Chapter 1

Introduction
INTRODUCTION

1.1 Background and Context of Study

Testing has been an essential part of the software development life cycle. Testing is one of the phases of all process models of software engineering. It remains the most indispensable part of software quality assurance. High reliability of software is expected in the real world as it, otherwise, becomes obsolete.

Functional test suites are used to discover bugs in Software Under Test (SUT) and improve its quality. A good test suite uncovers more faults in the SUT. As test suite contains many test cases, the order of their execution plays an important role in increasing the fault detection ratio which can provide early feedback to the development team so as to help them to improve the quality of software.

Paramount importance is given for discovering bugs in software towards the improvement of quality in the research arena. The automatic test case generation became essential as the manually test data generation and adding test oracles is a tedious task in discovering bugs. It is too complex when there are no formal specifications to unearth the faults in SUT. In the literature many authors concentrated on targeting only one test goal at a time. In this approach, it is not easy to cover all targets. There might be certain coverage goals for which no inputs are available in order to cover them. Such goals are known as infeasible goals.

Coverage goals are presented in the form of coverage criteria while generating test cases. In other words, coverage criteria are nothing but a set of coverage goals. The well known criterion found in the literature is branch coverage. However, other approaches such as mutation testing can be used to generate test cases automatically. Moreover, coverage goals need not be independent. High code coverage and smaller test suite generation is an ideal solution. Therefore, it is important to generate representative test suits that ensure complete code coverage. Genetic Algorithms (GAs) are proved to be very useful for the generation of unit tests and well suited for testing object oriented software systems. They are well known for their capabilities to test complex objects through sequences of method invocations. Genetic algorithms
are suitable for meta heuristic search in evolutionary approach. Therefore, it is the natural choice in computer applications to have the best solutions. GA considers a single population initially and then an iterative evolution process is carried out until a best solution is found based on the fitness function.

As test suite contains many test cases, the order of their execution plays an important role in increasing the rate of fault detection which can provide early feedback to the development team so as to help them to improve the quality of software. Therefore, it is very useful to prioritize test cases that will lead to the increase in the rate of fault detection. However, prioritization of functional test suites is a challenging problem to be addressed. Recently Haidry and Miller [76] has proposed a family of test case prioritization techniques that use the dependency information from a test suite to prioritize for the said test suite. The nature of the techniques preserves the dependencies in the test ordering. Dependencies in test cases have their impact on the discovery of faults in software. The authors proved this hypothesis and their empirical results revealed it. However, they do not automate the extraction of dependency structures among the test suits which can help in effective prioritization of functional test suites. The automatic discovery of dependency structures as part of the comprehensive testing tool which also facilitates the whole test suite generation, mutation testing and test case prioritization based on dependency structures discovered automatically is the need for this research.

1.2 Prior Works on Research Study

Gordon Fraser and Andrea Arcuri [26] studied the need for the whole test suite generation. They explored the problems with targeting a single goal at a time. The issue in the traditional test case generation, which focused on a single coverage goal, is that coverage goals are not independent, sometimes coverage goals are infeasible and all goals are not equally difficult. A coverage goal is nothing but a branch in branch coverage. The traditional approach is to generate different test cases that cover a single coverage goal and then combine all test cases into a test suite. This approach resulted in more number of test cases and there were issues with coverage. While generating test cases, GA can be used as a search mechanism in order to help in generating test cases. The solution in the process of the test case generation with GA
is the test suite, which is a collection of representative test cases that ensure high coverage with less number of test cases. This feature is desired as it can help to improve the quality of test cases. Once test cases are generated it is important to execute them in proper order. The test cases are to be ordered or prioritized to increase the fault detection ratio.

Haidry and Miller [76] focused on the process of test suit prioritization. They used a hypothesis “dependencies among test cases can have their influence on the rate of fault detection”. Thus the test case prioritization plays an important role. It is a process of ensuring that the test cases are executed in proper sequence, in order to achieve the high rate of fault detection. The rate of fault detection is measured using the number of faults detected. So, it is essential to prioritize test cases to achieve optimal results [58].

Dependency Structure Prioritization volume refers to the count of dependencies while the DSP height indicates the depth in dependency levels. Direct and indirect dependencies are considered while computing DSP volume. On the other hand, the height of all test paths is considered for computing DSP height. Two graph measures are used for the best ordering of test cases for optimal results. Experiments are made with open dependencies and closed dependencies. Many real time projects were considered for experiments. Out of them, Bash is recorded to have the highest dependencies and CRM1 and CRM2 recorded the lowest. Many SUTs are tested with prioritization of test cases. The experiments are useful to know the fault detection rate when dependency structures are used for prioritization. A measure [68] used in is known as the Average Percentage of Faults Detected (APFD) for fault detection. The more APFD value is the more in the rate of detection of faults in SUT. All SUTs are tested with APFD measures under open and closed dependencies. Many DSP prioritization methods were considered and some other methods that do not use DSP measures can also be used in the process. The experimental results showed that DSP prioritization methods achieved higher results while Non-DSP prioritization methods could not achieve the high rate of fault detection. The empirical results proved the fact that DSP measures were able to increase the rate of fault detection for any given SUT. As explored in [76] there are many test case prioritization techniques like model-based [20], history-based [23] and knowledge-based [88]. The first one works based
on the model of the system, second one works based on previous execution cycles, and the third one works based on human knowledge to know how the task for the purpose of test case prioritization.

We focused on improving the test suite generation with mutation testing, and implementing a framework for the automatic discovery of dependencies in this research. The implicit discovery of dependencies decreased the time taken to identify dependencies. The researcher feels that, it is for the first time to have the automatic discovery of dependencies without manual intervention. It is a significant step forward in software engineering for the automatic test suite generation.

1.3 The Problem Definition

Automatic test case generation is an important and essential activity in software engineering. It reduces the wastage of time and effort involved in manual test case generation and testing. It was common practice to generate a test case for every coverage goal and combine all test cases to form test suite can be found in the literature for many years. A problem in this approach is that the size of test suite becomes unpredictable because the test case generated for fulfilling one goal might be useful for other goals as well. This characteristic is exploited of late to produce representative test suite that not only covers the whole code besides ensuring smaller size test suite. There are many problems when one goal is targeted at a time. For instance, some branches are difficult to consider for coverage while some branches are infeasible. It is still an open issue to find out how much time needs to be spent and difficulty prediction of coverage goals. Recent studies on the whole test suite generation and test case prioritization could not focus on the automatic discovery of dependency structures. The purpose of research is to discover dependency structures automatically to prioritize test cases in order to improve the fault detection ratio in generated representative test suits.

1.4 Motivation

The whole test suite generation considers all testing goals and generates representative test suite that will provide high coverage and reduces test suite in size. Moreover, the ability of test suite to be used for mutation testing can help unearth
hidden bugs. On the other hand, prioritizing test cases can have its impact on the
discovery of faults in SUT. In other words, the test case prioritization based on the
dependency structures can result in improving the fault detection ratio. These facts are
conceived from the review of literature. Moreover, there are uncovered areas in
research for instance, the automatic discovery of dependency structures has no
evidence in literature. In SUT, we are the first persons in discovering the dependency
structures automatically. This is the motivational step in doing the research work.

1.5 Drawbacks of the Existing System

The following drawbacks are found in the existing system while changing to the
proposed system- automatic test suite generation and test case prioritization,

1. The whole test suite generation could be improved further with provision for
mutation testing to increase bug detection probability.
2. The existing system does not provide support for the automatic discovery of
dependencies. It is very important to discover dependencies of two categories
such as open and closed in order to prioritize test cases.
3. In the existing literature there is not the comprehensive tool that caters to the
whole test suite generation and mutation testing, the automatic discovery of
dependency structures, and test case prioritization.

1.6 Suitability of the Proposed Research

The proposed work is suitable for research study as it has the huge impact on
the software industry. The comprehensive tool built as part of this research can be
used by software industries in the real world for testing Object Oriented Software.
This tool is especially useful for the automatic generation of representative test suits,
the automatic discovery of dependency structures and the test case prioritization in
increasing the rate of fault detection in the SUT.

1.7 Aims of this Study

The main aim of the thesis is to build a comprehensive testing tool that has
underlying algorithms for the whole test suite generation and mutation testing, the
automatic discovery of dependency structures and test case prioritization to improve
the fault detection ratio. To achieve the aim of the research the following objectives are identified.

1. To investigate the present state of the art in the whole test suite generation and mutation testing, the automatic discovery of dependency structures, and test case prioritization which results in increasing the fault detection ratio?
2. To propose methodology and algorithms whole test suite generation and mutation testing, the automatic discovery of dependency structures, and test case prioritization.
3. To implement methodology and underlying algorithms for the whole test suite generation and mutation testing, the automatic discovery of dependency structures and test case prioritization.
4. To build a comprehensive testing tool that has underlying algorithms for the whole test suite generation and mutation testing, the automatic discovery of dependency structures and test case prioritization to improve the fault detection ratio.
5. To evaluate and discuss the research outcomes.
6. To draw conclusions and provide recommendations for future work.

1.8 Organization of the Thesis

The structure of the rest of the thesis as follows. This section provides the brief overview of the ensuing chapters to help the reader to understand the essence of the chapters before actually navigating for more details.

Chapter 2 – Literature Review

This chapter reviews literature on prior works on software testing. It throws light into various aspects of software testing, including the automated generation of test scripts from formal test specifications, class level regression testing, detection of concurrent execution, semi-automated generation of test cases, and the automatic generation of test suites from decision tables and so on. It also reviews tools pertaining to software testing such as TestFul, JAMTester, JML-JUnit tool, JUB and TestGen4J. There are many recent developments in software testing methodologies which are reviewed in this chapter. They are cyclic dependencies and the quality of
software, test-driven software testing approaches; search based testing approaches, saving effort through bad smell detection, the whole test suite generation and test suite prioritization.

Chapter 3 - Genetic Algorithm for the Automatic Generation of Representative Test Suite for Mutation Testing

This chapter provides details of the proposed genetic algorithm and the methodology for generating representative test suites that reflect high coverage besides reducing the size of the test suite. Genetic algorithms are applied successfully to generate unit tests for testing the object oriented software product. GA is the one of the search based algorithms widely used to generate test cases. In this research GA is applied for generating representative test suites and mutation testing. A tool is built to demonstrate the proof of the concept. Mutation testing became easy with the generation of test suite that covers all goals.

Chapter 4 - Automatic Discovery of Dependency Structures for Test Case Prioritization

In this chapter, a novel mechanism is proposed to discover dependency structures from SUT automatically and use them for prioritization of test cases. This work is related to that of Haidry and Miller. However, they did not automate the discovery of dependency structures. Dependencies are of two types namely direct and indirect. Both types are considered in this study. A prototype application is built that demonstrates the proof of the concept. The empirical results reveal that the automatic discovery of dependency structures can help in complete automation of test case prioritization.

Chapter 5 - Comprehensive Testing Tool for Automatic Test Suite Generation, Prioritization and Testing of Object Oriented Software Products

This chapter deals with the concept of generation, representative test suite, which ensures complete code coverage. Coverage criteria play an important role in automatic test case generation. Traditional approaches targeted one particular coverage goal. It is evident that the optimization of whole test suite generation is far
better than the traditional method of targeting one coverage goal from the recent experiments in software testing. Genetic algorithms have been applied successfully to generate unit tests for testing object oriented software. This chapter also covers the test case prioritization and its utility in improving the rate of fault detection in SUT. As test suite contains many test cases, the order of their execution plays an important role in increasing the rate of fault detection which can provide early feedback to the development team so as to help them to improve the quality of software. Therefore, it is very useful to prioritize test cases that will lead to the increase in the rate of fault detection.

Chapter 6 – Conclusions and Future Work

This chapter covers the conclusions made from the work done in this research. From the wealth of experience and observations, directions are provided for possible future work. The conclusions are drawn on proposed algorithms aimed at fulfilling the objectives such as whole test suite generation and mutation testing, the automatic discovery of dependency structures and test case prioritization for increasing the fault detection ratio.