International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Tool for Generation of Test Cases in Black Box Testing

Mamta Sharma[1], Durgesh Srivastava[2]

Computer Science Assistant Professor
Maharishi Dayanad University Rohtak Maharishi Dayanad University Rohtak

Abstract: ***Software Testing is one of the major area in software development. Each phase of software development life cycle includes the software testing as its major part. As the software development begin the role of software testing also begin. In this Research we have developed a tool to generate different test cases automatically. To show the validity of the tool, we have considered the line equation problem and generated different test cases, and finally we conclude that Robustness Technique is better than Boundary Value Analysis.***

Keyword: Black Box Testing, Boundary Value Analysis , Robustness Testing ,Black Box Tool ,White Box Testing,.

1. Introduction

Testing is the major quality control measure used during software development. Its basic function is to detect errors in the software. During requirement analysis and design. The output is a document that is usually textual and non-executable. After the coding phase, computer programs are available that can be executed for testing purposes. Thus the goal of testing is to uncover requirement, design and coding errors in the programs. Consequently, different levels of testing are used. The starting point of testing is unit testing [12]. In this, a module is tested separately and is often performed by the coder himself simultaneously along with the coding of the module. The purpose is to exercise the different parts of the module code to detect coding errors. After this, the modules are gradually integrated into subsystem, which are then integrated to eventually form the entire system. During integration of modules, integration testing is performed to detect design errors by focusing on testing the interconnection between modules.

2. Testing

It finds the errors in software design [3]. During software testing we find bugs and errors in the software and remove them. We can define the testing as:

(i) The process of exercising software to verify that it satisfies specified requirements.
(ii) The process of analyzing a software item to detect the differences between existing and required conditions (that is,bugs), and to evaluate the features of the software item.
(iii) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.

The paper is organized as follows: Section 3 explains different types of testing i.e., black box and white box testing Experimental work is carried out in section 4. Snapshots of the output are given in section 5. We have tabulated the results of the proposed tool in section 6, and finally we conclude the paper in section 7.

3. Types of Testing

There are 2 types of testing: [1, 2, 3]
A) Black box testing
B) White box testing

A) Black Box
This type of testing involves testing the software for functionality, it is used to find out the errors in data structure, faulty functions, interface errors, etc. Black box testing ignores internal mechanism of a system [1]. It Identifies bugs only according to software malfunctions as they are revealed in its erroneous outputs. It is used to find incorrect functions that led to undesired output when executed, incorrect conditions due to which the functions produce incorrect outputs, when they are executed [2]. Following techniques are used to test a program using black box testing techniques [1,3, 4].
1. Boundary Value Analysis (BVA)
2. Robustness
3. Worst case
4. Equivalence Partitioning
5. Decision Table
Black Box Testing allows us to carry out the majority of testing classes, most of which can be implemented solely by black box tests. Black box testing requires fewer resources. Brief description about white box testing is given in the next sub-section. In this paper we will discuss only two techniques i.e., Boundary Value Analysis and Robustness Technique.

B) White Box Testing
White-box testing takes into account the internal mechanism of a system or component. White-box testing is also known as structural testing, clear box testing, and glass box testing. It involves testing of all logic of program, testing of loops, condition testing and data flow based testing. This helps in detecting errors even with unclear and incomplete software specification. The objective of white box testing is to ensure that the test cases exercise each path through a program [1]. The test cases also ensure that all independent paths within the program have been executed at least once. All internal data structures are exercised to ensure validity. All loops are executed at their boundaries and within operational bounds. Each branch is exercised at least once.

4. Experimental Work

In this paper we have developed a tool and implemented in PHP to generate the test cases automatically. In order to validate the functionality of the tool we have considered a straight line problem, with (m1, c1) and (m2, c2) defining the lines of the form y= m x + c with following conditions;

(i) Parallel lines (m1=m2, c1 != c2
(ii) Intersecting lines (m1 != m2)
(iii) Coincident lines (m1=m2, c1=c2)

(A) Boundary Value Analysis

In black box testing, test cases are derived on the basis of values that lie on the edge of the equivalence partition [5].. It is found that most of the errors occur at the boundary rather then the middle of the domain. If an input consists of certain values, then test cases should be able to exercise both the values at the boundaries of the range and the values that are just above and below the boundary values. For example for the range [10,100] the input values for a test would be 10,11,55,99,100 Boundary Value Analysis focuses on the input variables of the function. Let us consider two variables Y1 and Y2, where Y1 lies between A and B and Y2 lie between C and D.

A < = Y1 < = B (1)
C < = Y2 < = D (2)

We have summarized the total 4n+1 number of test cases and the expected output in the table-I

Table 1: Test cases using Boundary Value Analysis

TEST CASE	*M1*	*C1*	*M2*	*C2*	*RESULT*
1	10	55	55	55	Intersecting Lines
2	11	55	55	55	Intersecting Lines
3	55	55	55	55	Coincident Lines
4	99	55	55	55	Intersecting Lines
5	100	55	55	55	Intersecting Lines
6	55	10	55	55	Parallel Lines
7	55	11	55	55	Parallel Lines
8	55	99	55	55	Parallel Lines
9	55	100	55	55	Parallel Lines
10	55	55	10	55	Intersecting Lines
11	55	55	11	55	Intersecting Lines
12	55	55	99	55	Intersecting Lines
13	55	55	100	55	Intersecting Lines
14	55	55	55	10	Parallel Lines
15	55	55	55	11	Parallel Lines
16	55	55	55	99	Parallel Lines
17	55	55	55	100	Parallel Lines

(B) Robustness Testing

A component is robust when it never fails or crashes, whatever the input is. Indeed, the failure of a single component may cause the failure of the entire system. IEEE defines robustness as the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [7]. The value of detecting errors early in the development cycle, both in terms of cost and schedule, is well known. Boehm and Basili reported “Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.” This statement is as true for robustness problems as it is for functionality defects. If we adapt our function *f* to apply to Robustness testing we find the following equation:

$f = 6n + 1$

We have summarized the total number of test cases and the expected output in the table-02

Table 2: Test cases using Robustness Testing

TEST CASE	*M1*	*C1*	*M2*	*C2*	*Result*
1	9	55	55	55	Intersecting Lines
2	10	55	55	55	Intersecting Lines
3	11	55	55	55	Intersecting Lines
4	55	55	55	55	Coincident Lines
5	99	55	55	55	Intersecting Lines
6	100	55	55	55	Intersecting Lines
7	101	55	55	55	Intersecting Lines
8	55	9	55	55	Parallel Lines
9	55	10	55	55	Parallel Lines
10	55	11	55	55	Parallel Lines
11	55	99	55	55	Parallel Lines
12	55	100	55	55	Parallel Lines
13	55	101	55	55	Parallel Lines
14	55	55	9	55	Intersecting Lines
15	55	55	10	55	Intersecting Lines
16	55	55	11	55	Intersecting Lines
17	55	55	99	55	Intersecting Lines
18	55	55	100	55	Intersecting Lines
19	55	55	101	55	Intersecting Lines
20	55	55	55	9	Parallel Lines
21	55	55	55	10	Parallel Lines
22	55	55	55	11	Parallel Lines
23	55	55	55	99	Parallel Lines
24	55	55	55	100	Parallel Lines
25	55	55	55	101	Parallel Lines

Following program shows the implementation of one of the module of the proposed tool.

Script in PHP

```
<Style>
.five {width: 125px; padding: 1px; display: inline-block ;}
.boundary form {text-align: centre ;}
Input {
 margin: 10px;
 padding: 5px;
 width: 250px;
}
</style>
<form method='post' action='' class="boundaryform">
<input type="text" value="" placeholder="First Value"
name="fboundary" required><br>
<input type="text" value="" placeholder="Second Value"
name="sboundary" required><br>
Enter 1 for Boundary Value Analysis<br>
Enter 2 for Robustness Testing<br>
Enter 3 for Exit<br>
<input type="text" value="" placeholder="Enter your
Choice" name="choice" required><br>
<input type="submit" value="enter">
</form>
<div class="boundaryform">
<?php
$a=array();
$s='';
```

```
$e='';
$count=0;
$mid='';
$steps='';
$ch='';
$ans='';
if (@$_POST['fboundary'] !='' && @$_POST['sboundary']
!= '' && @$_POST['choice'] != ''){
$s=$_POST['fboundary'];
$e=$_POST['sboundary'];
$ch=$_POST['choice'];
if($ch==1)
{
$mid=2;
$steps=5;
$a[0]=$s;
$a[1]=$s+1;
$a[2]=($s+$e)/2;
$a[3]=$e-1;
$a[4]=$e;
}
else if($ch==2)
{
$mid=3;
$steps=7;
$a[0]=$s-1;
$a[1]=$s;
$a[2]=$s+1;
$a[3]=($s+$e)/2;
$a[4]=$e-1;
$a[5]=$e;
$a[6]=$e+1;
}
```

5. Output of Proposed Tool

In this section we have shown the various snapshots of the output of the proposed tool under different situations. We have delineated the comparison between BVA and RT is shown.

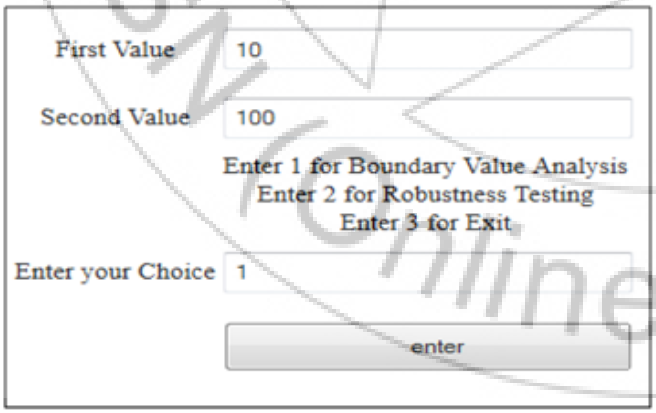

Figure 1: Form for entering Values

TestCase	m1	c1	m2	c2	Result
1	10	55	55	55	Intersecting lines
2	11	55	55	55	Intersecting lines
3	55	55	55	55	Coincident lines
4	99	55	55	55	Intersecting lines
5	100	55	55	55	Intersecting lines
6	55	10	55	55	Parallel Lines
7	55	11	55	55	Parallel Lines
8	55	99	55	55	Parallel Lines
9	55	100	55	55	Parallel Lines
10	55	55	10	55	Intersecting lines
11	55	55	11	55	Intersecting lines
12	55	55	99	55	Intersecting lines
13	55	55	100	55	Intersecting lines
14	55	55	55	10	Parallel Lines
15	55	55	55	11	Parallel Lines
16	55	55	55	99	Parallel Lines
17	55	55	55	100	Parallel Lines

Figure 2: Screen shot showing Test Cases Generated

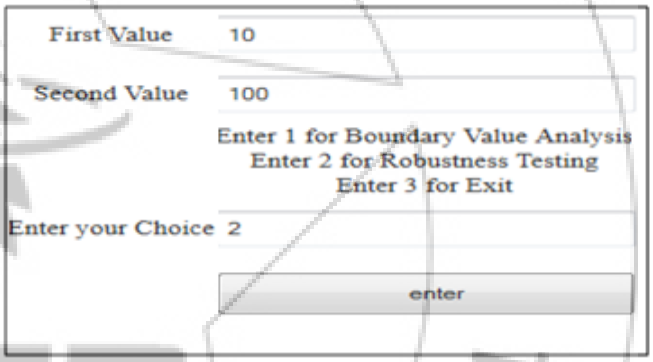

Figure 3: Screen Shot Showing form for second choice

TestCase	m1	c1	m2	c2	Result
1	9	55	55	55	Intersecting lines
2	10	55	55	55	Intersecting lines
3	11	55	55	55	Intersecting lines
4	55	55	55	55	Coincident lines
5	99	55	55	55	Intersecting lines
6	100	55	55	55	Intersecting lines
7	101	55	55	55	Intersecting lines
8	55	9	55	55	Parallel Lines
9	55	10	55	55	Parallel Lines
10	55	11	55	55	Parallel Lines
11	55	99	55	55	Parallel Lines
12	55	100	55	55	Parallel Lines
13	55	101	55	55	Parallel Lines
14	55	55	9	55	Intersecting lines
15	55	55	10	55	Intersecting lines
16	55	55	11	55	Intersecting lines
17	55	55	99	55	Intersecting lines
18	55	55	100	55	Intersecting lines
19	55	55	101	55	Intersecting lines
20	55	55	55	9	Parallel Lines
21	55	55	55	10	Parallel Lines
22	55	55	55	11	Parallel Lines
23	55	55	55	99	Parallel Lines
24	55	55	55	100	Parallel Lines
25	55	55	55	101	Parallel Lines

Figure 4: Screen Shot showing test cases generated for second choice

6. Summarized Test Cases

In table 3 showing total number of test cases generated.

Table 3: Test Cases Summarized

S. No	*Type of Test*	*No. of Test Cases*
1	Boundary Value Analysis	17
2	Robustness Testing	25

In Table 4 showing total number of Intersecting, Coincident & Parallel Lines in Boundary Value Testing and Robustness Testing.

Table 4: Comparing Results

Type of Test	*Intersecting*	*Parallel*	*Coincident*
Boundary Value Analysis	8	8	1
Robustness Testing	12	12	1

7. Conclusion & Future Scope

During the analysis of black box testing we conclude that the total number of test cases in case of BVA and Robustness are 17 and 25 respectively, and finally we have shown that the robustness technique is better than Boundary Value Analysis we can develop it and can use it for other test case Designing techniques also. Like Decision Table, Equivalence Class Testing. We can develop it and can use it for other test case Designing techniques also. Like Decision Table, Equivalence Class Testing.

References

[1] Collofello, J., AZ, Vehathiri, K. "An environment for training computer science students on software testing", Dept. of Computer. Sci. & Eng., Arizona State University pp 12-18, 2005.

[2] Xia Cai., Lyu, M.R., "Software Reliability Modeling with Test Coverage: Experimentation & Measurement with a Fault Tolerant Software Project" .pp 17-26, Chinese Univ. of Hong Kong, Hong Kong, 2007.

[3] Itkonen, J.; Software Bus. & Eng. Inst., Helsinki Univ. of Technol., Helsinki, Finland; Mäntylä, M.V., Lassenius, C. "How do Testers Do It? An Explarotory study on manual Testing Practices", Empirical Software Engineering and Measurement, 2009,Page:494-497,ISSN: 1938-6451,Oct 2009, Florida.

[4] Sandra Kukolj, Vladimir Marinkovi, Miroslav Popovi, "Selection and Prioritization of Test Cases By Combining White Box and Black Box Testing Methods, 3rd Eastern European Regional Conference, page 153-156, INSPEC: 13900367 Aug 2013, Budapest.L. Chung. B. Nixon, E. Yu, J. Mylopoulos, "Non-Functional Requirements in Software Engineering", Kluwer, 1999.

[5] Xia Cai, Lyu, M.R. "Software Reliability Modelling With Test Coverage Experimentation & Measurement with a Fault Tolerant Software Project" The IEEE 18th International Symposium, page: 17-26, 5-9 Nov 2007, ISSN: 1071-9458, , Hongkong.

[6] A. Avezienis, J. Laprie and B. Randell, "Fundamental Concepts of Computer System Dependability", IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments, Seoul, Korea, May 21-22, 2001.

[7] Yi-Tin Hu , Nai-Wei Lin, " Automatic black box method level test case generation based on constraint logic programming", Computer symposium,2010 International 16-18 Dec. 2010 , ISBN: 978-1-4244-7639-8, page 977-982, , Germany.

[8] Mohan, K.K. ; Reliability Eng., Indian Inst. of Technol. Bombay, Mumbai, India ; Verma, A.K. ; Srividya, A., "Software reliability estimation through black box and white box testing at prototype level", Safety & Hazard 2nd International Conference, 14-16 Dec 2010, Page:517-522,INSPEC:12030219 , Mumbai.

[9] Accioly, P. Borba, Bonifacio R., "Comparing Two Black-Box Testing Strategies for Software Product Lines", 23-28 Sept 2012, page 1-10, INSPEC: 13251932, 6th Symposium on Software Components Architecture Reuse, , Natal.

[10] Tilley, S., Praveen, T., "Migrating Software Testing to Cloud", Software Maintenance (ICSM), IEEE International Conference, 12-18 Sept. 2010, ISBN: 978-1-4244-8628-1, Page: 1-3, Timisoara.

[11] Liu Xue-mei, Gu Guochang, liu Yong-Po; Wu Ji, "Research and Implementation of Knowledge Management Methods in Software Testing Process ", Computer Science and Information Engineering, 2009 WRI World Congress on (Volume:7), March 31 2009-April 2 2009, ISBN: 978-0-7695-3507-4, page 739-749, CA.