frac{\text{No. of test cases passed}}{\text{Total No. of test cases}} \right) \times 100
\]  \hspace{1cm} (1)
In ECP, Test case pass % = (3/3)× 100 = 100% and BVA, Test case pass % = (4/5)× 100 = 80%.

- **Defect Leakage** - Defect leakage is the number of defects left uncovered after the test passes to next phase.
  Defect leakage in ECP = 1 and Defect leakage in BVA = 0.

- **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

- **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.

- **Cost of finding defect in testing**
  Cost of finding defect = \( \frac{Total \ effort \ spent \ on \ testing}{Defects \ found \ on \ testing} \) (2)

  Total effort = Total Function points × Estimate defined per function point based on task weightage (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

![Testing metrics for ECP vs BVA](image)

**Fig 5. Graph representation of testing metrics for ECP vs BVA**

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
### VI. CONCLUSIONS AND FUTURE WORK

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.

In this paper black box testing is studied in a detailed manner and is widely used in testing software. It is of importance to check which testing technique to use in which stage of software development life cycle (SDLC) apart from studying in detail which black box testing techniques stands where on the testing spectrum and here we are starting with equivalence class partitioning and boundary value analysis and how they are related to each other. These techniques are known to form the most important point at issue of black box testing methodology and a detailed review with the result has been carried out in this paper and it becomes a starting point to carry out empirical study of the various testing techniques, to compare them and somehow understand the pros and cons of each of them. Now since the analysis is done in this paper further work would be required to gauge them with the help of studying the experiments which evaluate the effectiveness of these techniques. It would rightfully take the grounded research of this paper to a more concrete and viable conclusion.

### REFERENCES

I. S.U. Farooq, “Evaluating effectiveness of software testing techniques with emphasis on enhancing software reliability” Ph.D thesis.
III. Pressman, “Software Engineering –A Practitioner’s Approach”.
VI. A. P. Mathur, Foundations of software testing.
VII. G. Davies, “Equivalence Class Testing” Swansea University -2007

<table>
<thead>
<tr>
<th>No.</th>
<th>Parameter</th>
<th>Equivalence Class Partitioning</th>
<th>Boundary Value Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of Test cases</td>
<td>Less</td>
<td>More</td>
</tr>
<tr>
<td>2</td>
<td>Quality of testing</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Identified bug count</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Testing time</td>
<td>Low</td>
<td>Slightly High</td>
</tr>
<tr>
<td>5</td>
<td>Cost effectiveness</td>
<td>Less</td>
<td>More</td>
</tr>
<tr>
<td>6</td>
<td>Testing Effort</td>
<td>Less</td>
<td>More</td>
</tr>
<tr>
<td>7</td>
<td>Defect leakage</td>
<td>More</td>
<td>Less</td>
</tr>
</tbody>
</table>
XIII. C. Ferriday, "A Review Paper on Decision Table Based Testing," Swansea University-CS399 - 2007
XXIV. L. Williams, “Testing overview and black-box testing techniques.”
XXV. “Software testing method (Black-box, White-Box, Grey-Box) by R. Pandey.