CHAPTER ONE
INTRODUCTION

1.1. The Evolve of Software testing
1.2. Problem definition
1.3. Objectives of the Study
1.4. Hypothesis
1.5. The Testing Criteria
1.6. Methodology
1.7. Organization of the Thesis
Chapter 1: Introduction

1.1. The Evolve of Software Testing

During the last 30 years, software engineering has evolved into a legitimate engineering discipline. Today, it is recognized as a subject worthy of serious research, continuous study, and tumultuous debate. Software engineer has replaced programmer as the job title of dereference. Software process models, software engineering methods, and software tools have been adopted successfully across a broad spectrum of industry applications [94].

Many individuals and companies are still developing software haphazardly, even as they build systems to services the most advanced technologies of the day. Many professionals and students are unaware of modern methods. As a result, the quality of the software that is produced suffers with errors and bad things happen [61,112].

Computer software has become a driving force in business domain for decision making. It serves as the bases for modern scientific investigation and engineering problem-solving. It is a key factor that differentiates modern
products and services. It is embedded in systems of all kinds: transportation, medical, telecommunication, military, industrial process, entertainment, office products …etc. Today software takes on a dual role, as a product and at the same time it is the vehicle for delivering a product. Building software is like building any successful product, by applying a process that leads to a high quality result that meets the needs of the people who will use the product.

The issue of determining software quality whether particular solutions to problems that involve a computer are, or not, accepted. An effective quality system leads to increased productivity and permanently reduced cost.

The International Standard Quality vocabulary (ISQ) defines quality as: ‘the totality of features and characteristics of a product or service that bear on its ability to meet stated or implied needs’. A software product displays quality to the extent that all aspects of customers’ needs are satisfied. A good software development process must enable the software organization to deliver quality product consistently and economically.

Quality measurement is usually expressed in terms of metrics. Software metric is a measurable property, which is an indicator of one or more of the quality criteria that we are seeking to measure.
The history of testing goes back to the beginning of the computing field [37, 48, 77, and 93]. The purpose of testing can be: quality assurance, verification and validation or reliability estimation. Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets the required results [88].

Testing is usually performed for the following purposes [88]:

- **To improve quality:** Test the software for being correct. The minimum requirement of quality, means performing as required under specified circumstances.

- **For verification and validation (V & V):** Testing can serve as metric. It is extensively used as V & V process.

- **For reliability estimation:** based on operational profile (an estimation of the relative frequency use of various inputs of the program), testing can serve as a statistical sampling method to gain failure data for reliability estimation.

The input data used to make a program demonstrates its feature called as test case or test data. During testing the software engineers produces a series of test cases that are used to “rip apart” the software they have produced. Each individual program is tested in a process called unit-testing.
Chapter 1: Introduction

The importance of software testing and its impact on software cannot be underestimated [103]. Software testing is a fundamental component of software quality assurance and represents a review of specifications, design and code. Testing is significant because software creators will appreciate having their oversights discovered, because managers will then feel more confident in the product, and since program testers can take pride on the software.

One of the main reasons for the release of low quality ‘buggy’ software is inadequate testing. [97].

Verifying software usually involves lot of investment, more than it takes to consider it as a regular activity in developing software. Testing usually requires a reasonable cost and investment in term of time and human resources. It can be performed directly on the system under test [80, 91].

Some definitions of software testing are [56]:

- Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements, or to identify differences between expected and actual results. (IEEE).
• The process of executing a program or system with the intent of finding errors. (Myers 1979).

• The measurement of software quality. (Hetzel 1983).

The field of software reliability is of course very closely allied to testing. The reliability estimate depends on testing being repetitive of the actual operational environment. Software reliability is closely related to the amount of testing performed.

Several levels of required software reliability can be defined for a software product [10, 81]:

1. Very low: the effect of software failure is simply the inconvenience incumbent on the developers to fix the fault.

2. Low: the effect of software failure is low level, easily recoverable loss to users, e.g. climate forecasting model.

3. Nominal: the effect of software failure is moderate loss to users. But a situation from which one can recover without extreme penalty, e.g. inventory control system.

4. High: the effect of software failure can be a major financial loss or a massive human inconvenience, e.g. banking systems and electric power distribution systems.
5. Very high: the effect of software failure can be loss of human life, e.g. Military Command Systems and Nuclear Reactor Control System.

One of the major difficulties in software testing is the automatic generation of test data that satisfies a given adequacy criterion. Because manually generating large number of test cases to fulfill the testing criteria which consumes too much time and effort.

A large number of test case design methods have been developed with systematic approach for testing [32, 78 and 122]. Automatic test data generation process is expected to significantly reduce the overall development cost of software product. It is the most important aspect of automatic testing. The use of meta-heuristic search techniques for automatic generation of test data has been a burgeoning interest of many researchers in the recent years [73]. The use of heuristic techniques such as genetic algorithms (GAs) can be useful in providing usable results in reasonable time.

1.2. Problem definition

This research was motivated by the rapid growth of software industry, where new measurement techniques and metrics are needed, for assessing the quality and reliability of software.
Chapter 1: Introduction

How to generate test data? How to evaluate them? These are the major questions that researchers in the area of automated software testing are trying to find answers for [63, 83 and 127].

In industry, test data selections generally a manual process, the responsibility for which usually falls on testers. However, this practice is extremely costly, difficult and laborious. Automation in this area has been limited [73]. Exhaustive enumeration of programs’ input is infeasible for any reasonable-size program. Similarly, random methods are unreliable and unlikely to exercise “deeper” features of software that are not exercised by simple chance.

In the early age of automation of software testing, most data generators were using gradient descent algorithms [62], however these algorithms were inefficient and time consuming and it could not escape from local optima in the space of the domain of possible Input Data [43]. These issues necessitate the need to investigate the suitability of meta-heuristic search algorithms e.g. simulated annealing, genetic algorithms and ant-colony optimization as better alternatives for developing test data generators. The use of meta-heuristic techniques such as genetic algorithms requires the definition of an objective function which rewards test data solutions on the bases of how close they were
to fulfilling the required test goal [73]. GA was chosen as testing method because in recent decades they have grown in popularity in optimization of engineering problems [121, 123].

1.3. Objectives of the Study:

Generating test data manually is labor-intensive and time-consuming process. This problem has motivated the researchers to create test data generator based on Genetic algorithm that can examine a program's structure and generate adequate test data automatically to traverse and cover all paths of software to reduce time and effort.

The aim of the work is to investigate the effectiveness of genetic algorithms over random testing and to automatically generate test data to traverse all paths of software. Another goal of the current work is to identify factors which improve the performance of genetic algorithm in generating test data.

This study presents an enhancement to the SIMILARITY fitness function introduced by [66]. Results obtained are compared with Random testing to assess the effectiveness and efficiency of the proposed algorithm. The thesis is focused on white box testing as it is more widely applied in software testing.
The proposed GA is also investigated for efficiency and effectiveness over random testing with experiments for different kinds of data types like integer values, floating values and characters. Further experiments were conducted to generate test data for programs with complex data structure like arrays and loops.

1.4. Hypothesis:

The following set of hypotheses are suggested and justified in the following chapters:

**Hyp (1.4.1):** GA requires less number of test data to achieve path coverage than random testing.

**Hyp (1.4.2):** considering the parameters and operators of the proposed GA to generate test data for path coverage:

i) Selection of parents for reproduction according to their fitness value is more efficient than random selection.

ii) Double-point crossover strategy is more efficient than single-point crossover.

iii) To get the best results, mutation rate is better adjusted with the problem at hand.
Hyp (1.4.3): the proposed GA improves from one generation to the next while searching for good test data, because of the feedback provided by fitness function value. While random testing generates less successful test cases and the search is absolutely random.

Hyp (1.4.4) test cases can be generated to test programs with inputs of different data types (integers, floats, characters) in either single form or array elements form.

Hyp (1.4.5) test case can be generated to test programs contains loops. The test is done for zero, one, two and more than two iterations.

1.5. The Testing Criteria:

It is almost impossible to say quantitively that how many faults are potentially uncovered by a given test [3, 46]. This is not only because of the faults in themselves, but because of the concept of “fault” is only vaguely defined. This has led to the development of test adequacy criteria.

A test criterion is the criterion that defines what constitutes an adequate test [127]. Structural coverage criteria can be used in the management and control of software testing. Test criteria help the tester to organize the test process. Criteria are used to design the test cases, test data selection and for code coverage measurement and analysis of test suites [50,108 and 129]. An
adequacy criterion is considered to be a stopping rule that determines whether sufficient testing has been done that it can be stopped. Furthermore, test data adequacy criteria provide measurement of test quality when a degree of adequacy is associated with each test set, in practice the percentage of code coverage is generally used as an adequacy measurement.

The criterion of testing in this work is path coverage. The aim is to develop test data to exercise every path of the software under test (SUT). The path coverage criterion is concerned with selected paths and it was adopted since it achieves the utmost coverage. Chapter 2 discusses the different test adequacy criteria and provides a justification for adopting path coverage as an effective criterion.

1.6. Methodology

The study pertains to propose new algorithm to automate test data generation for path testing and to determine the potential effectiveness of the proposed GA over random testing. The methodology used is an experimental approach to enhance the method used for automatic software test data generation. After studying the Genetic algorithms, their variation and applications it was intended to develop an algorithm that could automatically generate software test data for path testing in simple and fast manner.
The proposed GA based on SMS fitness function is developed in chapter 4, and then subjected to different types of experiments as follows:

- **Algorithms initial evaluation:**
  The performance of the proposed algorithm based on SMS as fitness function is traced and compared to the performance of a chosen GA based on SIMILARITY fitness function presented in [66]. The reason behind comparing SMS performance with SIMILARITY’s is that both fitness function use the distance calculated between any two given paths using the Extended Hamming distance (EHD). Both algorithms were applied on a famous bench mark program (Triangle-Classifier program).

- **The effect of the complexity of a program under test:**
  The proposed GA based on SMS as fitness functions applied to a suit of selected C programs to study the effect of the complexity of the program under test in the performance of SMS, since cyclomatic complexity is an important software quality and reliability metric.
  The selected object programs have different cyclomatic complexity (smaller to larger).
• **Studying GA parameters:**

Some additional experiments are conducted to have a better understanding of the operator and parameters of GAs.

Different parameter combinations of GA were studied and the performance of GA in the evolution of test data generation process is observed and recorded.

• **Studying the search progress in the proposed GA:**

Experiments have been conducted to study the generation-to-generation progress in GA, and the role of feedback from SMS that helps the GA to progress in the search from one generation to the next.

The experimental results on some sample programs are compared to random testing.

• **Study the performance with data types:**

The applicability of the proposed algorithm, with SMS as fitness function, to perform path testing on programs with different input data types (integer numbers, float numbers, characters) is examined and results are compared with random testing.
• **Studying loop testing:**

Finally some additional experiments are conducted to evaluate the performance of the proposed GA to achieve path coverage for programs with more complex structure, i.e. programs with loops and arrays. The tests conducted zero, one, two and more than two loops testing.

The results are also compared to random testing to assess the feasibility of the proposed algorithm in such type of testing.

In all the experiments conducted, control flow graph (CFG) of the targeted program is extracted then the list of goal paths is listed. It has been done manually, by instrumenting the source code of the sample program but it could be automated. The software probes are inserted at strategic points in the program. This can provide information pertaining to the quality of the coverage. To do this, several assertions to the code at different locations have been added. At the end of testing it is possible to check whether complete coverage was achieved or not. If not, more new test cases are generated and added, and the testing process repeats.

The test data, using the proposed algorithm, are then generated during a predefined number of generations with specified parameters. The evolutionary generation takes place according to the *fitness* of the test cases. The resulting
information, collected during each test execution of the program, is used to heuristically determine how close the test data has came to satisfy the test requirement i.e. covering the selected goal path. This allows the algorithm to modify the programs input gradually, to bring them closer to the value that actually satisfies the requirement.

In each experiment, the obtained results give clear know-how about the performance of the algorithm on the object program under test. These results include: the list of successfully covered paths, the successful test data that covered a goal path with coverage frequency, the generation number at which the specific path was covered and the list of uncovered paths if any.

The same groups of experiments are conducted with randomly generated data. To do fair comparison, each algorithm (genetic algorithm and random testing) are allowed the same resources. Results of both algorithms are then compared and analyzed.

1.7. Organization of the Thesis:

The thesis consists of six chapters and organized as introduction of software testing, background and methodology, genetic algorithms for test data
Chapter 1: Introduction

generation, the proposed algorithm, experiments and results, and concludes with conclusion and suggestions.

Chapter two gives a brief introduction of software testing. The stepwise testing strategies (unit, integration, system, acceptance and regression testing) have been provided. Testing techniques and test adequacy criteria have been presented. It also gives description of automatic test methods and presents random test data generation. Thereafter, it defines the quality and reliability of a software, software complexity metrics and measures, and software testing life cycle (STLC).

Chapter three gives an overview of basic concept of evolutionary testing using genetic algorithms and discusses the parameters, operators and performance of the algorithm. How to use GA in software testing is also explained. Finally a brief overview of the related work done in this area is expounded.

Chapter four presents the proposed algorithm for automatic test data generation. The fitness function is also developed. The chapter also illustrates the method to apply the algorithm for path testing.
Chapter 1: Introduction

The results of the conducted experiments using the proposed algorithm on various programs are presented in chapter five and have been classified according to the hypothesis defined above.

Conclusion remarks of and future work is given in chapter six which shows the goodness of research results.