Black Box Testing based on Requirement Analysis and Design Specifications

Harsh Bhasin
Assistant Professor
Department of Computer Science
Jamia Hamdard

Esha Khanna
M. Tech Scholar
Department of Computer Science
A.I.T.M, Palwal

Sudha
Assistant Professor
Department of Computer Science
A.I.T.M, Palwal

ABSTRACT
Black Box Testing is used when code of the module is not available. In such situations appropriate priorities can be given to different test cases, so that the quality of software is not compromised, if testing is to be stopped prematurely. This paper proposes a framework, which uses requirement analysis and design specification, to prioritize the test cases. The work would be beneficial to both practitioners and researchers.

General Terms
Algorithm, Reliability, Verification.

Keywords
Black Box Testing, Requirement Analysis, Design Specification, Prioritization.

1. INTRODUCTION
In order to develop robust software, sound testing techniques are required. It is important to comprehensively and qualitatively test each module of software so that the maintenance cost can be reduced. However, code of the software is not always available. In such scenarios, Black Box Testing comes to our rescue. Black box testing is a type of testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to the selected inputs and execution conditions [1]. There are many ways in which the task of Black Box testing can be carried out. Some of them are Boundary Value Analysis, Robustness, Worst Case, Equivalence, Cause-Effect and Decision Table Based testing.

In such cases, where the code of software is not available, test cases generated via Black Box testing technique should be such that the quality of the testing remains the same. The primary aim of testing should be to expose errors in the software and to avoid potential failures. The test cases designed, in order to accomplish the above task, are not equally important. Some of them deal with database entries while other deal with labels etc. The latter are not as important as the former. This paper proposes a framework to prioritize the test cases on the basis of requirement analysis and design. The test cases can be prioritized using the above premise. This prioritization would lead to the segregation of test case suite into different classes having different priorities. This would also help in situations where in testing is to be prematurely terminated. The prioritization techniques are important in regression testing is a well known fact but, their use in Black Box testing has largely being ignored. This paper paves the way of prioritization of test cases in Black Box testing.

The paper has been organized as follows. Section two of the paper presents the background of proposed work, section three presents a brief literature review, section four explains the proposed framework and section five concludes.

2. BACKGROUND
2.1 Requirement Analysis
In order to start a project, the first step is to gather requirements from the customers. These requirements serve as an objective of the project. Requirements gathered from a customer may be explicit or implied. These implied requirements must be transformed to the stated requirements. These requirements are then analyzed in order to know whether they are feasible and achievable. The purpose of requirement analysis is to refine the customer requirements on the basis of performance, functions and constraints. Requirements can be categorized as customer requirements, derived requirements, functional requirements, performance requirements and design requirements [2].

Analyzed requirements form the base of design phase during the software development process. Therefore the requirements after analysis must be achievable, verifiable, clear and consistent. Status of the requirements must be traced any time and must be unambiguous. These requirements are then documented properly. Thus activities performed during requirement analysis are:

- Requirement gathering
- Requirement analysis
- Requirement documentation

2.2 Design Specification
Software design is an iterative process through which requirements are translated into a ‘blueprint’ for constructing the software [3]. All the gathered project requirements and objectives are transformed into project design, which is then used by the developer for coding phase. Thus, software design acts as a bridge between requirements analysis and development phase. A good design should incorporate all the explicit and implicit requirements of the customer. During design phase, designers decide how the requirements can be translated into a working project. Therefore, this phase of software development deals with deciding what hardware and software will be used to build required product, what algorithms will be used, what data structures will be used, what will be the flow of data between the interfaces, what processes will be followed during coding and what will be the criteria of testing and acceptability of the product. All these details are documented properly. The objective of design documentation is to provide an efficient, modular design that

will reduce the system complexity and result in an easy implementation [4]. Output from design specifications is used by development team.

2.3 Previous Work

The present work is a part of our larger endeavor to develop an automated test case generator. The system would generate both types of test cases, via Black Box and White Box. The generation of test cases via White box testing has already been carried out using Cellular Automata [5, 6]. The generation of test cases by not considering the internal code has been accomplished using Artificial Life [7]. It is desired to compare the test cases generated via Artificial Life and Lenten's loop with another technique. This paper proposes that very technique. The results obtained so far are encouraging. The system which generates overall test cases has two components [8]. The first component generates test cases on the basis of the code. The present work intense to replace the second component.

3. LITERATURE REVIEW

An extensive literature review has been carried out to find the gap in the existing literature. The review has been carried out using the guidelines of Kitchenham [9]. Some of the works have been summarized in table 1.

Table 1. Literature Review

Author	Premises	Verification
Mariani, L., et. al. [10]	The work presents a technique 'AutoBlackTest' for automatic creation of system test cases for interactive applications. Q-learning agent and Test Case Selector have been used to create test cases.	The technique has been compared to GUITAR. Results showed that AutoBlackTest provides more coverage and uncover more failures in compared to GUITAR.
Frezza, S.T., et. al.[11]	The work proposes a technique to create automated test cases on the bases of 'graph data model'. Relationships between design and requirement have been captured and have been used to generate test cases.	The work has been evaluated on floating point arithmetic and logical unit examples.
Chan, E.Y.K., Yu, Y.T. [12]	The work uses partial dynamic analysis to reduce the number of test cases in the test suite.	The work has been examined on path based methods and their efficiency has been evaluated
Hu, Y.T., Lin, N.W. [13]	The work proposes a structure to create test cases automatically via Black Box testing. Unification mechanism along with constraint solving mechanism has been used to create test cases.	The technique has been applied to Java methods. UML class diagrams have been used to describe java methods.
Chen, T.Y., et. al. [14]	The work proposes a technique 'mirror ART' which has been used for testing. Mirror ART is a combination of mirroring and ART technique.	The technique is simulated using a square's input domain and results are more cost effective than ART.
Blanco, R. et. al. [15]	The work proposes a technique called Scatter Search metaheuristic, to generate the test cases on the basis of branch covering criteria.	The technique has been evaluated on 13 benchmark programs. It has been also compared with other test case generators like GA, TSGen, EDA, etc.
Kanatamnehi, H.V. et. al. [16]	In order to improve the coverage and efficiency of Black Box testing, the work proposes a dynamic measure 'potential of a branch'. It is evaluated by combining structural information and coverage information. An approach magnifying branches has also been used to increase branch coverage.	The technique has been implemented on four different varying size programs; triangle (90 lines, 20 branches), calendar (419 lines, 42 branches), roots (245 lines, 41 branches) and max (37 lines and 19 branches)
Tyer, B., Soundarajan, N. [17]	The paper proposes an approach to test grey box behavior of hook methods without knowing the source code.	The work has been implemented on the case study implemented on C.
Verma, D., Karambir [18]	The paper proposes finite automata based black box testing techniques for component based software. Both DFA and NFA testing techniques have been proposed.	The work has been tested using five UML diagrams on online shopping catalog.
Khan, M., Khan, F. [19]	The paper compares three strategies for testing, namely, White Box, Black Box and Grey Box along with their techniques.	All the conventional testing techniques of Black Box, White Box and Grey Box are compared.

Flores, A., Polo, M. [20]	The work proposes a technique called back to back testing for testing the replaced components in software.	The work has been implemented using testooj tool for java components.
Roshan, R., et. al [21]	The work presents a survey on various search based techniques like hill climbing etc. for software testing.	The work shows the increase in search based testing techniques in the recent years.

4. PROPOSED WORK

The work examines the importance of Requirement Analysis and Design Specification. The work creates a module description document and for each module input and output specifications are gathered. This is followed by determination of ranges and creation of test cases. In order to generate test cases, a Cellular Automata system is used [5, 6], if code is known and an Artificial Life system is used when code is not known [7]. A high level description of the procedure is given as follows.

```
Algorithm
1.  Modules[] ← extractModuleSpecification(RequirementAnalysis);
2.  for each module i  in modules[]
3.      {
4.      inputs[] ← extractInput(module_i);
5.      outputs[] ← extractOutput(module_i);
6.          {
7.          for each input in inputs[]
8.              {
9.              for each output in outputs[]
10.                 {
11.                 inputRange ← extractRange(input);
12.                 outputRange ← extractRange(output);
13.                 x ← random[] % inputRange;
14.                 y ← random[] % outputRange;
15.                 limit (x, y) in TCS;
16.                 }
17.             }
18.         }
19.     }
TCS- Test case suite
```

Fig 1: Algorithm to prioritize test cases using Requirement Analysis and Design Specifications

The methods and variables used in the algorithm are explained in Table 2 and Table 3 respectively.

Table 2. Methods used in Algorithm

Method Name	Parameters	Description
extractModuleSpecification	RequirementAnalysis	The function extracts the specification of modules from RequirementAnalysis and stores in array modules.
extractInput	$module_i$	The function extracts all the inputs from the given parameter module and saves it in array inputs.
extractOutput	$module_i$	The function extracts output of a corresponding module and saves it in array outputs.
extractRange	input, output	Range of input or output (as specified parameter) is extracted and is stored in inputRange or outputRange respectively.
Limit	(x, y)	The function stores the range (x,y) of a module in TCS.

Table 3. Variables used in Algorithm

Variable Name	Data Type	Description
modules	Array	It stores the module specifications for each module given in RequirementAnalysis.
Inputs	Array	It stores all the inputs extracted from a particular module.
outputs	Array	It stores all the outputs extracted from the corresponding module.
inputRange	Integer	It specifies the extracted input range for the corresponding input.
outputRange	Integer	It specifies the extracted output range for the corresponding output.
x	Integer	It stores the value calculated as random[] mod inputRange.
y	Integer	It stores the value calculated as random[] mod outputRange.

The work can be summarized as follows.

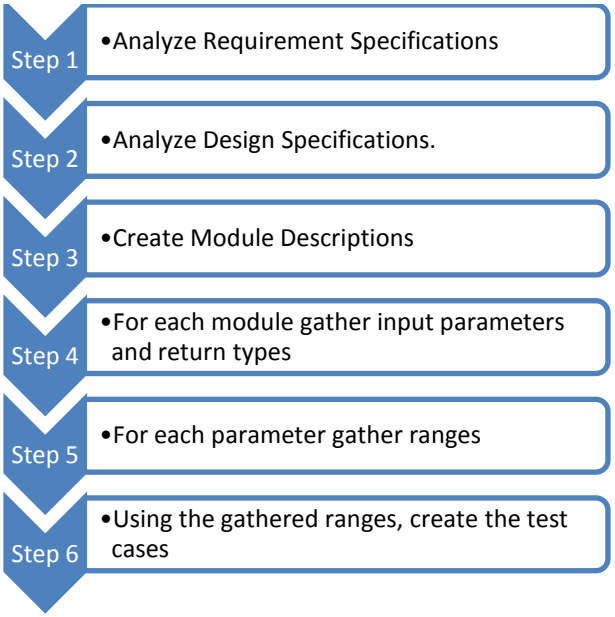

Fig 2: Steps of the proposed work

5. CONCLUSION

Requirement Analysis forms the backbone of software. This backbone was not being used in one of the most important tasks which ensure the quality of software, which is BBT. The work proposes a framework to prioritize test cases using the above premise. The concept of prioritization, though extensively used in regression testing, has not been used in BBT. The paper paves way of the concept of prioritization in BBT. The paper is based on an extensive literature review which was carried out to find gaps in the existing frameworks. It is a part of our larger goal to develop an automated test data generation system in which test cases have been assigned appropriate priorities.

6. REFERENCES

[1] IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries; IEEE; New York, NY.; 1990. Black, Rex; (2002). Managing the Testing Process (2nd ed.). Wiley Publishing.

[2] Kurniawan, S. 2001. System Engineering Fundamentals.

[3] Pressman, R.S. 2010. Software engineering: a practitioner's approach. McGraw-Hill Higher Education.

[4] Frank, J., et. al. Design Specification. SJSU Online Library System.

[5] Bhasin, H., Singla, N., Sharma, S. 2013. Cellular Automata Based Test data Generation. ACM SIGSOFT Software Engineering Notes.

[6] Bhasin, H., Singla, N. 2013. Cellular-Genetic Test data Generation. ACM SIGSOFT Software Engineering Notes, Volume 38 Number 5.

[7] Bhasin, H., Shewani, and Goyal, D. 2013. Test Data Generation using Artificial Life. International Journal of Computer Application. 67, 12, 34-39. paper

[8] Bhasin, H., 2014. Artificial Life based test data generation, ACM Sigsoft software engineering notes, 39, 1.

[9] Kitchenham, B.A. et. al. 2010. Systematic literature reviews in software engineering .A tertiary study, Information & Software Technology .INFSOF , vol. 52, no. 8, pp. 792-805, 2010

[10] Mariani, L., et. al. 2012. AutoBlackTest: Automatic Black-Box Testing of Interactive Applications, IEEE, DOI: 10.1109/ICST.2012.88

[11] Frezza, S.T., et. al. 1996. Linking requirements and design data for automated functional evaluation, Elsevier.

[12] Chan, E.Y.K., Yu, Y.T. 2004. Evaluating several path-based partial dynamic analysis methods for selecting black-box generated test cases, IEEE, DOI: 10.1109/QSIC.2004.1357946.

[13] Hu, Y.T., Lin, N.W. 2010, Automatic black-box method-level test case generation based on constraint logic programming, IEEE, Computer Symposium (ICS), 2010 International, pp: 977 – 982 , DOI: 10.1109/COMPSYM.2010.5685369.

[14] Chen, T.Y., Kuo, F.C., Merkel, R.G., Ng, S.P. 2004. Mirror adaptive random testing, Elsevier, DOI:10.1016/j.infsof.2004.07.004

[15] Blanco, R., Tuya, J., Adenso-Diaz, B. 2009. Automated test data generation using scatter search approach. Information and Software Technology, 51,4, 708-720.

[16] Harish V. Kantamneni, et. al. Structurally Guided Black Box Testing.

[17] Tyer, B., Soundarajan, N. 2004. Black Box Testing of Grey Box Behavior. Springer, volume 2931, pp 1-14

[18] Verma, D., Karambir. 2012. Component Testing Using Finite Automata, Indian Journal of Computer Science and Engineering (IJCSE).

[19] Khan, E., Khan, F. 2012. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No.6, 2012.

[20] Flores, A., Polo, M. 2009. Testing-based Process for Evaluating Component Replaceability. Elsevier, pp 101-115.

[21] Roshan, R., Porwal, R., Sharma, C.M. 2012 Review of Search based Techniques in Software Testing, International Journal of Computer Applications, volume 51, august 2012.