AIN SHAMS UNIVERSITY
Faculty of Computer
& Information Sciences
Information Systems Department

Generic Model for Test Case Generation from both UML Diagrams and Requirements

A thesis submitted to Information Systems Department, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt as a partial fulfillment of the requirements for The Degree of Master of Science in Computer and Information Sciences.

BY

Roaa Ahmed Mostafa El-Ghondakly

B.Sc. of Computer & Information Sciences, Information Systems Department, Faculty of Computer and Information Sciences, Ain Shams University

Under the Supervision of

Prof. Dr. Nagwa Lotfy Badr

Professor of Information Systems,
Vice Dean of Education and Students Affairs,
Faculty of Computer and Information Sciences,
Ain Shams University

Dr. Sherin Mohamed Mahmoud Moussa

Associate Professor,
Information Systems Department,
Faculty of Computer and Information Sciences,
Ain Shams University

October 2016

ABSTRACT

Software testing is accounted to be an important phase in software development life cycle in terms of cost and manpower. It is considered the key to success of any software. Consequently, many studies have been conducted to minimize the associated cost and human effort to fix bugs and errors, and to improve the quality of the testing process by generating test cases at early stages. There exist many software development models, as waterfall model, agile model, etc. Test cases can be generated at different phases (analysis phase, design phase and after development phase). Though, test cases generation at early stages is more effective rather than that after development, where time and effort used for finding and fixing errors and bugs are less than that after development. At a later stage, fixing errors results in enormous code correction, consuming a lot of time and effort.

Requirements-based testing is a testing approach in which test cases are derived from the requirements. Requirements represent the initial phase in software developments life cycle. Requirements are considered the basis of any software project. Therefore, any ambiguity in natural language requirements leads to major errors in the coming phases. Moreover, poorly defined requirements may cause software project failure. Model-based testing (MBT) is the automatic generation of software test procedures, using models (UML diagrams) of system requirements and behavior. Whilst Search Based Software Testing (SBST) is a branch of Search Based Software Engineering (SBSE), in which optimization algorithms are used to automate the search for test data that maximizes the achievement of test goals, while minimizing

testing costs. However, most of related studies considered only one type of behavioral diagrams with a lot of human intervention.

In this thesis, we study the different paradigms of testing techniques for generating test cases, where we investigate their coverage and associated capabilities. We then propose a consolidated model for a generic automated test cases generation. The model consists of two main parts to handle the UML diagrams, the software requirements specification (SRS) documents generated from the waterfall development model and the user stories generated from the agile development model, which are considered as the main deliverables of the analysis and design phases in the software development life cycle.

In the first part (UML Test Cases Generation Part), an optimized automated approach for generic and dynamic test cases generation is proposed. It is generic and dynamic as it can be applied on different types of behavioral diagrams (i.e. activity diagram, state diagram, uses case diagram, etc) for multi-disciplinary domains. While automation is considered to generate test cases with minimum human intervention, which will consequently help to minimize total cost. An optimization technique is applied to optimize the generated test cases to ensure the quality of results. The proposed approach merges model-based testing with search-based testing to automatically generate test cases from different behavioral diagrams, i.e. use case, activity, etc. Whereas in the second part (Requirements Test Cases Generation), we propose another novel automated approach to generate test cases from requirements. Requirements can be gathered from different models either waterfall model (functional and non-functional) or agile model. SRS documents, non-functional requirements and user stories are parsed to generate test

cases. The proposed consolidated model uses text mining and symbolic execution methodology for test data generation and validation, where a knowledge base is developed for multi-disciplinary domains. The proposed model is a time saving model where it will take about 70 minutes (4200000 milliseconds) to perform all the proposed steps manually for UML Test Case Generation Phase. While it takes about 0.204 minutes (12269 milliseconds) when using our automated system to perform the same steps which ensure time and cost saving.

Acknowledgment

Thanks God for your grace and mercy on my whole life. I am grateful to have Professor Dr. Nagwa Badr and Dr. Sherin Moussa as my supervisors. I would like to thank them for their cooperation, support and encouragement during my study and research, in addition to their final revision of the thesis.

Special thanks to my husband, daughter Laila and my whole family who have always supported me and helped me in this achievement.

TABLE OF CONTENTS

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Table **Page**

CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

Software testing is considered one of the important phases in software development life cycle. Testing process is considered the key to success of any software. The development life cycle total cost is considered to be high. For this reason, many studies have been conducted to minimize the associated cost and human effort to fix bugs and errors, and to improve the quality of testing process by automatically generating test cases [1]. Test automation is the process of using separate software to manage and evaluate the fulfillment of test cases and to compare the expected outcomes with the generated ones [2]. Moreover, the automation process is used to generate test cases with minimum human intervention, which will consequently help to minimize total cost. Different studies are carried out for test case generation techniques [3].

There were many approaches in testing, as code-based testing, requirement-based testing, model-based testing and search-based testing approach. Code-based testing corresponds to the testing that is carried out on code development, code inspection, unit testing in software development process. Requirements-based testing, on the other hand, is a testing approach in which test cases are

derived from requirements. These tests aim to test all the requirements. However, it is difficult to achieve complete testing as some requirements can be tested by a single test case, while others need to be verified by a set of test cases. Moreover, even when a full set of requirements-based test cases are applied to a system, this does not ensure that the entire system has been tested [13]. Testing requirements helps in increasing the quality of results specially when carried out at an early stage. In waterfall models, testing can be conducted as soon as executable software (even if partially complete) exists. Most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an agile approach, requirements, programming, and testing are often done concurrently. According to the continuous changing and increasing of requirements, customer involvement is mandatory. Agile development is aligned with the view to face the challenges of an increasingly volatile marketplace, changing requirements, customer involvement, priorities and shorter deadlines [14]. Whereas functional requirements are mostly written using use cases with textual description, which is too complicated to perform further processes on them as generating a test case process.

Model-based testing (MBT) is the automatic generation of software test procedures, using models (UML diagrams) of system requirements and behavior. The Unified Modeling Language (UML) diagrams are divided into two main parts; structural diagrams (i.e. class diagram) and behavioral diagrams (i.e. activity, use-case and state diagrams). Structural diagrams are used to show the structure, style or design of the software, while behavioral ones are used to clarify the steps in which the software will pass through until it reaches the desired output. In other words, it shows the flow of events. Whilst Search Based Software Testing (SBST) is a branch of Search Based Software Engineering (SBSE), in which optimization algorithms are used to automate the search for test data that maximizes the achievement of test goals, while minimizing testing

costs [9, 10, 11, 12]. The optimization of the generated output can have different techniques, as reducing the number of test cases, test cases prioritization, as well as minimizing time, increasing performance, maximizing the quality of outcomes, etc.

Testing may be performed in the last phase of the software development life cycle or at an early stage. If the process is carried out at early phase of the development life cycle; it saves more time and effort to detect errors. At later stages, enormous errors would be generated as soon as the code is completed, where it demands a lot of code correction and modification. Therefore, testing process should be carried out at the beginning of software development life cycle (requirements and design phases) to save time and cost. For this reason, model-based testing [4, 5, 6] and requirement-based testing [13] were preferable rather than code-based testing [7, 8] to generate test cases from UML diagrams (behavioral diagrams) during the design phase.

Considering the previously mentioned testing techniques, it was found that the code-based methodology does not detect faults early as in the model-based and requirement-based approaches because it depends on the source code of the program to be tested while the other two approaches depend on the diagrams and requirements specifications that was constructed and found in an early stage before implementation.

The quality of the generated test cases is the main factor that determines the quality and efficiency of the testing phase. Whereas the test cases should be validated against recognized quality standards, which would determine the degree of their functional coverage that identifies their level of applicability as well as their acceptable form [15, 16, 17, 18, 19].

Many metrics are being used to measure the quality of the test cases being generated, as the cost, time, complexity of generation, effort, coverage criteria, etc. [20, 21]). The coverage criteria are considered as a set of metrics that is used to check the quality of the test cases extracted from the behavioral models. This metric checks how well the test cases cover the executable test paths that they are mapped to [22]. They contain many types, such as the branch coverage criterion, full predicate and full condition coverage criteria and all basic paths coverage criterion. Cyclomatic complexity is software metric that provides a quantitative measure of the logical complexity of programs developed by Thomas J. McCabe in 1976 [23, 24, 25]. It can be used in the context of the basis path testing methods. Whereas the basis path testing is a structured testing or white box testing technique that is used to design test cases intended to examine all possible paths of execution at least once. The value computed for the cyclomatic complexity defines an upper bound for the number of linearly independent paths in the basis set of a program [26]. The cyclomatic complexity can be computed using different ways; the number of regions in the graph corresponds to the cyclomatic complexity.

$$V(G) = E - N + 2 \tag{1}$$

where E is the number of edges and N is the number of nodes in the graph G.

$$V(G) = P + 1 \tag{2}$$

where P is the number of predicates i.e. binary decision nodes (with two outgoing edges).

The V(G) can also be used to get all the test paths required to test the execution and the exit of the loops. Therefore, test paths number computed by the cyclomatic complexity covers: Branch coverage (two outgoing edges from decisions), Condition coverage (multiple outgoing edges) and the All-basis paths

coverage that covers loops at least once. The cyclomatic complexity function V(G) has two main properties that make up a powerful testing criterion. The first property is its ability to determine the number of test paths that are necessary to achieve the full branch coverage. The second property is determining the number of test paths needed to achieve a full path coverage, which only requires covering all the linearly independent paths and not all the paths whose calculation is impractical.

Test data generation, on the other hand, is the process of generating data (output) according to simulated inputs [29, 30, 31, 32, 33, 34]. Symbolic Evaluation (also referred to as Symbolic Execution) technique is used for test data generation. It is used to simulate symbolic values of variables, instead of actual values, to act as an input to the system under test to find the expected output according to those inputs [27, 28]. Test data generation techniques are used to validate the generated data, since the validation process increases the quality of the generated test cases.

1.1 Problem Definition

The quality of any software should be ensured in every phase of the software development life cycle, starting from the requirements elicitation to the deployment of the final product. Consequently, the software testing process is one of the main activities carried out in the software development life cycle. It is often accounted for more than 50% of the total development costs. Thus, it is imperative to reduce the cost and the human effort in finding bugs and errors and to improve the effectiveness of software testing by automating the testing process. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or add additional testing that would be difficult to perform manually.