Equivalence Class Partitioning and Boundary Value Analysis - A Review

Asma Bhat
Research Scholar, Department of Computer Science
University of Kashmir, Hazratbal, Srinagar, Kashmir,INDIA
Email Id: asmabhat26@yahoo.com

S. M. K. Quadri
Director, Department of computer science
University of Kashmir, Hazratbal, Srinagar, Kashmir,INDIA
Email Id : quadrismk@hotmail.com

Abstract –The purpose of this paper is to carry out a detailed review on the plethora of information available on two testing techniques which fall under functional testing methodology. A detailed analysis of equivalence class partitioning and boundary value analysis has been carried out based wholly and solely on literature survey. 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 – Black box testing, boundary value analysis, dynamic testing techniques, equivalence class partitioning, functional testing methodology.

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 verifiable. 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]. Even Pareto's principle applies to software testing wherein it states that 80% of the errors discovered during testing will likely be evident in 20% of all program components, which means that a small error can lead to a number of errors in the software. There is no doubt in the fact that the software developer must make sure that the process he/she is following must deliver a reliable software but nevertheless some errors and bugs are unraveled only after proper testing. 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 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. It is for this reason that testing should be unbiased. 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 empirically trying to bring them on a hierarchy. Before using any software testing technique it is of utmost importance to have a profound theoretical knowledge of that testing technique [1].

Software testing is broadly classified into two basic methodologies-Functional testing and structural testing. 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 i.e. the source code is used to devise test cases. 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 functional testing methodology and they have been studied with the help of thorough literature survey [2] [3]. We have studied equivalence class partitioning and boundary value analysis together as they are closely related and complement each other. 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 functional testing methodology and then in the next segment 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 [2]. Mostly the detailed analysis was done consulting the book sources available and also the search string "functional testing, "black box testing"," equivalence class partitioning" and "boundary value analysis" were given and the results were filtered according to requirement; to the few considered in this paper and mentioned in the references.

II. FUNCTIONAL TESTING METHODOLOGY

Functional testing technique comes under the arena of black box testing methods and is also known by the names of specification testing, behavioral testing, data-driven testing, input-output driven testing, opaque testing, closed box testing [3][4]. Black box testing denotes the opacity of the tester which means that he/she has no access to the source code [5]. It can be applied to every stage of software testing i.e. unit testing, integration testing, etc [6]. These various names of functional testing themselves reveal that this testing deals only with the behavioral patterns of the software and is only concerned to check the correctness of the program using the data. In black box testing the composition of the program is not considered and test case is selected on the basis of functional requirements. The figure 1 explains functional testing methodology [6].
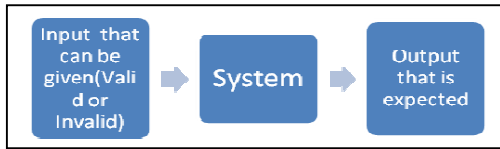

Fig. 1. Functional testing methodology in the simplest manner possible

Software is sensitive to certain input values and that helps us to check the validity of the software using the functional testing method. It is basically divided into 2 categories which are positive functional testing where valid input is given and we verify that output is correct and negative functional testing where we use a combination of invalid inputs, unexpected operating conditions and other out of bound scenarios [3]. Functional testing helps find out faulty or omitted functions, interface errors, errors in data structures or external data base access, behavioral errors, initialization and termination errors [7].

A. Advantages of functional testing methodology

It is good for a program with a large code. Code access is not needed. It separates users' perspective from developers perspective through visibly defined roles. Huge number of reasonably skilled testers can test the application having no familiarity with implementation, programming language or operating system information [8]. It has low granularity and is least exhaustive and time consuming. There is quicker test case development as well [4]. It helps find out ambiguities in specifications, if any. Test cases can be devised as soon as the specifications are completed [3].

B. Disadvantages of functional testing methodology

There is limited knowledge of tester and limited coverage. Tests are done by trial and error method and also it isn't suitable for algorithm testing. There is limited coverage and only a selected number of test cases are performed. Without clear specification test cases are not efficiently designed and thus it all comes down to unambiguous specification analysis [4]. Sometimes a tester with no information about the code might encounter a problem or a bug and not know how to correct it only due to the fact that he has no knowledge about the code. So, the advantage of having a choice to get just about any tester becomes a problem right here. Reason for a specific failure is not found out.

There are various test case design techniques as far as functional testing is concerned mentioned below, but given the scope of this paper we will be studying equivalence class partitioning and boundary value analysis [6][7][9][10][11]. 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. Functional testing techniques can primarily be divided in two types i.e. static and dynamic. Static can be further classified into performing high level or low level review of specification. Dynamic can further be classified into boundary value analysis, equivalence class partitioning, all-pairs testing, cause effect graphing, orthogonal array testing, fuzzing, error guessing, decision-table testing, combinatorial testing, model based testing and state transition table testing.

The relation will be understood as we go through the paper; however the basic differences are as follows. Equivalence class partitioning finds the number of test cases for a certain program to be tested whereas boundary value analysis determines the effectiveness of those test cases. Equivalence class partitioning checks all possible partitions for errors whereas boundary value analysis checks values at the edges because the probability of finding the error at the edges is maximum. Equivalence class method needs to be supported by boundary value analysis and not the other way round. In spite of all these differences these two testing techniques have been studied together in this paper because the relation between them is very strong and it is only fitting that they be studied together. Now, the essential point to note here is that boundary value analysis can be properly studied and understood only after fully comprehending equivalence class partitioning, as its only after equivalence class partitioning has been carried out, that boundary value analysis can be done. In really plain words the test cases that boundary value analysis creates are borne off its particular equivalence class partition only. As the names also suggest the partitions that are created in equivalence classes are further prodded in boundary value analysis and its test cases are modeled taking into consideration the edges of the equivalence class partitions. Boundary value analysis is thus like an aid to equivalence class partitioning method which helps it work smartly and reveals a lot many errors. But the irony here is that boundary value analysis supports equivalence class partitioning but itself needs its partitions in the first place to accurately identify the boundary values with which it plays.

The sections which follow unfold equivalence class partitioning method and boundary value analysis method. It would also be worthwhile to mention here that although this is

a review paper and is basically concerned only with pointing of merits and demerits but nonetheless the technical side has also been touched and we have comprehensively explained the steps to generate test cases for these two types of testing methodologies.

<center>III. EQUIVALENCE CLASS PARTITIONING</center>

Exhaustive testing includes all possible inputs to the system and is highly impractical. And it's because of this that the input domain is divided into a set of equivalence classes, so that if a specific program works appropriately for a certain value then it will apparently work properly for all the further values in that class [9][12]. One approach to partition the input domain is to consider one input variable at an instance. Herein, each input variable leads to a partition of the input domain. This is called uni-dimensional equivalence class partitioning. Then there is multi-dimensional partitioning where the input domain is considered the set product of the input variables and then association is defined on the input domain. This course of action leads to one partition consisting of several equivalence classes [13].

Basically, grouping of similar inputs or similar outputs or similar operations is done. And it is these that constitute the equivalence class partition [11]. Equivalence class partitioning is as much an art as it a science because it is subjective to a great extent and even if two different testers get two different sets of partitions it's okay as long as the bugs are properly revealed [10][11]. So, what this concept signifies is that if such classes are recognized to a high level of precision, even if one value from the equivalence class is tested it's like checking the whole program exhaustively within the boundaries of that values equivalence class [9]. This concept thus reduces the effort of the tester to a great degree and the results that are obtained are not of a lower level at all. Where behavior of inputs is expected to be alike; equivalence class can be formed and if a set of objects can be linked by relationships which are symmetric, transitive and reflexive an equivalence class is surely present [7] [9].The main goal of equivalence class partitioning is to define a test case that unravels classes of errors and that actually helps to reduce total number of test cases to be developed [7]. This is how equivalence class method helps us to find a way out of exhaustive testing but still delivers at par to the former. The fundamental idea of equivalence class partitioning method is that it either works correctly for all the values or none. In equivalence classes there is no near about similarity and they ought to be split if there is a minute dissimilarity [9]. Thus, it can easily be said that equivalence class partitioning is a heuristic approach to test a software whose state spaces, inputs and outputs have value ranges of a cardinality inhibiting exhaustive enumeration of every likely value within a test suite. Herein its meant that equivalence classes are partitions created within the input or the output domain to create disjoint sets of data which exhibit completeness and whose behaviour is expected to be the same based on the specification [12][14] .

Equivalence classes are typically formed by taking into consideration each condition specified on an input as specifying a valid equivalence class and one or more invalid equivalence class. At times we take into consideration equivalence classes in the output. Here, the idea is to have inputs such that the output for that test case lies in the output equivalence class [9]. Both input as well as output classes have been found to be effective in finding errors and both should be used in tandem with one another. Also we must always design separate equivalence class partitioning for default, empty, blank, null, zero or none condition as they need to be handled differently [11]. The efficacy of tests generated by means of equivalence partitioning for testing any application, is judged by the ratio of the number of faults these tests are able to expose to the total faults lurking in that particular application [13]. Now there are some general guidelines which are generally followed while designing equivalence class partitions as mentioned.

Input data values to a system is specified by range of value where one valid equivalence class is formed and two invalid equivalence classes are formed or it is specified by values from members of a set made of discrete elements where one equivalence class for valid input value and one equivalence class for invalid input values is formed. Another way to specify the input values is by some specific value where one valid equivalence class is formed and two invalid equivalence classes are formed .As also input values can be specified by boolean value where one valid and one invalid equivalence class is formed.

A. *Steps to generate test case for equivalence class partitioning*

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. Secondly, divide the input domain into various categories where the participants of every category have some sort of relation with each other .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 behaviour. 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. 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 [9] [10] [13].

B. *Types of equivalence class testing:*

1) 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. In the figure below, one variable is taken from each equivalence class [16].
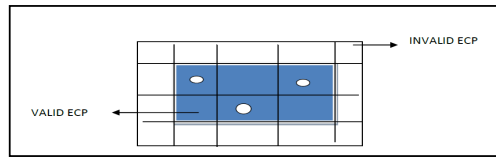

Fig. 2. One variable is taken and that too only from valid equivalence classes.

2) 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.
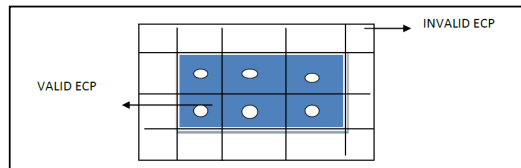

Fig.3. Test every combination of elements formed as a result of Cartesian product of the equivalence relation

3) Weak robust equivalence class testing: Here we test for one variable from each equivalence class but we authenticate and test for invalid values as well. Since it is based on the single fault assumption a test case will have one invalid value and the remaining values will all be valid [16].
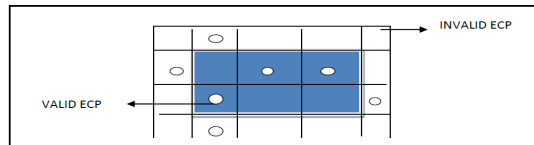

Fig. 4. Check valid as well as invalid values

4) Strong robust equivalence class testing: Here, test cases are produced for all valid and invalid elements of the Cartesian product of all the equivalence classes [16].
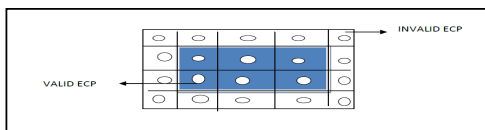

Fig. 5. Checks for all out of range combinations

C. *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.

D. *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 [17] [18]. 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 [19].It is a .sophisticated method as it is concerned with the values inside the domain.

IV. BOUNDARY VALUE ANALYSIS

Programs that work tolerably for a set of values in an equivalence class fall short on some special value which are often seen to be on the boundary of the equivalence class [9][13][20]. Programming by character is prone to bugs at boundaries. When working with boundary value analysis we play with the values on the boundary or just outside the boundary [11]. Some boundaries are quite internal to the software and might not be apparent to the end user but need to be checked nonetheless. These are called sub boundary or internal boundary conditions. There are two ways to generate boundary value analysis. One is by the number of variables and second by the kind of range [21]. A boundary value for each equivalence class includes the equivalence classes of the output as well. Figure 6 explain the phenomenon of boundary value analysis.
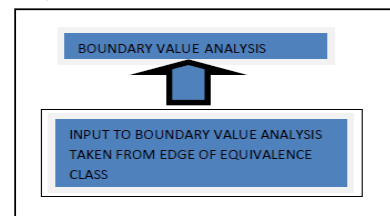

Fig. 6. Boundary value analysis

A. *Steps to generate test case for boundary value analysis*

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. 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. Thirdly, select the test data such that every boundary value occurs in at least one test input [7] [9] [10][13].
So, in order to generate test cases here the values will be taken within the valid input domain of any program which is under

consideration. As the following figure depicts, values on the x axis are taken on the y axis as well.
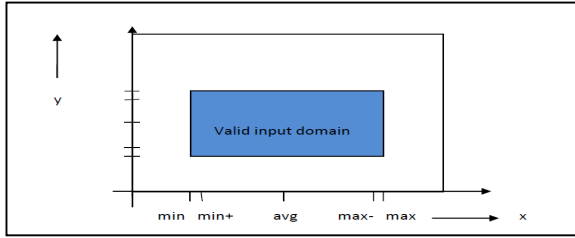

Fig. 7. Values that boundary value analysis checks

B. Types of boundary value analysis

1) 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 [20].

2) 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 [20]. 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 afore 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 [20].

3) 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 [20]. The difference between these various extensions of boundary value analysis is actually in the number of test cases that they generate [10].

C. Divide and rule approach

A fresh algorithm is devised by [22] and based on it a testing means is developed. In here a divide and rule approach is used to sever the dependencies amongst the variables to generate self-sufficient sets of variables. When there is no dependency among the variables visibly there is no influence on each other. Sets of independent variables are thus fashioned from a set of inter dependent variables. This modus operandi is used to form sets of independent variables from sets of interdependent variables plus afterward carry out conservative boundary value analysis on every independent set fashioned. In a set of inter-dependent variables, boundaries of a variable are prejudiced by the values assumed by other variables. This style of dependency amongst variables can be called boundary dependency [22].

These are the steps to create test case when looking at boundary value analysis w.r.t divide and rule approach [22].

- Foremost divide the variables into 3 categories. First category is called D i.e. set of dependent variables whose boundary values are influenced by the values of some other variable. Second category is called B i.e. set of boundary determining variables i.e. the values of these variables manipulate boundary values of other variables. Third category is I, i.e. independent variables which are neither dependent nor influential.

- For each and every dependent variable shape a set of determining variables. This means for the dependent variable whose boundary value is influenced, find the influential boundary determining variable which causes the influence here and has an effect on the dependent variable.

- Identify and outline the boundary value range of dependent variables and structure their separate sets.

- Now, outline sets of values (boundary formative variables) which influence the boundary values of dependent variables already recognized in step 3.

- For every dependent variable, produce all likely combinations of its boundary determining sets. E.g. x and y are elements of boundary determining variables. x has 2 determining sets namely x_1 and x_2. Y has 3 determining sets namely y_1, y_2, y_3. Combinations will be x_1y_1, x_1y_2, x_1y_3, x_2y_1, x_2y_2, x_2y_3.

- For every likely combination of boundary determining sets, allot a boundary value dependent set. Association between combination of boundary determining sets and the dependent boundary value set is that if any determining variable specific combination assumes a particular value, then dependent variable can assume a value from corresponding dependent variable only i.e. the dependent variable which matches the determining set combination.

- Now, produce all probable combinations of boundary determining sets of all boundary determining variables. E.g. x, y, z are elements of boundary determining variables and have 2, 3 and 2 determining sets respectively we have 12 combinations, going about similarly on the lines of step 5.
$x_1y_1g_1$, $x_1y_1g_2$, $x_1y_2g_1$, $x_1y_2g_2$, $x_1y_3g_1$, $x_1y_3g_2$, $x_2y_1g_1$, $x_2y_1g_2$, $x_2y_2g_1$, $x_2y_2g_2$, $x_2y_3g_1$, $x_2y_3g_2$

- For every grouping in step 7 assign boundary value sets for each dependent variable via associations in step 6.

- For every grouping assign all independent variables, giving all possible independent variable sets.

- For every independent variable set produced, carry out typical boundary value analysis to fabricate test cases [22].

D. 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 [10]. It is not concerned with the data or logical dependencies as it is a domain based testing technique [19]. It is computationally less costly as far as the creation of test cases are concerned [20].

E. 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 [19]. It is more proper to more "free-form" languages which are weakly typed than to strongly typed languages which necessitate that all constants or variables defined must have an associated data type. These languages were in reality created to make sure that the faults boundary value analysis discovers wouldn't arise in the first place. Even though boundary value analysis can be used with languages of this type and is constructive to a certain extent, but overall it can be seen as unsuitable for systems created using them [20]. It doesn't consider the nature of the function or the dependencies amid the variables. [20].

V. CONCLUSION AND FUTURE SCOPE

In this paper functional testing methodology was studied in a detailed manner as it falls under the arena of black box testing and is widely used in testing software. It is of immense importance to check which testing technique to use in which stage of software development life cycle apart from studying in detail which functional 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 crux of functional testing methodology and a detailed review 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

[1] S.U. Farooq, *"Evaluating effectiveness of software testing techniques with emphasis on enhancing software reliability"* P.hDthesis,from:*ojs.uok.edu.in/jspui/bit stream/1/1041/3/Umar%20PhD%20Thesis.pdf- 2012*

[2] Research Methods/Types of Research. *Wiki books, The Free Textbook Project.* Retrieved from http://en.wikibooks.org/w/index.php?title=Research_Methods/ Types_of_Research&oldid=2616361 - 2014

[3] B. B. Agarwal, S. P. Tayal & M. Gupta - *Software Engineering and testing.* Jones & Bartlett Learning, 2010.

[4] M.E. Khan, & F. Khan, "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques". *International Journal of Advanced Computer Sciences and Applications,* *3*(6), pp. 12-1 ,2012.

[5] L. Williams," Testing overview and black-box testing techniques." S*ource: http://www. agile. csc. ncsu. Edu/SEMaterials/Black Box. Pdf* - 2004.

[6] Black-box testing. In *Wikipedia, The Free Encyclopedia*, from http://en.wikipedia.org/w/index.php?title=Black-box_testing&oldid=607624827 – May, 2014.

[7] R. S. Pressman*, Software Engineering: A Practitioner's Approach*, ISBN*: 0-07-052182-4*, 1996.

[8] *"Software testing method (Black-box. White-Box. Rey-Box*)."In Tutorial Point from www.**tutorialspoint**.com/**software_testing/testing_**methods.ht m - 2014

[9] P. Jalote - An *integrated approach to software engineering.* Springer, 1997.

[10] Y. Singh - *Software Testing.* Cambridge University Press, 2012

[11] R. Patton - Software *testing.* Sams Publication, 2006.

[12] D. Galin - *Software quality assurance: from theory to implementation.* Pearson education, 2004.

[13] A. P. Mathur, *Foundations of software testing.* Source: https://www.cs.purdue.edu/homes/apm/FoundationsBook Second Edition/Slides/ConsolidatedSlides.pdf- 2008,Updated 2013.

[14] W. L.Huang & J. Peleska, " Equivalence Class Partitions for Exhaustive Model-Based Testing" - 2012

[15] R. Mall, *Fundamentals of software engineering.* PHI Learning Pvt. Ltd - 2009

[16] G. Davies, "Equivalence Class Testing*"* Swansea University - 2007

[17] T. Murnane , K. Reed & R. Hall, *"On the learnability of two representations of equivalence partitioning and boundary value analysis."* In Proceedings of the 2007 *Australian Software Engineering Conference* (pp. 274-283). IEEE Computer Society- April- 2007.

[18] T. Murnane, & K. Reed. *"On the effectiveness of mutation analysis as a black box testing technique."* In *Software Engineering Conference,* 2001. Proceedings. 2001 Australian (pp. 12-20). IEEE - 2001

[19] C. Ferriday." A Review Paper on Decision Table Based Testing.*"* Swansea University-CS339 - 2007

[20] B. Neate." Boundary value analysis." University of Wales Swansea – 2006.

[21] M. Khan. "Different Approaches To Black Box Testing Technique For Finding Errors.*" International Journal of Software Engineering & Applications*, 2(4)- 2011

[22] K. Vij & W. Feng," Boundary value analysis using divide-and-rule approach*"*, in *Information Technology: New Generations. ITNG 2008.* Fifth International Conference, pp. 70-75, IEEE-April - 2008.