BBTool: A Tool to Generate the Test Cases

Mumtaz Ahmad Khan, Preeti Bhatia, Mohd. Sadiq

Abstract:- Software testing (ST) is an important phase of a software development life cycle (SDLC). During ST, software are verified and validated by the software testers to check whether it meets the stakeholder's expectations or not. It is time consuming process to check each and every condition of the software during ST, if we check it manually. Therefore, to reduce the time of software testing, an effort has been made to reduce the time of testing. In this paper, we have proposed BBTool, i.e. Black Box Tool, to generate the test cases. Straight line problem is employed to show the validity of BBTool.

Keywords:- Software, Software Testing, Black Box Testing, Boundary Value Analysis, and Robustness Technique

I. INTRODUCTION

This paper is an extension of our previous work [15], in which we have presented a case study for the analysis of black box software testing. This paper contains the architecture and implementation [15] of the BBTool, i.e. a tool to generate the test cases, as shown in figure 1. In the software development process, software testing is used to provide a platform to the stakeholders about the quality of the product or services under test. It provides an objective to understand the risks of software implementations. Software testing can be defined as the process of validating and verifying that a software programs/application/ product meets the requirements which satisfy the needs of customers. 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 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. Robustness3. Worst case
4. Equivalence Partitioning
5. Decision Table

Manuscript Received on May 25, 2012.

Mumtaz Ahmad Khan is with Section of Electrical Engineering, University Polytechnic, Faculty of Engineering and Technology, Jamia Millia Islamia, A Central University, New Delhi-110025, India.

Preeti Bhatia is M. Tech. Scholar, Computer Science and Engineering, AL-Falah School of Engineering and Technology, Rohtak, Haryana, India.

Mohd. Sadiq is with Department of Computer Engineering, National Institute of Technology, Kurukshetra-136119, Haryana, India.

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.

1. 1 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.

Using white-box testing a software engineer can design test cases that (1) exercise independent paths within a module or unit; (2) exercise logical decisions on both their true and false side; (3) execute loops at their boundaries and within their operational bounds; and (4) exercise internal data structures to ensure their validity [1].White box testing covers the larger part of the program code while testing, and it uncovers typographical errors. Test cases need to be changed, if implementation changes. In this paper we have used only black box testing approach to check the functionality of two different lines based upon the different test cases using BVA and Robustness Testing.

A strong link between testing and requirements can benefit both disciplines but this link is missing because both the field has different mindset of people. A survey was conducted by the Eero J. Uusitalo et al [13] to link between requirements and testing. There is several benefit of the early tester's participation during requirements analysis, like testing activities are properly taken into account in planning, i.e. budgeted for both in scheduled and resources, testers domain and system knowledge are improved because of additional exposure of the subject, improved test coverage, as it is more apparent which requirements the test case cover, efficient of change management is improved because requirement change can be traced to appropriate test case.

Readers are advised please refer to [12] for more explanation about the linking of testing with requirements.

Linking Testing with requirements could be useful to manage the uncertainty because it provides the open communication between tester's and requirements writers during the development and testing process. From this approach we can reduce the assumption made by the testers and there is a chance of increased reliability of test results and subsequent products. In the next section we have consider an example to test a different types of line and with the help of this we have shown that how testing could be used to generate the test cases of a program and how these test cases could be used to make a list of requirements that we cannot get through only the elicitation techniques. A software project is nothing but a combination of program, documentation and operating procedure. That is why we have considered a small program to validate our results.

The paper is organized as follows: Section 2 explains steps involved in SDLC. Architecture of BBTool and Experimental work is carried out in section 3. Snapshots of the output are given in section 4. We have tabulated the results of the proposed tool in section 5, and finally we conclude the paper in section 6.

II. SOFTWARE DEVELOPMENT LIFE CYCLE

In the literature of software development there are basically four steps that are used in the software industry. These steps are requirement analysis, object design, coding and testing. A brief discussion about the different phases of SDLC is given in the following sub-section [15].

2.1. System Requirements Analysis

Requirements engineering (RE) is a processes which determine the requirements of the customers, so that software can be developed according to the needs of customers and users. There are five sub-processes in RE namely, requirements elicitation, requirements modeling, requirements analysis, requirements verification and validations, and requirements management [13, 14]. In this phase software engineer deals with finding out what the problem is. It is required to find out the correct problem or flaws of the existing system. This phase takes as the input goal identified by the requirement section of the project planning. Feasibility study is required to find out if a system development project can be done. One of the most important points in this phase is the software requirement elicitation. In requirement elicitation, an analyst collects requirements and prepares a plan for software development. The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Requirement engineering is the process of discovering the need of the stakeholder and documenting these needs in a form that is amenable to analysis [10, 11, 12]

2.2 Object Design

Once we have finalized the requirements, the next phase is the object design. A design is a pattern which is a reusable solution to commonly occurring problems within a given context in software design. It is a description or template that can be used in many different situations like, how to solve a problem. Objected oriented design patterns typically show relationship and interactions between classes and objects. Requirements engineers generally prefer the UML notations for the objects design. There are many types of patterns like, algorithm strategy patterns i.e., addressing concerns related to high-level strategies describing how to exploit the application characteristics on a computing platform, computational design patterns i.e., related to key computation identification, and execution patterns i.e., relating to supporting execution, implementing strategy patterns i.e., relating to implementation source code. The major steps in this phase include output design, input design, file design, processing design and control design etc.

2.3 Code Generation

This is the phase where actual development of the system takes place. The blue prints are translated into software (i.e. Documentation+ Programs+ Operating Procedure). The entire project is broken into modules and each module is assigned to separate teams according to expertise of the programmers in the respective teams.

2.4. Testing

It finds the errors in software design [3,15]. 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.

III. III. ARCHITECTURE AND EXPERIMENTAL WORK

This paper presents the architecture of BBTool which is an extension of our previous work [15]. In this paper, we have considered a very simple example to validate the usefulness of our tool. 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)

As we know that Software testing is an important process to find out as many errors as possible in the software.

In SDLC, testing and requirements engineering are an important activity, which are employed to test and identify the requirements respectively. The main objective of the requirements engineer is to find out the requirements, which satisfy the needs of the customers. In this paper we have developed a tool to generate the test cases using Boundary Value Analysis and Robustness Testing on the two straight lines equation problem. The architecture of the proposed tool is given in the following figure.

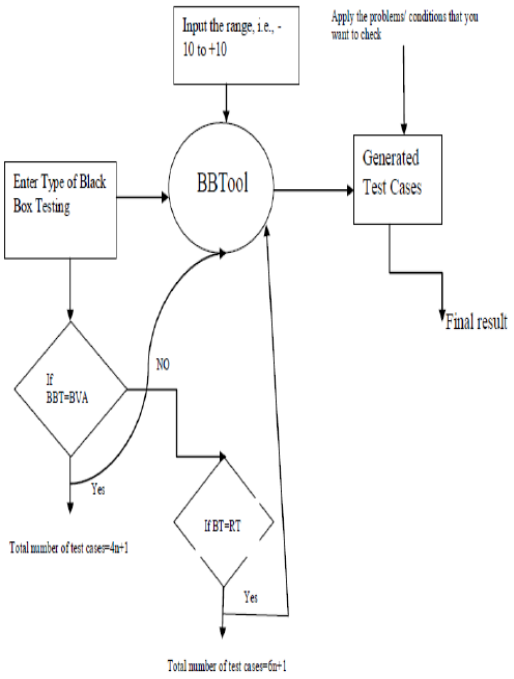

Fig.-1

3.1 Boundary Value Analysis (BVA)

In black box testing, test cases are derived on the basis of values that lie on the edge of the equivalence partition [5]. The values can be either at the edge or within the permissible range from the edge of the equivalence partition. It is found that most of the errors occur at the boundary rather than 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.

Table-01
Test cases using Boundary Value Analysis

Test Cases	M1	C1	M2	C2	Expected Output
1	0	50	50	50	Intersecting lines
2	1	50	50	50	Intersecting lines
3	50	50	50	50	Coincident lines
4	99	50	50	50	Intersecting lines
5	100	50	50	50	Intersecting lines
6	50	0	50	50	parallel lines
7	50	1	50	50	parallel lines
8	50	99	50	50	parallel lines
9	50	100	50	50	parallel lines
10	50	50	0	50	Intersecting lines
11	50	50	1	50	Intersecting lines
12	50	50	99	50	Intersecting lines
13	50	50	100	50	Intersecting lines
14	50	50	50	0	parallel lines
15	50	50	50	1	parallel lines
16	50	50	50	99	parallel lines
17	50	50	50	100	parallel lines

In BVA the total number of test cases would be 4n+1. Here the value of n=4, so the total number of test cases is 17.

3.2 Robustness Testing (RT)

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, as it happened with the Ariane fight 501, where arithmetic overflow of a simple piece of software calibrated for Ariane 4 led to the change of direction of the launcher, and therefore to its crash [6]. 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. We have tested the lines using RT and summarized the total number of test cases and the expected output in the table-02.

Table-02
Test cases using Robustness Testing

Test Cases	M1	C1	M2	C2	Expected Output
1	-1	50	50	50	Intersecting lines
2	0	50	50	50	Intersecting lines
3	1	50	50	50	Intersecting lines
4	50	50	50	50	Coincident lines
5	99	50	50	50	Intersecting lines
6	100	50	50	50	Intersecting lines
7	101	50	50	50	Intersecting lines
8	50	-1	50	50	parallel lines
9	50	0	50	50	parallel lines
10	50	1	50	50	parallel lines
11	50	99	50	50	parallel lines

12	50	100	50	50	parallel lines
13	50	101	50	50	parallel lines
14	50	50	-1	50	Intersecting lines
15	50	50	0	50	Intersecting lines
16	50	50	1	50	Intersecting lines
17	50	50	99	50	Intersecting lines
18	50	50	100	50	Intersecting lines
19	50	50	101	50	Intersecting lines
20	50	50	50	-1	parallel lines
21	50	50	50	0	parallel lines
22	50	50	50	1	parallel lines
23	50	50	50	99	parallel lines
24	50	50	50	100	parallel lines
25	50	50	50	101	parallel lines

In this section we have shown the various snapshots of the output of the proposed tool under different situations in figure 2 and figure 3. We have delineated the comparison between BVA and RT in figure 4.

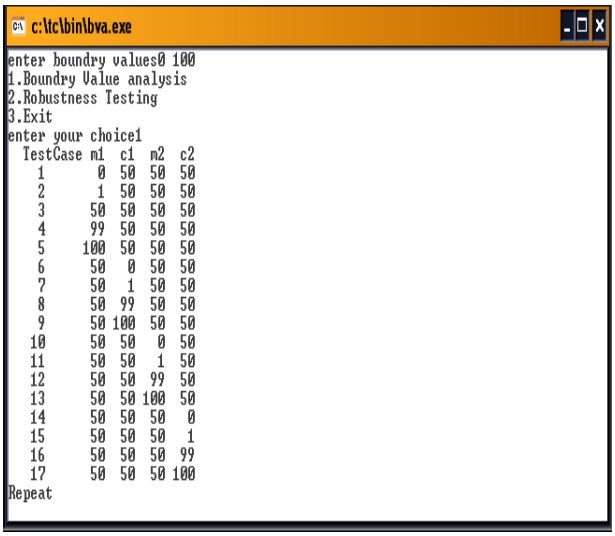

Figure-2

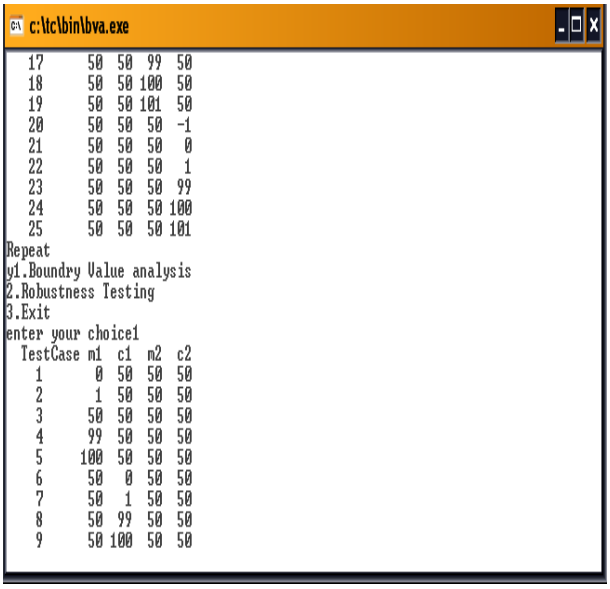

Figure-3

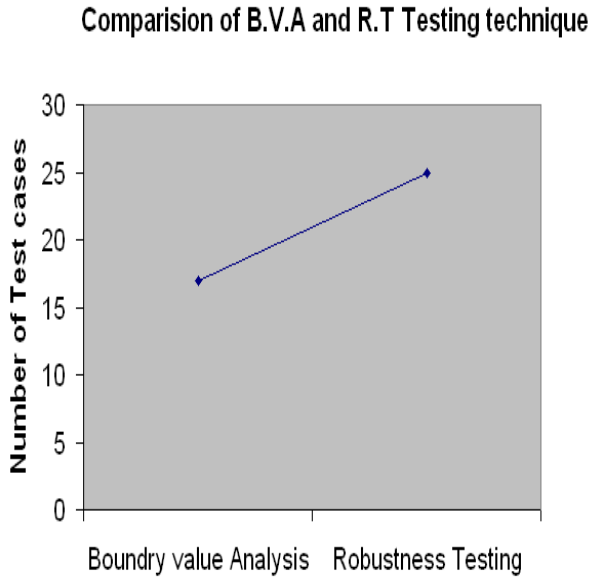

Figure-4

IV. RESULTS

We have summarized the results of BBTool in the following table, i.e., table 03. 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 because number of times the expected output in case of robustness techniques for intersecting lines, parallel lines and coincident lines are 12,12 and 1 respectively while in case of Boundary Value Analysis it is 8,8 and 1 respectively. Automatic generation of test cases using BBTool reduce the time of the testers. In this architecture, we have considered two different testing techniques like boundary value

analysis (BVA) and robustness techniques (RT). In BBTool, the user will enter the type of black box testing and the range of the input. Depending on the type of technique and the value of the input, BBTool will generate the total number of test cases and then these test cases will be used on the problems, that we want to test. We hope that, automation of the black box testing; will definitely reduce the time to check the condition in a program or a problem.

Table-03

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

Type of Line	Boundary Value Analysis	Robustness Testing
Intersecting lines	8	12
parallel lines	8	12
Coincident lines	1	1

V. CONCLUSION AND FUTURE WORK

This paper presents the architecture of the BBTool. One of the limitations of this tool is that it supports only two techniques of black box testing and we can't apply it in all general cases. In future paper: (i) we will expand to this work and will include the remaining black box testing (BBT) techniques (ii) and will present the complete tool using BBT techniques so that it can be used in all general cases.

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]. Mohd. Sadiq, Mohd. Shahid, Shabbir Ahmed, "Adding Threat during Software Requirements Elicitation and Prioritization", International Journal of Computer Application.
[12]. Mohd. Sadiq, Javed Ahmad, Abdul Rahman, R. Suman and Shweta Khandelwal, "More on Adding Threat during Software Requirements Elicitation and Prioritization", IACSIT International Journal of Engineering and Technology, pp.286-290, Vol.2, No.3, June 2010, ISSN: 1793-8236.
[13]. Eero Uusitalo, Marko Komssi, Marko Kauppinen, Alan M. Davis, " Linking Requirements and Testing in Practice", 16th IEEE International Requirements Engineering Conference, 2008, pp.265-270
[14]. Ian Sommervilla, "Integrated Requirements Engineering: A Tutorial", IEEE Software, 2005.
[15]. Mumtaz Ahmed Khan, Mohd. Sadiq, "Analysis of Black Box Software Testing Techniques: A Case Study", IEEE International Conference and Workshop on Current Trends in Information Technology, pp.1-5, December, 2011, Dubai, UAE.

AUTHOR PROFILE

Dr. Mumtaz Ahmad Khan has received his B. Tech.(Electrical Engineering) from Jamia Millia Islamia, New Delhi, M. Tech. (Electrical Engineering) from Aligarh Muslim University, Aligarh and Ph.D (Electrical Engineering) from Jamia Millia Islamia, New Delhi in 1992, 1995 and 2005 respectively from India. Before joining to the academics in 1998, he served in the industry in India and in KSA around 4 ½ Years. He is involved in teaching and research since then.

Dr Khan had served BITS, Pilani-Dubai for Five years, from 2005 to 2010 as Assistant Professor, EEE and currently he is the Professor/Head of Electrical Engineering in Electrical Engineering Section, Faculty of Engg. & Technology, JMI Central University, Govt. of India. New Delhi, India.

Dr Khan is having many publications to his credit in the national and international Journals and conferences. He has supervised one Ph.D. in BITS-Pilani, Dubai on Intelligent Buildings and three are under supervision in India. His areas of research are Intelligent applications in Electrical & Computer Engg, Intelligent and Energy Efficient Buildings, Wireless communications etc. He is an active member of IEEE and other national & International bodies.

Preeti Bhatia is pursuing M.Tech.in Computer Science and Engineering from AL-Falah School of Engineering and Technology, Faridabad, affiliated to Maharshi Dayanand University- Rohtak, Haryana. She did B. Tech. in Computer Science and Engineering in 2009 from Institute of Technology and Management, Faridabad, affiliated to Maharishi Dayanand University- Rohtak, Haryana. She has more than 2.5 years of teaching experience. Her area of interest includes Software Engineering, Theory of Automate, and Object Oriented Languages.

Mohd. Sadiq is working as Assistant Professor, Computer Engineering, in Section of Computer Engineering, University Polytechnic, Faculty of Engineering and Technology, Jamia Millia Islamia (A Central University), New Delhi, India and he has more than 7 years of teaching experience. Currently he is pursuing Ph.D. in Computer Engineering from National Institute of Technology, Kurukshetra, Haryana, India.

He did Master of Technology in Computer Science and Engineering with specialization in Software Engineering from Aligarh Muslim University (AMU), Aligarh, U.P., India, in 2005 and Bachelor of Engineering in

Computer Science and Engineering from Al-Falah School of Engineering and Technology, Dhauj, Faridabad, Haryana, affiliated to Maharishi Dayanand University, Rohtak, Haryana, in 2001with first position in second year.

He has published more than 30 research papers in international and national journals like IACSIT International Journal of Engineering and Technology, International Journal of Recent Trends in Engineering, Finland and International Journal of Futuristic Computer Application, Duke University, USA; and in international and national conferences (like IEEE , Springer) International conferences at Singapore, Thailand, Chennai, Kerala and Bangalore (India). His research interest includes Software Engineering, Requirements Engineering, Data Structure and Algorithms.

Mr. Sadiq is a member of International Association of Engineers (IAENG) England, International Association of Computer Science and Information Technology (IACSIT), Singapore and member of Computer Science Teachers Association (CSTA), USA.

Mr. Sadiq is a member of Editorial Review Board of Journal for Computing Teachers (JCT), Buffalo State College, New York, U.S.A., and also the Member of the Editorial Review Board of Journal of Internet and Information System, Victoria Island Lagos, Nigeria. He is the Reviewer of the International Journal of Computer Science and Technology and International Journal of Electronics and Communication Technology. He is also the Regional Coordinator of International Journal of Emerging Technologies and Applications in Engineering Technology and Sciences and International journal of Computer Applications in Engineering, Technology and Sciences.

Mr. Sadiq has chaired the session of IEEE International Conference on Advances in Recent Technologies in Communication and Computing, 2009, Kerala, India. He has also served as the member of Technical Program Committee and Reviewer of Springer and ACM International Conferences on Advances in Computing and Communication (ACC-2011), Kochi, Kerala, India, 3rd IEEE International Conference on Computer Technology and Development (ICCTD 2011), Chengdu, China; and 2nd International Conference on Computer, Communication, Control and Information Technology (ScienceDirect-Elsevier), 2012.