Automated Generation of Functional Test Cases and Use Case Diagram using SRS Analysis

Arjinder Singh
P.G. Student
Chandigarh University
Amritsar, India

Sumit Sharma
Assistant Professor CSE
Chandigarh University
Ludhiana, India

ABSTRACT

Software testing mainly consists of three types of approaches i.e. specification based testing, model based testing and coding based testing. In specification based testing, major concern is to find the missing logic defects that cannot be find by using other types of testing. Specification testing mainly uncovers the specification problems in Software Requirement Specification (SRS). An approach for the generation of functional test cases from SRS has been presented in this paper. This approach will help to achieve the early detection of faults and will reduce the time, cost and effort of the developer. Proposed model will automatically generate the functional requirements from the SRS. Template for Use Case diagram will be automatically generated from the functional requirements. Activity diagram will be used to generate the Activity Dependent Table (ADT) and hence Activity Dependent Graph (ADG) will be generated from ADT. Functional test paths will be generated by applying the Depth First Search Algorithm (DFS) as a searching algorithm. Finally we will generate the test cases from the functional test paths.

Keywords

SRS, Use Case Diagram, Activity Diagram,DFS

1. INTRODUCTION

Software testing is one of the most salient parts of software development process. Testing is liable for assuring the reliability and quality of the software system. Software products size and complexity is growing with the growing demands of users, hence time and effort needed for appropriate testing is increasing at an expeditious rate. Test cases are designed for every software application. Test cases can be generated in two ways either automatic or manual. Manual generation of test cases have number of shortcomings like time consuming, error prone and high probability of uncovering the important scenarios of the system [1]. Whereas automatic generation of test cases overcome the problems faced by the manual testing. Prior generation of test cases, results in finding ambiguities and inconsistencies in the software requirement specification and various other design documents. Hence it will leads towards letting down the cost of developing the software systems as number of errors are reduced at early stages of software development [2]. Test cases generation can be done on the basis of three testing strategies i.e. specification based testing, model based testing and coding based testing. Code based testing ensures the detection and correction of errors in the software system but it does find the missing logic defects in the software system. Code based testing does not guarantee the availability of all features in the software system that are specified by the users. Design based testing starts from the design of the software system. With the development of UML diagrams it becomes easy for the software tester to tests the system, whether the system is fulfilling the requirements specified. But design based testing also have some flaws as it does not start from the root level i.e. the specification level as it totally dependent on the design document of the software system. Specification based testing starts from the root of the testing process i.e. by analyzing the software requirement specification document that is generated at the time of requirements gathering. Specification testing approach is also known as functional testing. Specification based approach has following benefits

- Unveils specification problems
- Best for missing logic defects
- Applies at all granularity levels of software system
- Also responsible for improving schedule and budget problems in software testing [3]

Mostly test case generation uses code based or design based testing strategies. Specification based testing also incorporates the design based testing but at the second level. Firstly it generates the functional requirements from the software requirement specification and after identifying all the actors and use cases from the software requirement specification, it generates the use case diagram. Unified modeling language has facilitated us with the number of UML diagrams. From nine UML diagrams, UML Use Case diagram and Activity Diagram are used for describing the behavior by portraying the sequence of use cases and activities in a system. Use case diagram and activity diagram are two of the diagrams that plays vital role in the generation of test cases. Use case diagram becomes the major source for the activity diagram. For the appropriate coverage of all activities it is necessary to select the suitable coverage criteria which ensure the covering of all activity nodes in the system.

The rest of work is organized as follows. Section II will illustrate the Preliminaries and basic concepts. A review of work related to problem is presented in Section III. Section IV includes the proposed methodology for the generation of functional test cases. And section V represents the conclusion and future work.

2. PRELIMINARIES AND BASIC CONCEPTS

In this section we will discuss the basic concepts and definitions.

2.1 Functional Testing

Functional Testing also have other names referred as black box testing and specification based testing [4]. It is named as black box because in this approach tester sees the program as a black box i.e. tester ignores the way how it is written. Tester just considered about the requirement of the software.

Functional testing is not similar with white box testing where internal behavior of the system is needed. It is complementary approach that is related with uncovering the class of errors that with code based testing cannot be done.

2.2 Test Case Generation

Test case generation is mainly focused on the programming portion of the system. Test cases based on source code does not resolve the specification problems. It only handles the coding problems so it is necessary to generate the test cases from the software requirement specification available for developing the software. Generation of test cases from the specification also provide the advantage of shifting highly needed modules earlier in the development process. If test cases are generated from the specifications then test engineers will also be able to find inconsistencies in specifications and design of the system, allowing the improvements in specification and design of the system. Test cases generation can be possible in two ways either manually or automatically. Manual generation of test cases leads towards many problems like time consuming, effort etc. Automatic generation of test cases are benefited to software industry as it is responsible for generating relevant and reliable test cases ensuring the possible coverage criteria.

2.2 Software Requirement Specification

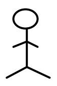
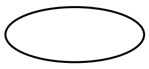
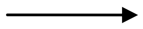
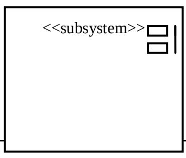
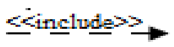
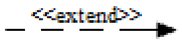
Software requirement specification is one of the most important components of software. This component includes all requirements specified by the users. This component is prepared after the requirement gathering phase of the software development life cycle. This component can also be serving as a legal document. Functional requirement specified in the SRS is of major concern. The SRS documents contain the entire necessary requirements that are needed for the development of any software project. Complete understanding of the product being developed or to be developed is required to better obtain the requirements from thee customers or users. And this can be achieved by having detailed and continuous meeting between project team and customer.

2.3 UML Diagrams

Use case diagram is a one of the simple diagram of the UML. Use case diagram mainly shows the interaction of different users with the system. Use case diagram associates possible functionality of the system with its appropriate user.

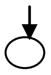
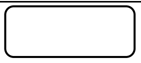
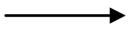
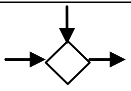
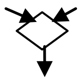
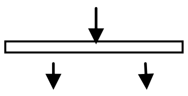
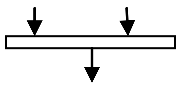
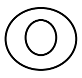
In use case diagram users are represented as an actor and the numbers of functions are represented as a use cases. An actor has number of functions to be performed. Use case diagram is used to associate each with their corresponding appropriate functions.

Table 1. Control Nodes of Use Case Diagram

Names	**Symbols Used**	**Description**
Actor		It is a stick persons which represents the various types of users of the system
Use case		It is of oval shaped which is used to represents the various functionality of the system
Association		It is an arrow which is used to provide association between actors and use cases
Subsystem		It is used to occupy the all the use cases of the system
Include		It is used to provide association between different use cases which have been include by the use case
Extend		It is used to provide association between use case which are extended

Activity diagram provides the graphical representations of different stepwise activities carried in a system. Activity diagrams depict the complete flow of the system. In activity diagram all activities are arranged in a stepwise manner so that each task can be easily executed. Different symbols used in activity diagrams are represented as below.

Table 2. Control Nodes of Activity Diagram

Names	**Symbols used**	**Description**
Start		It is start symbol which represents start of the activity
Action node		Action node is used to represents the every action
Arrow		Arrow symbol is used to show the activity moving from one action to another
Decision node		Decision node contains one incoming and multiple outing nodes
Merge node		Merge node contains multiple incoming nodes and one outgoing node
Fork		Fork node is used to split the activity into multiple parts
Join		Join node is used to join the multiple activities
Final node		Final node indicates the end point of the activities

3. RELATED WORK

Test cases are mostly generated from activity diagram but there are also some other alternatives available for the generation of functional test cases.

Noraida Ismail et al. [5], presented an approach for generation of test cases from the use case diagram based on specification based testing. Test cases from use cases are driven automatically in this approach. This approach mainly considers two steps i.e. generation of use case diagrams from the software requirement specification of the system and then generating test cases from the use case diagram.

Ajay kumar jena et al. [6], presented an approach for the generation of test cases from the activity diagram which is design based testing. Test cases from the activity diagrams are generated by using the coverage criteria method. This paper has considered the case study of ATM cash withdrawal and generated test cases are further optimized sing genetic algorithm.

Andreou et al. [7], presented an approach for the generation of test data using data flow graph. This paper proposes the integrated approach of data flow module with existing testing framework. The performance of this approach is tested on the basis on number of different samples of programs having different size and complexity.

Nebut et al. [8], presented an approach for test case generation by considering the UML use case diagram. It also addresses the problems associated between major views and execution of test cases. This approach uses the statement coverage criteria for test cases development. This approach is divided into two parts i.e. handling high levels concerns and data complexity is taken into account with the use case scenarios.

Kundu et al. [9], the main scope of this paper is to consider use case as a base for the generation of test cases. This approach has used the activity path coverage criteria for the generation of test cases. This approach is also able to handle the faults like synchronization and loop faults.

Tripathy et al. [10], presented an approach for the generation of test cases with the togetherness of UML activity diagram and Sequence diagram. Activity graph from the activity diagram and sequence graph from the sequence diagrams are derived first and these theses are integrated together to form a new graph and then traversing is applied on that graph for the generation of test cases.

Mingsong et al. [11], presented an approach for test case generation using activity diagram by considering a design as a specification. Rather than generating test cases from activity diagram, it is using some randomly generated test cases based on certain coverage criteria.

From the previous work it is concluded that the most of the work is done in the field of design based testing by using the activity diagrams. Test cases generation from the specification based testing is very important as it disclose the specifications problems and responsible for the delivery of software with its complete specifications.

It is also observed that no work of automated analysis of software requirement specification has been done. And also there is no automated framework for the generation of use case diagrams. Use case diagrams are manually drawn by users in various UML tools.

4. PROPOSED FRAMEWORK

An approach is introduced to perform the specification based testing to unveil the specifications problems in the system. This approach mainly deals with the automated analysis of software requirement specification to find the appropriate input for the use case diagram. As use case diagrams needs input like various actors and use cases, our motive is to find the actors and use cases from SRS automatically. Another goal of our approach is to automatically generate the use case diagram from the refined input generated from the SRS. Then automatically generated use case diagram are used to generate the activity diagram. Hence functional test cases from activity diagrams are created.

4.1 Proposed Framework

A framework that we are proposing will work as mentioned below and the framework is shown in Fig. 1:

i. Analyze the software requirement specification (SRS) to find the appropriate use cases and actors. Analysis of SRS is of two types automated analysis and manual analysis
ii. In automated analysis, two types of use case diagrams are generated i.e. fully automated and semi-automated.
iii. In manual analysis complete user based input use case diagrams are generated.
iv. Users have to choose the each way to generate use case diagrams.
v. Analyze the use case diagrams developed by fully automated, semi-automated and complete user based input approach.
vi. Develop one or more Use Case Activity Diagram (UAD) based on fully automated, semi-automated and complete user based input approach.
vii. Checks the uniformity of the UADs.
viii. Perform parsing of XML files of different UADs.
ix. Generate Use Case Activity Dependent Table (UADT) based on UADs.
x. Generate Use Case Activity Dependent Graph (UADG) based on UADTs.
xi. Apply traversing technique to generate functional test cases.
xii. Develop one or more functional test cases.
xiii. Finally compare the results of all three approaches.

4.1 Analysis of Software Requirement Specification

Software requirement specification for a software application is analyzed by automatically and manually. SRS is analyzed to find the input for the use case diagrams. So functional requirement of the SRS is analyzed to obtain the appropriate input i.e. use cases and actors for the use case diagram by both way.

4.2 Generation of Use Case Diagrams

Use case diagrams are generated by three ways i.e. fully-automated, semi-automated and complete user based input approach. In fully automated approach, use case diagram are generated from the input provided by automated analysis of SRS. In semi-automated approach, use case diagrams are generated from the input provided by automated analysis of SRS but with human invention. In complete user based input approach, use case diagrams are generated from the input provided by manual analysis of SRS.

4.3 Analysis of Developed Use Case Diagram

Analysis of use case diagram involves the manual checking of use case diagram generated by all three approaches. In this software engineer or user will check the use case diagram to ensure that whether it is according to the specified requirements. If any changes found, then use case diagrams are modified to ensure the better covering of functional requirements mentioned in the system specification.

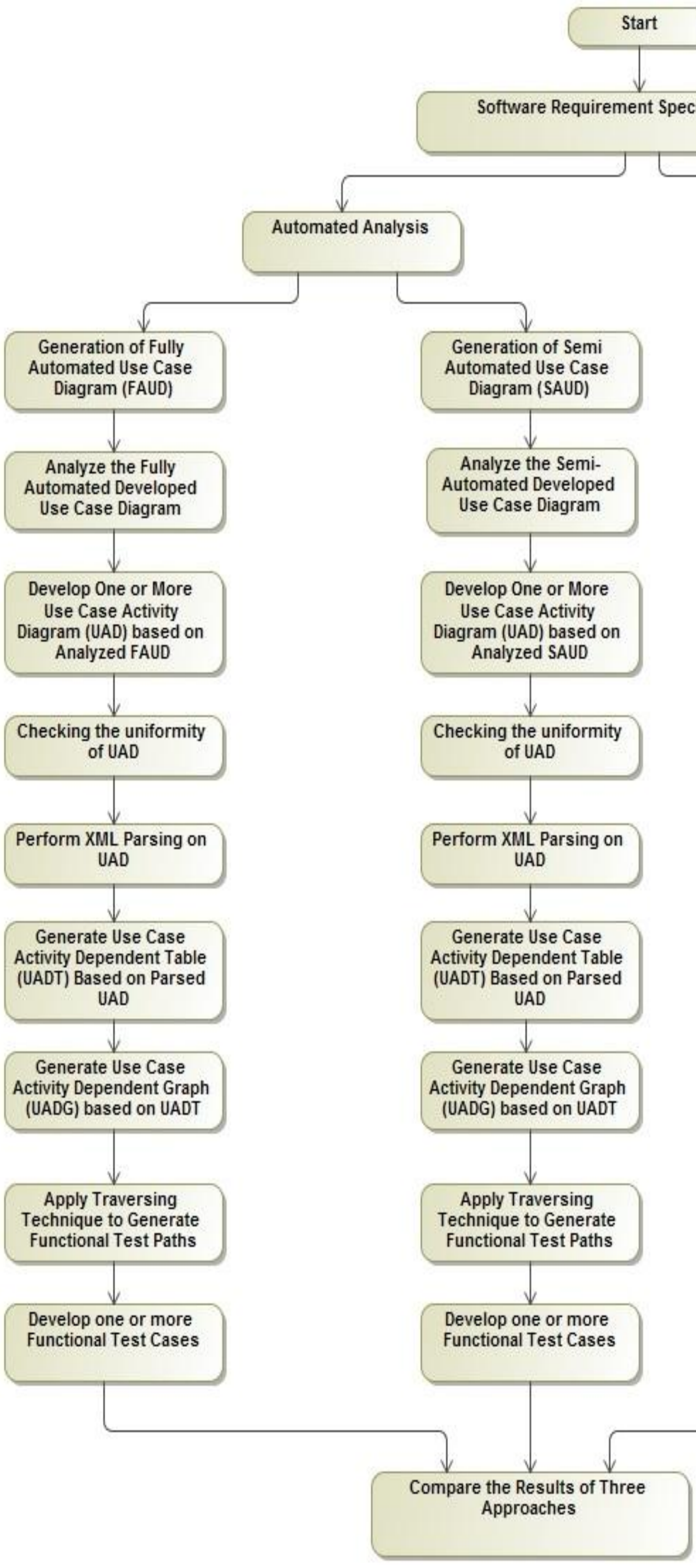

Start
Software Requirement Specification (SRS)
Automated Analysis
Manual Analysis
Generation of Fully Automated Use Case Diagram (FAUD)
Generation of Semi Automated Use Case Diagram (SAUD)
Complete User Based Input Use Case Diagram (UBUD)
Analyze the Fully Automated Developed Use Case Diagram
Analyze the Semi-Automated Developed Use Case Diagram
Analyze the User Based Developed Use Case Diagram
Develop One or More Use Case Activity Diagram (UAD) based on Analyzed FAUD
Develop One or More Use Case Activity Diagram (UAD) based on Analyzed SAUD
Develop One or More Use Case Activity Diagram (UAD) Based on Analyzed UBUD
Checking the uniformity of UAD
Perform XML Parsing on UAD
Generate Use Case Activity Dependent Table (UADT) Based on Parsed UAD
Generate Use Case Activity Dependent Graph (UADG) based on UADT
Apply Traversing Technique to Generate Functional Test Paths
Develop one or more Functional Test Cases
Compare the Results of Three Approaches

Fig. 1: Proposed Framework

4.4 Developing Use case Activity Diagrams (UAD)
Activity diagrams from the use case diagrams are generated. Various tools can be used to generate the activity diagrams. Activity diagrams clearly depict the flow of one activity to another activity to easily understand the working of system.

4.5Checking the uniformity of UAD
Uniformity of activity diagrams are checked to ensure the flow of one action to another. Consistencies of modules are checked in this step. It ensures that all units of activity diagrams are placed at their position and according to their roles. Once the uniformity is checked, XML files of activity diagrams are generated.

4.6 Parsing of XML Files
Parsing of XML files of different activity diagrams are performed to find the relevant information needed for the generation of Activity Dependent Table (ADT), Activity Dependent Graph (ADG) and for traversing the graph.

4.7 Activity Dependent Table (ADT)
ADTs are generated from parsed content provided by the parsing step. Activity Dependent Table contains the information like vertex name, dependency nodes, dependent nodes, in degree and out degree of the activity diagram.

4.8 Activty Dependent Graph (ADG)
Activity Dependent Graph is of tree like structure which has one root node and all other nodes are child nodes. ADT is defined as G (V,E) where G represents the graph, V represents the vertices i.e. set of nodes (activities) and E represents the edges i.e. flow from one activity to another.

4.9 Traversing Technique
Depth First Search (DFS) algorithm is used to traverse the ADG to show all possible functional test paths in the ADG. DFS ensures the proper coverage of all paths available in the graph. DFS starts traversing the graph with one as a root node and traverse it as far as possible before backtracking

4.10 Developing Functional Test Cases
By using all the functional test paths, functional test cases are generated. Test case represents some conditions, by following these conditions tester checks that whether a software system or one of its modules is working accordingly as it was specified to do.

4.11 Comparing the Results
Functional test cases generated by all three approaches are compared to measure the number of functional test cases produced. Results are also compared to know which approach is covering all specifications specified in the SRS. And also time complexities of these approaches are analyzed.

5. CONCLUSION AND FUTURE WORK
Functional test cases from software requirement specification are generated by developing use case diagram and activity diagram. Automation approach for the analysis of SRS, generation of use case diagram and generation of functional test cases has been introduced. Automation results in saving maximum effort consumed in analysis of SRS and drawing of use case diagram manually in any UML tool or by using pen or paper. Automation also results in saving cost included in the testing of the software application. Mostly in code based testing, we are able to find errors in coding or logics, we are not able to ensure the covering of all requirements specified in the SRS. Our automation approach ensures the maximum coverage of all requirements specified in the SRS. Hence possibility of reliable system delivery to the customer increases. An approach for automated analysis of SRS must be improved to identify the composite words. And this approach can be further extended for the other UML diagrams.

6. REFERENCES
[1] Isabella, A., and Emi Retna. "Study Paper on Test Case generation for GUI Based Testing." *arXiv preprint arXiv:1202.4527* (2012).

[2] Prasanna, M., et al. "A survey on automatic test case generation." *Academic Open Internet Journal* 15.part 6 (2005).

[3] Young, Michal. *Software testing and analysis: process, principles, and techniques*. John Wiley & Sons, 2008.

[4] Gill, Nasib Singh. *Software engineering: software reliability, testing and quality assurance*. Khanna Book Publishing, 2007.

[5] Kundu, Debasish, and Debasis Samanta. "A Novel Approach to Generate Test Cases from UML Activity Diagrams." *Journal of Object Technology* 8.3 (2009): 65-83.

[6] Jena, Ajay Kumar, Santosh Kumar Swain, and Durga Prasad Mohapatra. "A novel approach for test case generation from UML activity diagram." *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*. IEEE, 2014.

[7] Tripathy, Abinash, and Anirban Mitra. "Test Case Generation Using Activity Diagram and Sequence Diagram." *Proceedings of International Conference on Advances in Computing*. Springer India, 2012.

[8] Nebut, Clementine, et al. "Automatic test generation: A use case driven approach." *Software Engineering, IEEE Transactions on* 32.3 (2006): 140-155.

[9] Mingsong, Chen, Qiu Xiaokang, and Li Xuandong. "Automatic test case generation for UML activity diagrams." *Proceedings of the 2006 international workshop on Automation of software test*. ACM, 2006.

[10] Andreou, Andreas S., Kypros A. Economides, and Anastasis A. Sofokleous. "An automatic software test-data generation scheme based on data flow criteria and genetic algorithms." *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*. IEEE, 2007.