International Journal Of Engineering Research & Management Technology ISSN: 2348-4039

Email: editor@ijermt.org May- 2014 Volume 1, Issue 3 Website: ijermt.org

TEST CASES USING BOUNDARY VALUE ANALYSIS AND EQIVALENCE CLASS

Mamta Sharma
Research Scholar
Computer Science & Engineering
MD University Rohtak

Durgesh Srivastava
Assistant professor
Computer Science & Engineering
MD University Rohtak

Saurabh Jaiswal
Assistant professor
Computer Science & Engineering
NIT, Raipur

Abstract— The practice of testing software has become one of the most important aspects of the process of software creation. When we are testing software the first and potentially most crucial step is to design test cases. There are many methods associated with test case design. This Paper will document the two main approaches in testing known as Boundary Value analysis and Robustness Testing. "Testing can show the presence of bugs, but not the absence". Although this is true we find that testing can be very good at the first, if implemented correctly. For this reason we need to know of the techniques available so we can find the correct method for the system under test (SUT).

Keywords— Testing, White Box, Black Box, Boundary Value Analysis, Robustness, Equivalence.

I INTRODUCTION

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view about the Software to allow the business to appreciate and understand the risks of software implementation.[1] Software testing, depending on the testing method employed, can be implemented at any time in the software development process. A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. [5]

II TESTING METHODS

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

•**White-box testing** (also known as clear box testing, glass box testing, and transparent box testing and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases.[6][5]

•**Black-box testing** treats the software as a "black box", examining functionality without any knowledge of internal functionality. The testers are only aware of what the software is supposed to do, not how it does it Black-box testing methods include: equivalence partitioning, boundary value analysis, decision table testing, and state transition tables.

III TESTING PROBLEMS

Developing effective and efficient testing techniques has been a major problem when creating test cases; this has been the point of discussion for many years. There are several well known techniques associated with creating test cases for a system.

There are many issues that Can undetermined the integrity of the result from and given test suite (set of tests) implementation. These issues or questions can be as basic as where do we start? They can become more complicated when we try to ascertain where testing should end and if we have covered all the required permutations.

IV BOUNDARY VALUE ANALYSIS

Boundary value analysis is a test case design technique to test boundary value between both valid boundary partition and invalid boundary partition). A boundary value is an input or output value on the border of an equivalence partition, includes minimum and maximum values at inside and outside boundaries. Normally Boundary value analysis is part of stress and negative testing. [11]

Boundary Value Analysis focuses on the input variables of the function. For the purposes of this report I will define two variables (I will only define two so that further examples can be kept concise) X1 and X2. Where X1 lies between A and B and X2 lies between C and D.

$A \leq X1 \leq B$

$C \leq X2 \leq D$

The values of A, B, C and D are the extremities of the input domain.

Figure 1 Showing Boundary value analysis test cases for a function of two variables. The red dotted rectangular area is Input domain of given function.

As the name suggests Boundary Value Analysis focuses on the boundary of the input space to recognize test cases. The idea and motivation behind BVA is that errors tend to occur near the extremities of the input variables. The defects found on the boundaries of these input variables given can obviously be the result of countless possibilities.

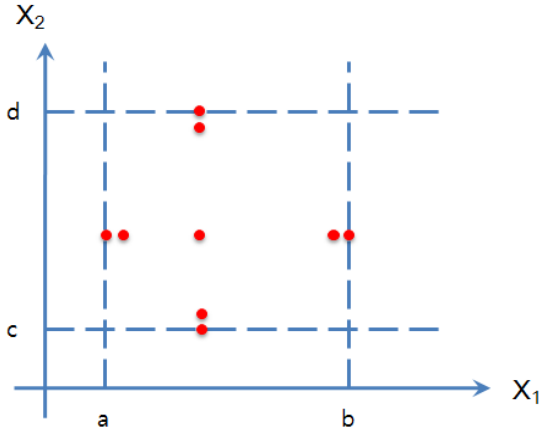

Fig. 1 Test case for Boundary Values

But there are many common faults that result in errors more collated towards the boundaries of input variables. For example if the programmer forgot to count from zero or they just miscalculated the values. Errors in the code concerning loop counters being off by one or the use of a < operator instead of ≤. These are all very common mistakes and accompanied with other common errors we find an increasing need to perform Boundary Value Analysis for test cases. Boundary Value Analysis can be done in a uniform manner. [5][15] The basic form of implementation is to maintain all but one of the variables at their nominal (normal or average) values and allowing the remaining variable to take on its extreme values. The values used to test the extremities are as follows:

- Min----------Minimal
- Min+----------Just above Minimal

- Nom----------Average

- Max----------Just below Maximum
- Max----------Maximum

In continuing our example this results in the following test cases shown in figures 2 and figure 3.

$< X_{1nom}^{1}\ X_{2min} > < X_{1nom}^{1}\ X_{2min+} >$
$< X_{1nom}^{1}\ X_{2nom} > < X_{1nom}^{1}\ X_{2max-} >$
$< X_{1min}^{1}\ X_{2nom} > < X_{1min+}\ X_{2nom} >$
$< X_{1max}^{1}\ X_{2nom} > < X_{1max}^{1}\ X_{2nom} >$

Fig. 2 Test Cases

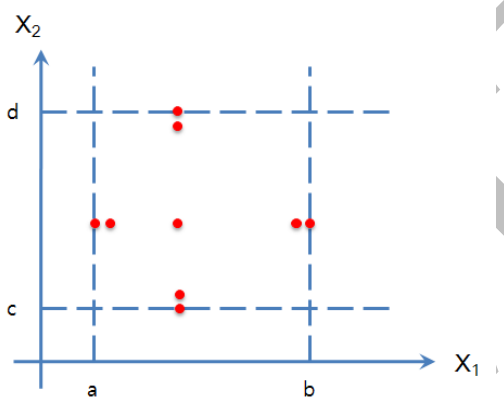

Fig.3 Function of Two Variables

Boundary Value Analysis Example Identifying shape by length, Breadth
Standard Boundary Value Test Cases showing in Table I:

Min = 1
Min+=2
Nom=100
Max-= 199
Max= 200

TABLE I
Boundary Value Test Cases

CASE	A	B	C	Result
1	100	100	1	Isosceles
2	100	1	2	Scalene
3	100	100	100	Equilateral
4	100	1	100	Isosceles
5	100	100	200	Not a Triangle

Boundary Value Analysis "if practised correctly, is one of the most useful test-case-design methods". But he goes on to say that it is often used ineffectively as the testers often see it as so simple they misuse it, or don't use it to its full potential. This is a very true interpretation of the use of Boundary Value Analysis. [3][1]

BVA can provide a relatively simple and formal testing technique that can be very powerful when used correctly.

ADVANTAGES:

1) Very good at exposing potential user interface/user input problems
2) Very clear guide lines on determining Test Cases.
3) Very small set of test cases generated.

DISADVANTAGES:

1) Does not test all possible inputs.
2) Does not test dependencies between combinations of inputs.

V EQUIVALENCE CLASS

Equivalence Partitioning refers to a testing technique, with two prime goals like:
1) Reduction of number of test cases to the bare minimum.
2) Identification of the ideal test cases, which should be able to cover maximum number of scenarios.
This testing Technique is typically applied to the inputs of a tested component, although in some isolated cases it can be used on the outputs of components in the software application. [18] Equivalence partitioning technique utilizes a subset of data which is representative of a larger class. Equivalence partitioning involves partitioning of the input domain of a software system into several equivalent classes in a manner that the testing of one particular representative from a class is equivalent to the testing of some different value from the subject class. Since it is practically not possible to carry out testing so exhaustively, the other better alternative is to verify to check as to whether the program treats particular input groups in a similar way or not. In case some group of similar values are present in the input domain, then such values can be treated as a single equivalent class & any single representative out of them can be tested. Equivalence Partitioning is carried out as a substitute of doing exhaustive testing for each value of data in a larger class. The methodology may be described with the help of following example.

TRIANGLE EXAMPLE: A Triangle has three input variables as its three sides a, b, c.
Output can be here Isosceles, Scalene Equilateral, Not a Triangle shown in Table II.

To identify output Equivalence classes are as under:

1 = Triangle is Equilateral
2 = Triangle is Isosceles
3 = Triangle is Scalene
4 = side a, b, c, do not form a Triangle

Four different Equivalence Classes are as follows:

1) Weak Normal Equivalence Class
2) Strong Normal Equivalence Class
3) Weak Robust Equivalence Class
4) Strong Robust Equivalence Class

1) Weak Normal Equivalence Class: The four weak normal equivalence class test cases can be defined as under:

TABLE II
Equivalence Class Test Cases

Test Case	Side 'a'	Side 'b'	Side 'c'	Output
1	5	5	5	Equilateral
2	2	2	3	Isosceles
3	3	4	5	Scalene
4	4	1	2	Not a Triangle

2) Strong Normal Equivalence Class: Since no valid subintervals of variables a, b and c exist, so the strong normal equivalence class test cases are identical to the weak normal equivalence class test cases.

3) Weak Robust Equivalence Class: Considering the invalid values for a, b and c yields the following additional weak robust equivalence class test cases shown in Table III.

TABLE III
Weak Robust Equivalence Class Test Cases

Test Case	Side 'a'	Side 'b'	Side 'c'	Output
WR1	1	5	5	Invalid value of 'a'
WR2	5	-1	5	Invalid value of 'b'
WR3	5	5	-1	Invalid value of 'c'
WR4	200	5	5	Out of range 'a'
WR5	5	200	5	Out of Range 'b'
WR6	5	5	200	Out of Range 'c'

4) Strong Robust Equivalence Class: Test Cases falling under this category are shown in Table IV.

TABLE IV
Strong Robust Equivalence Class Test Cases

Test Case	Side 'a'	Side 'b'	Side 'c'	Output
SR1	-1	5	5	Invalid value of 'a'
SR2	5	-1	5	Invalid value of 'b'
SR3	5	5	-1	Invalid value of 'c'
SR4	-1	-1	5	Invalid value of 'a','b'
SR5	5	-1	-1	Invalid Value of 'b','c'
SR6	-1	5	-1	Invalid Value of 'a','c'
SR7	-1	-1	-1	Invalid Value of a,b,c

ADVANTAGES:

1) Equivalence partitions are designed so that every possible input belongs to one and only one equivalence partition.

DISADVANTAGES:

1) Doesn't test every input.
2) No guide lines for choosing inputs.

VI CONCLUSION

We find that Boundary Value Analysis where boundaries of the Input Domain are used to generate test cases to ensure proper functionality of System. It is best way to catch common user input errors which can disrupt program functionality. Where as in Equivalence Class Test input are selected from each partition. This Testing minimize the number of test cases by dividing tests in such a way that the system is expected to act the same way for all tests of each equivalence portion. Test inputs would be selected from each partition. The underlying fact is that generally Boundary Value Testing techniques are computationally and theoretically inexpensive in the creation of test cases. For this reason in many cases it can be desirable in its results to effort ratio. This means that Boundary Value Analysis still has a part to play in modern day testing practises.

REFERENCES

1. Pressman, Roger S. software Engineering. New Delhi: Prentice Hall of India.
2. Ghezzi, Carlo. Fundamentals of Software Engineering. New Delhi Prentice Hall of India.
3. Mali, Rajib, Fundamental of Software Engineering. New Delhi Prentice Hall of India.
4. Summerville, Ian Software Engineering New Delhi: addision Wesley.
5. B. Beizer, Black Box Testing. New York: John Wiley & Sons, Inc., 1995.
6. M. Dowson. The Ariane 5 software failure. SIGSOFT Software Engg. Notes, 22(2):84, 1997.
7. IEEE. Standard glossary of software engineering terminology. IEEE Std 610.12-1990, 10 Dec 1990.
8. Boehm, Barry & Basili, Victor R. "Software Defect Reduction Top 10 List." IEEE Computer 34, 1 (January 1991): 135-137.
9. IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.
10. Mohd. Sadiq, Mohd. Shahid, "Elicitation and Prioritization of Software Requirements", International Journal of Recent Trends in Engineering, Finland.
11. B. Beizer, *Software Testing Techniques*. London: International Thompso Computer Press, 1990.
12. A. Bertolino, "Chapter 5: Software Testing," in *IEEE SWEBOK Trial Version1.00*, May 2001.
13. B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
14. L. Copeland, *A Practitioner's Guide to Software Test Design*. Boston: Artech House Publishers, 2004.
15. R. D. Craig and S. P. Jaskiel, *Systematic Software Testing*. Norwood, MA: Artec House Publishers, 2002.
16. E. W. Dijkstra, "Notes on Structured Programming," Technological University Eindhoven T.H. Report 70-WSK-03, Second edition, April 1970.
17. D. Galin, *Software Quality Assurance*. Harlow, England: Pearson, Addison Wesley, 2004.
18. IEEE, "ANSI/IEEE Standard 1008-1987, IEEE Standard for Software Unit Testing," no., 1986.
19. IEEE, "ANSI/IEEE Standard 1008-1987, IEEE Standard for Software Unit Testing," no., 1987.
20. P. Jorgenson, Software Testing- A Craftsman's Approach, CRC Press, New York, 1995 .
21. Naryan C Debnath, Mark Burgin, Haesun K. Lee, Eric Thiemann, A Testing and analysis tool for Certain 3-Variable functions, Winona State University.
22. Glenford J. Myers, The Art of Software Testing, John Wiley and Sons, Inc. 2004.