Value based Regression Test Case Prioritization using Evolutionary Algorithm

Developed by:

Erum Ashraf 301-MSSE/FBAS/F09

Supervised by:

Dr.Abdul Rauf

Department of Computer Science and Software Engineering
Faculty of Basic and Applied Sciences
International Islamic University Islamabad
(2011)

MS
005·3
ERV

1- Computer Software - Testing

Department of Computer Science and Software Engineering
International Islamic University Islamabad

Date: 02/03/2012

Final Approval

This is to certify that we have read the thesis submitted by **Erum Ashraf, registration# 301-MSSE/FBAS/F09**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by International Islamic University. Islamabad for the degree of **MSSE.**

Committee:

External Examiner:

Dr. Sajid Anwar

Assistant Professor

[illegible]

Internal Examiner:

Dr. ZuneraJalil

Acting Chairperson

[illegible]

Supervisor:

Dr.AbdulRauf

Assistant Professor

[illegible]

Dedicated to my family, my teachers and everyone who helped and prayed for my success.

A dissertation Submitted To

Department of Computer Science and Software Engineering,

Faculty of Basic and Applied Sciences,

International Islamic University, Islamabad

As a Partial Fulfillment of the Requirement for the Award of the

Degree of *MSSE*.

Declaration

We hereby declare that this Thesis ***"Value based Regression Test Case Prioritization using Evolutionary Algorithm"*** neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this research with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers especially our supervisor *Dr.Abdul Rauf.* If any part of the system is proved to be copied out from any source or found to be reproduction of any project from any of the training institute or educational institutions, we shall stand by the consequences.

[illegible]

Erum Ashraf

[Registration# 301-MSSE/FBAS/F09]

Acknowledgement

First of all I am obliged to Allah Almighty the Merciful, the Beneficent and the source of all Knowledge, for granting us the courage and knowledge to complete this Project.

I would not have reached this stage but for the prayers, love and moral support of my mother and father. I am greatly thankful to my supervisor Dr.Abdul Rauf, who provided me useful and helpful assistance. I would like to thank my son Muhammad Umar, my brothers, sisters, my friend uzma shaheen, my colleagues and teachers Dr Zunera Jalil, Ehsan Ahmed, Dr.Waseem Shehzad, Dr.ramzan and all other well-wishers.

I am thankful to my husband Khurrum Mahmood who supported and helped me throughout my research work.I am thankful to each and every person who helped and prayed for me.

[illegible]

Erum Ashraf

Registration# 301-MSSE/FBAS/F09

Project In Brief

Project Title:	Value based Regression Test Case Prioritization using Evolutionary Algorithm
Undertaken By:	Erum Ashraf
Supervised By:	Dr. Abdul Rauf
Start Date:	1st Mar, 2011
Completion Date:	24th Nov, 2011
Tools & Technologies	MATLAB 9.0
Documentation Tools	MS WORD 2007
Operating System:	Windows XP
System Used:	Pentium 4

Abstract

Regression testing is type of software testing that is done to uncover new software faults in existing functional areas of a system after changes. Common methods of regression testing include rerunning previously run tests and checking whether program behavior has changed. The execution of the complete set of test cases will require time and is a complex process, which may not be feasible in limited time to detect maximum faults at earlier stages. Adequate time should be dedicated for testing but due to some limitations it can't be so. Experience has shown that without proper prioritization of test cases, the end product usually fails to meet its objectives optimally. In fact in many instances, the product is considered a failure because it fails to meet its core objectives due to mainly shortened time pressure.

To reduce the effort required and to meet the time to market pressure, test cases are being reordered now. Several test case prioritization techniques have been presented by various researchers over the past years. But we have found a lack of work in integration of artificial intelligence algorithms for value based test case prioritization. In this thesis, we have presented a novel value based intelligent test case prioritization algorithm using particle swarm optimization. We have considered the criterion of maximum fault coverage in minimum execution time for test case prioritization. We have performed experimentation using our proposed algorithm and compared the results with existing technique. The experiments have shown that our proposed algorithm is capable of delivering impressive prioritization under varying and often conflicting circumstances.

Prioritization of test cases has been done by considering the factors found in literature. The weightage of any factor can be maximized according to the required criteria of prioritization of test cases. Therefore, our proposed strategy of prioritization of test cases is customizable and it gives much better results than random technique of prioritization. The proposed algorithm can search the best new positions of the test case which have high fault detection ability. Our proposed value based particle swarm optimization test case prioritization algorithm can discover reasonable quality solution effectively and efficiently. The proposed solution may have some biasness issue as many stakeholders are being involved in our proposed solution to get factor values of test cases. These values are validated from project managers to reduce this biasness factor.

List of Abbreviations

AI	Artificial Intelligence
AST	Automated Software Testing
EA	Evolutionary Algorithms
GA	Genetic Algorithm
KBSE	Knowledge Based Software Engineering
PSO	Particle Swarm Optimization
SE	Software Engineering

TABLE OF CONTENTS

TABLE OF CONTENTS -- X

LIST OF TABLES -- XIII

LIST OF FIGURES -- XIV

CHAPTER 1: INTRODUCTION -- 1

1.1 Regression Testing -- 1

1.2 Test case prioritization -- 1

1.3 Regression Test Case Prioritization -- 1

1.4 Motivation -- 2

1.5 Problem Statement -- 3

1.6 Goals -- 4

1.7 Objectives -- 4

1.8 Thesis Organization -- 5

CHAPTER 2: BACKGROUND STUDY -- 6

2.1 Introduction -- 6

2.2 Optimization Algorithms -- 6

2.3 A Classification of Optimization Algorithms -- 7

2.4 Optimization and search techniques -- 7

2.4.1 Traditional Optimiza on Algorithms -- 7

2.4.2 Gradient-Based Local Optimization Method -- 7

2.4.3 Random Search -- 8

2.4.4 Stochastic Algorithms -- 8

2.4.5 Evolutionary Algorithms ---------- 9

2.5 Value Based Software Engineering ---------- 9

2.6 Background ---------- 10

2.6.1 What is Value? ---------- 10

2.6.2 "Value" in Software Engineering ---------- 11

2.7 Foundations of VBSE ---------- 12

2.7.1 Theory Context ---------- 12

2.7.2 Theory W ---------- 12

2.8 Summary ---------- 13

CHAPTER 3: LITERATURE REVIEW ---------- 14

3.1 Test Case Prioritization ---------- 14

3.2 Test Case Prioritization Techniques ---------- 14

3.2.1 Coverage-based Test Case Prioritization techniques ---------- 14

3.2.2 Non coverage-based Test Case Prioritization ---------- 16

3.3 Summary ---------- 19

CHAPTER 4: PROPOSED STRATEGY ---------- 20

4.1 Value based Test Case Prioritization Factors ---------- 20

4.1.1 Customer Priority ---------- 20

4.1.2 Implementation Complexity ---------- 20

4.1.3 Requirement Volatility ---------- 21

4.1.4 Requirements Traceability ---------- 22

4.1.5 Execution Time ---------- 22

4.1.6 Fault Impact of Requirement ---------- 22

4.2 Factor Collection Process ---------- 22

4.3 Value Based Test Case Prioritization Block diagram ---------- 23

4.4 Fault Detection Analysis Using PSO ---------- 24

4.4:1 Swarm Initialization ---------- 24

4.4.2 Velocity Update ---------- 24

4.4.3 Position Update ---------- 25
4.4.4 Quality Measure---------- 25
4.4.5 Completion Criteria ---------- 25
4.4.6 Optimal Priority Check Equation---------- 26

4.5 Pseudo Code for Proposed Method ---------- 26

4.6 Summary ---------- 31

CHAPTER 5: RESULTS AND DISCUSSION ---------- 32

5.1 Introduction ---------- 32

5.2 Experimental Design ---------- 33

5.3 Project Description ---------- 35

5.4 Performance Measure---------- 37

5.5 Experimental Results ---------- 37
5.5.1 Project 1 Results ---------- 39
5.5.2 Project 2 Results ---------- 41
5.5.3 Project 3 Results ---------- 43

5.6 Summary ---------- 46

CHAPTER 6: CONCLUSION AND FUTURE WORK ---------- 47

6.1 Conclusion ---------- 47

6.2 Recommendations for Future Work---------- 47

REFERENCES---------- 48

INDEX 1 (RISK MANAGEMENT DATA) ---------- 54

INDEX 2 (IT PROCUREMENT SUITE DATA) ---------- 56

INDEX 3 (SALES INTRANET DATA) ---------- 57

LIST OF TABLES

Table 2.1 Theory W ____ 12

Table 4.1 Test case values I#1 ____ 27

Table 4.2 Fitness Function Results I#1 ____ 27

Table 4.3 Velocities and Positions I#1 ____ 28

Table 4.4 Test Case Values I#2 ____ 28

Table 4.5 Fitness Function Results I#2 ____ 29

Table 4.6 Velocities and Positions I#2 ____ 29

Table 4.7 Test Case Values I#3 ____ 30

Table 4.8 Fitness Function Results I#3 ____ 30

Table 4.9 Velocities and Positions I#3 ____ 31

Table 5.1 Simple Design Table ____ 34

Table 5.2 Project Description ____ 35

Table 5.3 Data Collected from Stakeholder ____ 36

Table 5.4 Parameters used for Project 1 ____ 38

Table 5.5 Parameters used for Project 2 ____ 38

Table 5.6 Parameters used for Project 3 ____ 38

Table 5.7 Summary of Results ____ 46

LIST OF FIGURES

Figure 2.1 Theory W 12
Figure 4.1 Block Diagram of PSO for VBTCP 23
Figure 4.2 Particle Representation 24
Figure 5.1 Comparison of Results for P#1 39
Figure 5.2 Random Technique Results P#1 39
Figure 5.3 PSO Results P#1 40
Figure 5.4 Comparison Bar Chart for P#1 40
Figure 5.5 Comparisons of Results for P#2 41
Figure 5.6 Random Technique Results P#2 42
Figure 5.7 PSO Results P#2 42
Figure 5.8 Comparison Bar Chart for P#2 43
Figure 5.9 Comparisons of Results P#3 43
Figure 5.10 Random Technique Results P#3 44
Figure 5.11 PSO Results P#3 44
Figure 5.12 Bar Chart Comparison for P#3 45

Chapter 1 Introduction

1.1 Regression Testing

Regression testing is the process of having assurance regarding any modification in the software. It makes sure the working of functional characteristics of software after having modifications in software. It is quite expensive technique to be used. Some techniques such as test selection, test prioritization [4] has been proposed by researchers for effective cost reduction in regression testing. Regression testing is a part of software development process. It is the process of testing the changes made to the software, by executing a set of test cases. The major task in this process is to fix the bugs and ensure that the issues are resolved. Any software may require upgrade or change with time, but a small or large change may disrupt the performance of the new software with old functions or data. Regression testing is needed in order to ensure that software is working as it was before changes. It is a control measure to maintain good quality of the software. Also the old test cases are run against the new version to validate that all the previous capabilities still work.

1.2 Test case prioritization

Test case prioritization can be performed by meeting some predefined criteria. This criterion can be maximum fault detection, reduction of cost or maximum code coverage. Selection of test cases from test suite is not a wise option when high quality software is required. Another feasible option is the prioritization of the test cases, so that limited number of test cases can be run to check the functionality of the software in available time.

1.3 Regression Test Case Prioritization

Basically there are two methods to carry out regression testing. The first method involves testing of entire system by re-executing all test cases, but this is a very lengthy process and a complex process. As regression testing is quite expensive process so it is usually not possible (due to limited resources) to see the effects of changes, every time the change is made by re executing all test cases. There is another strategy for this purpose which is executing regression testing by reordering the test cases in a specific order to meet few performance aims, such as coverage etc. To do this, the test cases which give more coverage will be executed first. This strategy is called as regression test case prioritization.

1.4 Motivation

Efficient AI systems can be designed by integrating AI and SE disciplines together. Advantages of AI applications can be seen in situations where complex decisions are needed to be taken. AI knowledge can be integrated into computer science field to deliver more efficient and correct systems to its customers. Software testing tools can benefit from such knowledge. An active research is being happening for this purpose.

This thesis also puts an effort in test case prioritization using AI specifically evolutionary algorithms. So that it is the connection of two entirely different research fields, test case prioritization and evolutionary algorithms. Considering the significance of testing, and the maturity of evolutionary algorithmic methods, we believe the time has come for the software testing and AI researchers to join forces in assisting software testers in testing of software systems.

Software testing is difficult process as customers want to be delivered thoroughly tested system in given constraints. There are different hurdles involve in this process such as complexity of system, shortage of resources humans and otherwise, less mature testing process etc. It gets often impossible to meet testing goals while using traditionally manual testing methods. Automated Software Testing (AST) helps software testers to address these challenges by reducing the time and cost of software testing, AST can help software testers to improve software quality by enhancing the manual testing efforts via increased testing coverage and labour intensive tasks can also be replaced with it. Automated systems are used to ensure reliable system, to increase quality of testing effort, to reduce manual testing effort and to minimize the testing schedule.

Exclusive research is being done to propose efficient solutions for difficulties of regression testing. According to Myers [11], maximum test efficiency should be tried to achieve by increasing the identification of faults with a limited number of test cases. Fewster also gives almost same views about test efficacy. According to fewster, software testing should be done to identify faults at lower cost in terms of time and cost [23].

Regression testing search space as well as most test objects implies complex search space [29]. Regression test complexity can be resolved by using EA for search space optimization. The population-based stochastic nature of evolutionary algorithms makes them more appropriate to solve such complicated problems. EA are capable to find global optimal solution without being stuck in local optimum. Fault detection based prioritization can also be done by using Metaheuristics algorithms. [29].

A PSO is a population-based stochastic optimization algorithm that has been modeled after the simulation of the social behavior of bird flocks. It is a simpler and easily implementable algorithm and has fewer parameters to adjust for optimization. It has been successfully applied to solve a wide range of search based optimization problems [26, 27, 28]. Thus, due to its simplicity and efficiency in navigating large search spaces for optimal solutions, PSO is used in this research to develop efficient, robust and flexible algorithms to solve test case prioritization problem. Because test case prioritization is an important strategy in regression testing, more benefits can take from it by integrating it with value based agenda. Focus of this thesis is to combine the VBSE into test case prioritization using evolutionary algorithm.

1.5 Problem Statement

Testing software is hard and to give guarantee that system is well being tested is even more difficult. During testing it is difficult to execute all code in limited time. It is not possible to test system exhaustively to declare it fault free system [15].Testing is often done in time to market pressure and is supposed to test whole software in quality manner. Regression testing involves executing large size of test cases which is time consuming process [11].

It is not possible to test the software by executing all test cases for regression testing under time, quality and resource constraints for fault identification at early stage. Software testing gives an equal importance by giving same amount of time to all parts of software to test but it does not meet to business value because 80% of value often comes from its 20% of software.

Major constraint of testing is time. Time to market constraint pressurize the testers to test software as early as possible for in time release of software, but this ultimately rises the possibility of potential risks in software, on the other hand time slippage occurs for satisfactory quality assessment of software [30].

Test case prioritization is quite complex problem as it is closely related to regression test case selection problem. Regression test case selection problem can be modeled as set covering problem, which is a well-known NP-Hard problem [4]. This detail gives an insight into difficulty of the problem of Test Case Prioritization. The selected set of problem in this thesis is NP-hard.

To tackle with these challenges, concept of value has been introduced by giving more value to critical and major functionality according to stakeholder. As ultimate objective of software or any business is to increase the return of investment (ROI) so by introducing value in testing, testers can focus on important part of software to be tested to increase ROI. Overall testing performance can be increased by making investment decisions like defining coverage criteria or prioritizing tests to optimize the overall testing progress [30].

1.6 Goals

Our goal of prioritization is to increase the likelihood of revealing maximum faults earlier in the testing process. We have proposed to use PSO incorporated with value concept to achieve the test case prioritization.

1.7 Objectives

The primary objectives of this thesis can be summarized as follows:

- To solve the difficult problem of prioritization of test cases in regression testing by applying Particle Swarm Optimization.
- To develop an efficient value based algorithm using PSO.
- To analyze the effects of VALUE based test case prioritization in fault detection.
- To compare the findings of particle swarm optimization with random ordering for fault detection.

1.8 Thesis Organization

There are 6 chapters in this thesis. Introduction is given in Chapter 1. Chapter 2 focuses on optimization methods and gives a comprehensive overview of value based software engineering. In chapter 3 test case prioritization problem has been formally defined with the help of literature support and detail of different existing test case prioritization techniques has been presented. In chapter 4 test case prioritization factors, its structure and factor collection process is discussed. Chapter 5 is about implementation of proposed algorithm and results. These results are compared with random technique. The thesis is concluded in the last chapter.

Chapter 2 Background Study

2.1 Introduction

Optimization techniques are extensively applied in scientific, business, industrial manufacturing, resource allocation, scheduling, computer science or engineering discipline. A common person also seeks for optimization to achieve some certain goal in his/her practical life like a traveler may want to adopt some shortest path to reach somewhere. Manufacturer wants to produce reliable machine parts by designing efficient architecture and operation of their production process. Basic theory of optimization is to fairly allocate the scarce resources or assets. In terms of Mathematics, the minimization or maximization of a function keeping in view the constraints in its variables is known as optimization [74]. Research in the optimization field is very active and new optimization methods are being developed regularly

This chapter comprises on two sections. The first section is about optimization and optimization algorithms while the second section explains the concept of value in software engineering disciplines.

2.2 Optimization Algorithms

Optimization is mainly to search values for certain set of predefined parameters or variables that will ultimately optimize some objective function to certain constraints. Not all the optimization problems can be solved by some specific method [4]. There exists different algorithm for different type of problems. Selection of appropriate algorithm depends on nature of problem. The most feasible solution will solve the optimization problem more rapidly [74]. Optimization includes both maximization and minimization problems. Global optimization is the process of finding the global optimum solution. [82].

By nature optimization algorithms are iterative. Initial guess is needed to start them that is the most favorable values of the variables and generate a sequence of improved results until they reach a solution [74]. Optimization algorithms are guided by some objective functions [82]. The fastest optimization algorithms search only a local solution, a point at which the objective function is smaller than at all other feasible points in its surrounding area. They do not always find the best of all such minima, that is, the global solution [74]. Global optimization problems are generally very difficult and are categorized under the class of nonlinear programming (NLP) [74]. Global optimization algorithms are optimization algorithms that provide work for measures that prevent junction to local optima and increase the probability of finding a global optimum. [82].

2.3 A Classification of Optimization Algorithms

Generally, we can divide the optimization algorithms in two basic categories: deterministic and probabilistic algorithms. Heuristics are the functions or part of an optimization algorithm that uses the information currently gathered by the algorithm to help decide which one of a set of possible solutions is to be tested next. Heuristics are usually problem class dependent. A meta heuristics a problem solving method for very general classes. It combines objective functions or heuristics efficiently in an abstract way, usually without utilizing deeper insight into their structure, i. e., by treating them as black-box-procedures [82].

2.4 Optimization and search techniques

Following are the search and optimization techniques found normally in literature.

2.4.1 Traditional Optimization Algorithms

In traditional optimization algorithms precise methods are used to obtain the best result. It is based upon the idea that if solution of a problem exists than global best solution should also be found by the algorithm. Brute force search is one of the exact methods, in which every solution in the search space is tried for global optimal solution. Cost of brute force algorithm increases with the increase of search space. Therefore, brute force algorithms are not appropriate for NP-hard problems. The time to exhaustively search an NP-hard problem increases exponentially with problem size. [81]

2.4.2 Gradient-Based Local Optimization Method

Gradient based methods are also known as Hessian based optimization methods. These methods are used to achieve efficient local optimization in the presence of smooth objective function. Performance and reliability of these methods can be improved by using them with some other optimization methods. [4]

2.4.3 Random Search

Random search is a simple and fundamental technique. It selects the solutions randomly from the search space and evaluates the fitness value of the solution that is selected. This is relatively an obtuse strategy, and is infrequently used by itself. It is easier to implement it, and considerable number of assessments can be done quite speedily. For new unresolved and vague problems, it can be useful to compare the results of a more advanced algorithm to random search for the same number of assessments. A random search never gets trapped in any point such as a local optimum. Hypothetically random search is assured to reach the optimal solution for finite search space. [4]

2.4.4 Stochastic Algorithms

Near optimal solutions are found by using stochastic search algorithms NP-hard problems in polynomial time. This algorithm is based on the idea that good solutions are closer to each other in the search space. This is a good assumption for largely real world problems. It is not necessary that stochastic algorithms may always find a global optimal solution. In other algorithms a solution is generated after the run is completed, whereas in stochastic algorithm the best solution found during the run can be found by stopping the run. Stochastic search algorithms are easier to implement. These algorithms are suitable for discrete and combinatorial problems to find optimal or near-optimal solutions. They can also be efficiently used in a multiprocessor environment. Three major stochastic algorithms are Hill-Climbing, Simulated Annealing and Tabu search[81, 82]

Hill Climbing

In Hill-Climbing, randomly a candidate solution is picked by assuming it as a potential solution. Then this solution is compared with its neighboring solutions. If the surrounding solution is found to be better according to some fitness criteria then the new solution is considered to be potential solution. This methodology is continued till the solution is constant means that there is no more significant improvement in the solution. [82].

Simulated annealing

In Simulated annealing the algorithm is started by picking a potential solution randomly. In the next iteration the new solution is obtained by adding a small value in the previous solution. The new solution is tested by the fitness criteria and if found better than the previous solution, then it becomes the current solution. Otherwise, the solution will move to the new location with a probability that reduces as the run increases. The simulated annealing behaves like a hill climbing but with the possibility of going downhill to evade being trapped at local optima [4].

2.4.5 Evolutionary Algorithms

EAs are population-based metaheuristic optimization algorithms that use biology-inspired system like mutation, crossover, natural selection, and endurance of the fittest to process a set of solution candidates iteratively [4]. Evolutionary algorithms have benefit of their black box characteristic over other optimization algorithms because it makes very few assumptions about objective function and requires less insight to structure of problem space. These are the reasons which make EAs to perform consistently in many different problems. Evolutionary algorithms copy the behavior of natural evolution and take solution candidates as individuals that contend in a virtual environment [4].Hill climbing and simulated annealing consider only one candidate solution as potential solution while evolutionary algorithm maintain population of potential solutions [81].

2.5 Value Based Software Engineering

Software engineering is not a new field. Extensive search is being done in this field in the last few decades. Presently, one of the features of this field is its equal stress on each and every extraneous aspect. It can be said that majority of the research and practice of the current software engineering a value-neutral settings.

Previously, when software decisions had quite minor influences on a system's cost, schedule, and value, the value neutral approach was rationally effective. But today and progressively more in the future, software has a major influence on most systems' cost, schedule, and value [80].

If we talk about engineering discipline the value-neutral methods are insufficient for it. The definition of "engineering" in [31] is "the application of science and mathematics by which the properties of matter and sources of energy in nature are made useful to people." It is hard for a value-neutral approach to provide guidance for making its products useful to people, as this involves dealing with different people's utility functions or value suggestions.

A lot of research is being progressing over the years to integrate some value-oriented perception into software engineering. In mid-nineties, Straut Faulk and his colleagues were pioneers to propose Value Based Software Engineering. Later Barry Boehm and colleagues laid down the theoretical fundamentals, agenda and application areas for Value Based Software Engineering.

2.6 Background

The methodology of introduction of value in literature is very important. Detail regarding certain methodologies of value and valuation techniques being applied in current era of software development is given in this section.

2.6.1 What is Value?

The definitive purpose of any business is to earn profit by delivering such kinds of product to its customer which will add value to existing worth of stakeholder. Same is true for software as software is also designed for the same purpose to satisfy its customers by producing more beneficial product or service.

It is important to understand value in order to maximize the objectives of all stakeholders.

Merriam-Webster online dictionary define value as, a fair return or equivalent in goods, services, or money for something exchanged or the monetary worth of something or relative worth, utility, or importance or a numerical quantity that is assigned or is determined by calculation or measurement [32].

Dictionary of Canadian Economics defines value as: "The quantity of one product or service that will be given or accepted in exchange for another" [33]. This definition although elucidate value pretty well for almost all classical products or services, it is unable to define the value of software products or services that well.

According to Oxford Companion to Law and it states that"...value may consist of spiritual or aesthetic qualities or in utility in use, or in the amount of money or other goods which could be obtained in exchange for the thing in question..."[34].

Dictionary of Sociology defines value as a"...generalized principle of behavior to which the members of a group feel a strong commitment and which provides a standard for judging specific acts and goals" [35]. This definition is unable to define the value with respect to software products or services.

All these definitions describe value in terms of money as relative unit. Value is defined in context of business not particularly for software. Moreover stakeholders are not being considered to establish value for product. In context of Value Based Software Engineering, value is relative worth, importance and utility. Engineering is not only making a product or delivering a service it also incorporates the purpose behind them. Therefore in engineering value is considered and so also in software engineering. But majority of the software engineering activities are taking place in value neutral settings. This results in diverting from the most critical stakeholder and features, it merely becomes delivery of product at any cost and any quality. So VBSE is a discipline that considers these stakeholder concerns in delivering any product or artifact [36].

2.6.2 "Value" in Software Engineering

The relationship between software engineering and value is quite obvious. According to Stefan Biffl et al the definitive purpose of software engineering is adding value to the existing conditions through forming products, services and processes [37]. This whole process can have negative impact in absence of value considerations explicitly. Following is the history of development of concept of value in software engineering.

In software engineering different cost models have been described by value. Boehm was the first one who described the concept of value away from cost models namely Boehm's software engineering economics [38]. After establishing relationship between value and software, Boehm introduced spiral model in 1986. McTaggart's further worked on it and presented value as very significant fundamental which was given name as value based management movement [39]. After this management movement Favaro in 1996 presented an essay titled as "When the Pursuit of Quality Destroys Value" to argue that focusing on merely quality in some cases can weaken the value of the product [40].Favaro et al. has also used value to address economics of software reuse [41]. In 1998 Boehm et al. has proposed winwin model which was basically to deal with the concept of requirement negotiations [43]. Formally in 2003 Boehm et al. has put forward the formal agenda of Value Based Software Engineering. [42].

2.7 Foundations of VBSE

In the coming sections a detailed context of the value-based software engineering theory and the concept of VBSE theory is elaborated.

2.7.1 Theory Context

A VBSE theory needs to address all of the concerns essential for developing and evolving successful software-intensive systems. This includes not only managerial aspects of software engineering but also to cater personal, cultural, and economic values for successful software. The theory of VBSE rests on these foundations.

2.7.2 Theory W

The main proponent of VBSE is Barry Boehm who introduced this concept [44] in 1989. Popularly known as Win-Win model [85], this technique relies heavily on negotiation to resolve any conflicts of opinion among various stakeholders. The negotiations are conducted in such a way that each stakeholder is in a "Win" situation. This technique is conducted on progress based predefined plan, risk assessment and risk handling. In this technique, users are asked to rank their requirements before actual negotiations start. Users are asked to carefully categorize which requirements they are willing to negotiate and which they are not. Theory W has been an active area of research among researchers which has been applied in not only requirement engineering but also in other domains of software engineering. Theory W is a major constituent of Value Based Software Engineering (VBSE) agenda and principle as well. Theory W is sometimes also known as four point agenda. The Theory W establishes a set of win-win conditions. The four point agenda is given below:

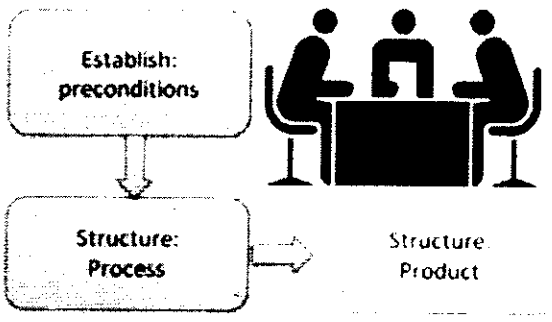

Figure 2.1 Theory W

2.8 Summary

The concept of introducing value in software engineering domain is quite critical but often ignored area. However in recent times, its significance has been realized and a lot of effort has been made to deliver a system fulfilling stakeholder's perspective. There are several existing optimization algorithms to solve complex problems such as GA, PSO, and ACO. In the next chapters, we shall discuss our proposed test case prioritization strategy for test case prioritization.

Chapter 3 Literature Review

3.1 Test Case Prioritization

Test case prioritization techniques are used to reorder the test cases to achieve some performance goal [8, 84].This chapter gives a general idea of test case prioritization methodologies found in literature.

Test case prioritization is a very essential practice of software testing which is usually neglected. Several test case prioritization techniques have been presented by authors.

The technique of obtaining a sequence of the test cases that achieves the required aim faster of an available test suite is known as test-case prioritization. Rothermel et al. [8] defines test-case prioritization problem as below:

Given: T, a test-suite; PT, the set of permutations of T; f a function from PT to the real numbers.

Problem: Find T0 2 PT such that (8T00)(T00 2 PT)(T00 6= T0)[f(T0) _ f(T00)].

PT represents all possible orderings of T, and f is a function that yields an award value for any given ordering it is applied to. f represents the goal of the prioritization. For example, the goal might be to reach a certain coverage criterion as fast as possible, or to improve the rate at which faults are detected. There are different test-case prioritization techniques that can be used to achieve such goals.

3.2 Test Case Prioritization Techniques

Mainly test case prioritization techniques can be classified in two categories. These are coverage based techniques and non-coverage based techniques for test case prioritization. Detail of these techniques from literature is as follows.

3.2.1 Coverage-based Test Case Prioritization techniques

Coverage-based TCP techniques [8, 29, 79] comprise the test cases ordering which are reliant on the code coverage that is provided by the test cases. Ordering of test cases are dependent on the number of statements covered by the test case that is, more lines of code test performs, the earlier it is executed in the test cycle. Code coverage techniques also include branch coverage techniques and function coverage techniques. Branch and coverage techniques are the techniques to prioritize tests are done on the basis of coverage of the program branches or program function.

Rothermal et al. [8] has investigated coverage based prioritization by examining a wide range of prioritization techniques for specific objective function to give insight into trade off among these techniques for test case prioritization. This work has been conducted for early rate of fault detection on general rather than modified version specific prioritization. To measure the efficiency of methods for fault detection, average percentage of fault detection (APFD) metric has been used. FEP-based techniques outperformed coverage-based techniques; however the total increase in APFD was not significant. These results run opposing to preliminary perception and suggest that given their expense, FEP-based prioritization may not be as cost-effective as coverage-based techniques. Randomly prioritized test suites outperformed untreated test suites. These results suggest that these techniques can improve the rate of fault detection of test suites.

Li et al. [29] proposed a technique for prioritization of test cases for code coverage. This technique includes the statement coverage, block coverage and decision coverage. An experiment has been conduct to compare greedy, metaheuristics and evolutionary search algorithm for test case prioritization and to figure out the factors that affects the effectiveness of algorithms for prioritization of the test case regarding regression testing. Experiment has been performed on six programs, the primary criteria that is used is the size and coverage criteria of test suits. Results indicate that size of the program does not but the size of the test suite directly affects test case prioritization complexity because it determines the size of the search space. Results suggest that for larger suit of regression test case prioritization, global search techniques perform much better than local search techniques.

Harman used coverage based metrics, which gives high value of coverage effectiveness to those test cases which cover test requirements more quickly. As test cases which give high coverage are more likely to determine program faults as compared to those with low coverage [29], so test cases are reordered according to improved rate of requirement coverage. Proposed coverage based metrics is different than average percentage of block coverage (APBC) metrics which is also coverage based metric which reorders the test cases according to how quickly it covers the blocks inside program. This metric prevents the requirement for fault seeding but it overlooked the test case running cost and so it may imprecisely differentiate efficiency.

Regression testing can also be used to reduce the occurrence and persistence of residual defects [75]. An experiment has been conducted to compare original and heuristic techniques to see the effectiveness of prioritization in reducing occurrence and age of residual defects. He has used one control (representative of current practice) and two heuristics techniques (total coverage prioritization, additional coverage prioritization having feedback mechanism in addition). It concludes that heuristic techniques are better than original technique in reducing the occurrence of residual defects as heuristic techniques do not show any pattern in detecting residual defects and reveal less residual defects in all versions while original technique describes higher residual defect value in earlier version and very less in later versions.

3.2.2 Non coverage-based Test Case Prioritization

Fazlalizadah has proposed an innovative equation for the prioritization of test cases in test suite for early fault detection in time constraint environment. Mainly three factors are contributing in the proposed equation 1) Priority of the test case in previous regression test session, 2) Historical demonstrated performance in fault detection during the regression test lifeline, and 3) Duration of not execution for each test case. Results are validated through an experiment on eight C programs and by case study. The results are compared with random technique. APFD metric has been used to measure the detected faults and proved it more effective way to prioritize the test cases in detection of faults under time constraint.

To obtain early fault detection effectively, model based test case prioritization has been used in literature [68]. Its idea is to use model dependence analysis to identify different ways in which marked transitions interact with the remaining parts of the model. This information is used to prioritize high priority tests. This paper focuses on EFSM system model. Six methods of prioritization has been used namely random prioritization, selective prioritization, model dependence based prioritization, heuristic no.1 prioritization, heuristic no.2 prioritization and heuristic no.3 prioritization. An experiment has been performed by focusing the source code faults. RP (d), the most likely relative position of the first failed test that detects fault, as the measure of effectiveness of early fault detection is used. It is deduced that on average some model based tests prioritization methods may improve the effectiveness of early fault detection as compared to random prioritization These results may suggest that only the number of execution of marked transitions may not have a significance influence on the improvement of the early fault detection.

APFD metric has been widely used for test case prioritization which measures the detected faults by using some technique. But this metric assumes that test cost and fault severity are uniform. But this is not the case forever. Techniques that prioritize test cases by using fault based metrics could fail to generate satisfactory results if test cost and fault severity vary widely [72]. To address this problem cost cognizant metric APFDc has been introduced to measure test case cost and fault severities to evaluate various test case orders. Results of proposed APFDc metric have been analyzed for three practical heuristics (additional statement coverage, additional functional coverage, additional fault index prioritization) and one experimental control (random technique). A case study is being performed to show the effect of test case cost and fault severity distributions on the rate of fault detection as measured by APFDc. The differences between the new and old metrics are also explained.

Genetic algorithm is meta heuristic approach and used for test case prioritization as a regression technique under time constrained which is based on coverage information (block and method) [70]. Effectiveness of Genetic algorithm has been compared using APFD values with initial, reverse of initial test suit ordering, random and fault aware prioritization and found it most effective in terms of rate of fault detection. It was required for each test case to be independent from other test cases to maximize the fault detection ability. An experiment and two case studies has been conducted to evaluate effectiveness of parameterized genetic algorithms and to compare it with other mentioned techniques. On average block coverage outperformed method coverage in relation to APFD while not increasing time overhead of test suit prioritization. Higher APFD values of GA are found than random prioritization. Genetic algorithm prioritization were improved in reverse ordering and showed up to 120% improved than initial ordering.

The concept of ILP (integer linear programming) has also been used for test case prioritization. Two GA and four ILP based techniques are compared for test case prioritization. Experimental results show that ILP based techniques are better than all other techniques for rate of fault detection in general and version specific prioritization. ILP based techniques are more time efficient than GA based techniques. Unlike ILP techniques, the analysis time of GA based techniques get increases under less tight time budget [71].

Kaur et al has proposed hybrid PSO algorithm for the prioritization of test case for regression testing in order to obtain maximum fault coverage in minimum execution time. PSO has been used with GA to generate diversity in population. In each iteration random test case is selected and added in all test cases. Velocities and positions of test cases are updated if it is better than previous one otherwise previously updated velocity and positions are recorded APFD metric has been used to asses' effectiveness of proposed algorithm and it showed its efficacy up to 75.6% for fault coverage [84].

PSO is optimization technique of swarm intelligence paradigm. Hla et al. has obtained best possible ordering of test cases using PSO in modified software units [76]. Existing test case priorities are supposed to correspond to velocities and fitness of test cases. These values are considered to be the prioritization parameters in this system. Twenty test cases from JUnit test suite are used in experiment. Change in position velocity vector is found by the new values of fitness of test cases. Experiment is performed over 100 runs. Due to randomized weighting factor, false positive rate is 8.2% under 100 runs that shows results are promising. Application of test case prioritization technique gives 64% of coverage in just 10 runs of test cases. By placing the test cases randomly, only 47% of test case coverage was achieved after running 10 test cases. Effectiveness of PSO algorithms is measured by comparing with greedy algorithm. Total prioritization cost and run time complexity of this algorithm is found less than greedy algorithm [O (mn square)].

Bayesian network (BN) approach is also proposed to prioritize the test cases [83]. In this paper BN approach has been modified by adding feedback mechanism and new change information gathering strategy. The impact of various variables on BN approach is observed in this paper. An empirical study on five java objects indicates the effectiveness of feedback mechanism of BN approach in terms of early fault detection. Moreover cost and benefit tradeoff is also provided depending on various parameters used in approach.

Fayoumi et al. [77] has proposed the algorithm (OptiTest) to envisage the modeling of unit test for object oriented source code. Ant Colony Optimization (ACO) and Rough Set Theory concepts are presented to find best quality test case. this approach use method call, passing arguments and control flow dependency graphs and a hybrid novel framework is proposed by inspiring natural ant. According to proposed algorithm OptiTest, the distribution and search of best test case value has been done through Ant colony pheromone matrix and once the search of best test value is achieved, the search terminates through Rough set. Rough set is used as stopping criteria rule in proposed model.

Prioritized test cases those are aiming to reduce the fault detection effort, also minimizes the information needed to locate the fault in the program. In the result debugging cost gets increase. So it is a big challenge to reduce the quality assurance cost which includes both the testing and debugging cost while minimizing the loss of diagnostic fault information [78]. SFL (Spectrum-based Fault Localization) technique has been used for fault diagnosis. An experiment has been conducted on Siemens set (composed of seven programs having test case inputs and ensure full code coverage) and showed this approach has reduced overall 53% of QA cost. SFL technique performed better than previous techniques because it uses online prioritization in which order of test case has to be dependent on output of previous tests. It is shown by an example that test cases whose aim is to cover many statements does not provide much information needed to diagnostic algorithm. Author has proposed the on-line greedy diagnostic prioritization approach that uses the observed test outcome to determine the next test case. In this approach high utility tests would be those tests which will maximize the reduction of diagnostic cost at each step on average. Reduction of diagnostic cost will ultimately increase the diagnostic information. Experiment is performed in permanent fault setting which is not very common in software as diagnostic approach needs prior information.

3.3 Summary

Test case prioritization is not a new field in software testing. Many researchers have presented different ways of ordering the test cases focusing on different criteria such as code coverage or maximum fault detection rate. This chapter has summarized the work previously done in this field. In next chapter we will define our proposed strategy for test case prioritization for maximum rate of fault detection.

Chapter 4 Proposed Strategy

4.1 Value based Test Case Prioritization Factors

Time and budget constraints usually don't allow executing all test cases. Therefore, project manager can use prioritization as a tool to help him in selection of those test cases which are more fault revealing or which are meeting some other defined criteria. This chapter presents the test case prioritization scheme. Test case prioritization involves seven prioritization factors. These factors are discussed in detail.

Test cases should be prioritized objectively; i.e., there must be some parameters that shall be used to assign values to each test case. Following are some important parameters for prioritization:

4.1.1 Customer Priority

Customer-assigned priority (CP) denotes the significance of a requirement to the customer. For each requirement a value is assigned by the customer that ranges from 1 to 10. Highest customer priority is denoted by 10 [45, 47].

Reasoning: Obviously, testing requirement priorities and test case costs should have a great impact on the test case prioritization [46]. Approximately 36% of the software functions are only constantly used, while 19% are only often used and the rest percentage is not used at all i.e. 45 % [54]. Frequent failures are caused by the fault that is situated along the course of regular execution, and greater effort must be made to detect such kind of faults [57, 58]. Customer-perceived value and satisfaction can be increased by giving priority to the customer requirements for development [53, 54, 55]. Identification and more thoroughly testing the highest important fraction of requirement to the customer sooner in testing can raise the business value. If the efforts for testing were reduced because of schedule demands, the requirements of highest value to the customer would have been tested early and exhaustively [47].

4.1.2 Implementation Complexity

Developer-perceived implementation complexity refers to individual measure of amount of difficulty perceived by the development of the requirement by the development team.

Analysis of every requirement is made to evaluate the estimated implementation complexity. It is given a value between 1 to 10; smaller value shows lower complexity while the greater value shows higher complexity [45, 47].

Reasoning: A number of studies show higher number of faults are present on that requirements that have high complexity in its implementation. Amland [52] carried out an investigation to determine that the functions with greater McCabe complexity is those with h high number of faults [52]. From the total system 20 % modules of the system resulted in 80% of the faults [52, 59, 60, 61], and approximately there was no fault in 50% of the modules [59]. This was shown by Don ONeill from National Software Quality Experiment on a DoD project which had roughly one million source lines of code [59, 47].

4.1.3 Requirement Volatility

In literature requirement volatility (RV) is adapted as one of most important prioritization factor [45, 46, 47]. Requirements volatility is measured as the number of times the development cycle of a requirement has been changed with respect to when the requirement was initially introduced. It is basically a judgment of the requirements change with respect to its start date. It also ranges from 1 to 10 [45, 47].

Reasoning: Approximately 50% of the total faults discovered in a project comprise of those errors that are introduced in requirement phase [62]. Rigorous defects that deliver to the customer costs hundred times greater averagely to resolve as compared to resolving the same problem in the requirements time [59].

Standish Group has conducted many studies. It has deduced that 70% of the total projects are unable to provide the obligator functionality of the system whereas 30% are cancelled before completion. Changing requirements is the most important factor to cause these project failures [63]. Some studies show that lack of user input can also be the cause for project failures cans, and volatile or deficient requirements [65, 62].

On average approximately 25% to 40% of the requirements changes before completion of project [64]. The changing requirements cause the testing activities to be complicated and ground the software to have high fault bulk [66]. Volatile requirements result in re-design, and an increase in the program's fault density [66].

4.1.4 Requirements Traceability

The relationship between various artifacts in a software development process like requirements, design and test cases is known as traceability [48].

Reasoning: The quality of the software can be improved by considering the traceability of the requirement [21].

4.1.5 Execution Time

Test case costs should have a great impact on the test case prioritization. In terms of test case cost, it is related to the resources, such as execution time of test case, hardware costs or even engineers' salaries. In literature many authors have considered execution time of test case as test case cost [45, 46, 49, 50].

4.1.6 Fault Impact of Requirement

Fault proneness (FP) of requirements is the identification of the requirements that have the most failures in the previous version by the development team [46].

Reasoning: The test efficiency can be enhanced by concentrating on the functionalities that have higher number of faults has been shown by Ostrand[67, 45].

4.2 Factor Collection Process

There are four stakeholders in this process. Roles of these stakeholders are defined below.

Developer

- To provide system requirements during development
- To provide the priority for the each requirement during development
- To provide any changes to the requirements during development

Requirement Analyst

- To record the requirements and related priorities
- To record any variations to requirements

Maintenance Engineer

- To resolve the field failures defects

- To links the failure back to the requirements impacted
- To write test cases for each requirement
- To map the requirement to its test case
- To run the test cases

4.3 Value Based Test Case Prioritization Block diagram

We can have clear understanding of working of value based Particle Swarm Optimization (PSO) with the help of following block diagram.

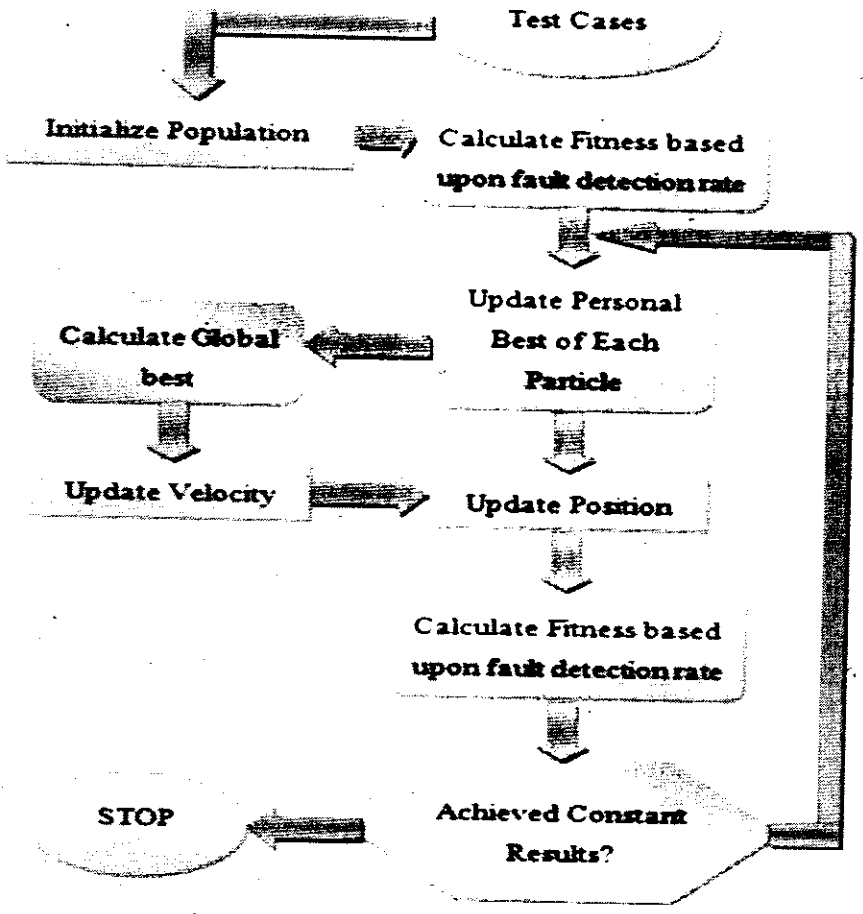
Test Cases
Initialize Population
Calculate Fitness based upon fault detection rate
Update Personal Best of Each Particle
Calculate Global best
Update Velocity
Update Position
Calculate Fitness based upon fault detection rate
Achieved Constant Results?
STOP

Figure 4.1 Block Diagram of PSO for VBTCP

4.4 Fault Detection Analysis Using PSO

Following is the detailed description of how our proposed algorithm works for test case prioritization problem.

4.4.1 Swarm Initialization

The number of particles in our scheme is equal to the number of test cases. Each particle represents a test case. Particle consists of 6 values. Each particle's position in our scheme represents the priority of the test case to be executed.

Generate random population of n particles. A random particle in our approach is depicted as follow:-

8	4	3	6	4	5

Figure 4.2 Particle Representation

4.4.2 Velocity Update

The velocity of each particle is updated according to the following equation.

$$v_i(t) = W \times v_i(t-1) + c_1 \times r_1(x_i^{pb} - x_i(t)) + c_2 \times r_2(x_i^{gb} - x_i(t)) \text{ ---- } \mathbf{(4.1)}$$

Where:-

V_i (t) = Velocity of particle at current iteration

V_i (t-1) = Velocity of particle at previous iteration

W = Inertia factor

C_1 = Self confidence of particle

C_2 = Society confidence

r1 = Constant

r2 = Constant

4.4.3 Position Update

The standard equation of PSO for position update has not been used. The main reason for this is that this problem is prioritization problem; therefore each particle has to have a position in terms of its priority. The idea is used that the particle whose velocity is the least will be given highest priority, this is due to the fact that that particle will be close to the optimum point. Similarly, the particle that has the highest velocity will be given the least priority. Chapter 5 depicts that this concept of position update in prioritization problem gives good results.

4.4.4 Quality Measure

Given an input program, the fitness function returns a number whose value indicates the factors that are to be optimized for the current sequence. Following is our fitness function

$$F = \frac{\Sigma C - \Sigma E}{n} \quad \text{---- (4.2)}$$

where:-

C denotes the summation values of the factors of a test case which are to be maximized

E denotes the summation values of the factors of a test case which are to be minimized

$\sum$C = sum of C of the test cases that have been executed

$\sum$E = sum of E of the test cases that have been executed

n = the position in which the test case is being executed

The updated fitness values of the test cases help in finding the position change vector, velocity change vector. The rate of velocity is used to change the current positions of the test cases to the new positions.

4.4.5 Completion Criteria

Following are the types of completion criteria:-

Completion of maximum iterations

Swarm global fitness shows no improvement for successive iterations

4.4.6 Optimal Priority Check Equation

$$Value = \sum_{i=0}^{n} (Fault\ Detection\ by\ ith\ priority\ Test\ Case)\ X\ (Total\ Number\ of\ Test\ Cases - i) \quad \text{--- } \mathbf{(4.3)}$$

The above equation maximizes if the test case with most fault detection are played prior to the respective test cases, this is because as the priority of the test case is decreased the multiplying factor (i.e. Total Number of Test cases – i) decreases.

In the end of each iteration of PSO the above equation is checked and compared to the previous value of the equation, in our algorithm this equation improves and becomes constant that shows that the optimal results have been obtained and therefore is used as a cross check to the PSO Algorithm Performance.

4.5 Pseudo Code for Proposed Method

To analyze the fault detection rate, particle swarm optimization algorithm has been used with minor adjustments. Each particle represents the test case consisting of factor values in our proposed algorithm. Following is the pseudo code for our proposed method:

Start: -Random generation of population of n particles.

Fitness Evaluation: -Fitness evaluation f(x) of each particle x in the population. We have calculated fitness of particle based upon the fault detection analysis.

Following is the example for illustration of working of our proposed algorithm. For the sake of easiness, we assume inertia w = 0.1; and c1r1= c2r2 =0.2. We have taken five particles, p1, p2, p3, p4, p5 and fitness of each particle is calculated.

ITERATION 1

We are assuming that we have only five test cases for prioritization. In start we execute these test cases sequential. The sum of maximizing factors (customer priority, requirement traceability and fault impact of requirement) and minimizing factors (implementation complexity, requirement volatility and execution time) of these test cases are given below.

Order	Test Case No.	∑Maximizing Factors	∑Minimizing Factors
1	TC1	2	6
2	TC2	9	5
3	TC3	4	2
4	TC4	6	4
5	TC5	2	3

Table 4.1 Test case values I#1

Fitness function results of these test cases are given below.

Test Case Number	Fitness Function Results
TC1 executed first	-4.00
TC2 executed secondly	0.00
TC3 executed thirdly	0.67
TC4 executed fourthly	1.00
TC5 executed fifthly	0.60

Table 4.2 Fitness Function Results I#1

Velocities and positions are calculated by using their standard equations in below table. Test cases having more velocities will be assigned lesser positions to execute at that point.

Test Case Number	Velocities	Positions based on velocities
TC1	1.4000	5
TC2	0.6000	4
TC3	0.5333	2
TC4	0.5000	1
TC5	0.54	3

Table 4.3 Velocities and Positions I#1

ITERATION 2

In second iteration again we have same values of maximizing and minimizing factors of test cases described below but with different execution order.

Order	Test Case No.	∑Maximizing Factors	∑Minimizing Factors
5	TC1	2	6
4	TC2	9	5
2	TC3	4	2
1	TC4	6	4
3	TC5	2	3

Table 4.4 Test Case Values I#2

Then again we have calculated fitness function of these test cases mentioned in table 4.5.

Test Case Number	Fitness Function Results
TC1 executed fifthly	0.6000
TC2 executed fourthly	1.7500
TC3 executed secondly	2.0000
TC4 executed first	2.0000
TC5 executed thirdly	1.0000

Table 4.5 Fitness Function Results I#2

TH 7584

Velocities and positions are updated. More velocity means that, that particular particle needs more velocity to reach the optimal point so we mark it less position than others.

Test Case Number	Velocities	Positions based on velocities
TC1	0.3200	4
TC2	0.1917	1
TC3	0.2000	2
TC4	0.4667	5
TC5	0.2778	3

Table 4.6 Velocities and Positions I#2

ITERATION 3

Again we have maximizing and minimizing factor values of test cases with updated sequence in below table.

Order	Test Case No.	∑Maximizing Factors	∑Minimizing Factors
4	TC1	2	6
1	TC2	9	5
2	TC3	4	2
5	TC4	6	4
3	TC5	2	3

Table 4.7 Test Case Values I#3

Fitness function values are calculated.

Test Case Number	Fitness Function Results
TC1 executed fourthly	0.2500
TC2 executed first	4.0000
TC3 executed secondly	3.0000
TC4 executed fifthly	0.6000
TC5 executed thirdly	1.6667

Table 4.8 Fitness Function Results I#3

Velocities and positions are updated.

Test Case Number	Velocities	Positions based on velocities
TC1	0.5729	5
TC2	0.1167	1
TC3	0.1500	2
TC4	0.4200	4
TC5	0.234	3

Table 4.9 Velocities and Positions I#3

In further iterations the results will be constant as all the test cases have find their personal best positions; they will just go towards the global best as close as they could.

4.6 Summary

In this chapter we have discussed our proposed strategy in detail. The designed fitness function has been explained with the help of example. The prioritization factors and the process of collection of the values from different have been explained as well.

Chapter 5 Results and Discussion

5.1 Introduction

In most of the situations, due to budget and time constraints, it becomes impossible to test software system exhaustively. In these scenarios, we need test case prioritization. We can prioritize test case to realize which test case may urgently be executed by meeting some predefined constrained to fulfill time to market pressure. We have found it very important to prioritize test cases in their true sense in order to deploy a quality and successful product. Test case prioritization was a new practice in our specific testing environment. So, the nature of our work required us to study further into various test case prioritization techniques so that we can select one which can best suit our peculiar testing environment. The main hindrances faced by testers while testing the software system are related to cost and time.

In order to overcome these problems, one solution was to develop an artificially intelligent expert driven test case prioritization technique. This work is inspired from "value based requirements prioritization" technique [25]. This technique was very much similar to Theory W. In this technique the end users and experts were asked to prioritize their requirements based upon the value that accomplishment of this requirement may have for the system. The salient feature of this technique was an amalgamation of end users and experts in the process of requirement prioritization. However, while implementing test case prioritization technique, we encountered one major problem. The technique was completely manual. The prioritization was done through human endeavor and element of human bias was noticeable.

In this chapter, we present and elaborate upon a PSO based intelligent test case prioritization technique. This technique uses PSO to prioritize test cases ranked by various stakeholders. This modified scheme is basically a single level prioritization where development team gives value to requirement and test cases according to mentioned factors and then intelligent system performs test case prioritization. In order to apply the utility of intelligent test case prioritization technique, we applied this as well as a representative random technique on several projects and determined the degree of success.

In this section, we will explain general idea of PSO technique and its use for test case prioritization.

5.2 Experimental Design

The motivation behind using experiment as a validation tool is that we can create a controlled environment in which an application can be tested. Secondly experiments are suitable for validation of applications or techniques. We will start the experiment by falsifying the null hypothesis.

Hypothesis:

1. The proposed algorithm is able to generate ordering of test cases that can detect maximum faults in the application.
2. The proposed work is extendable to support future work.

Null Hypothesis:

1. The proposed algorithm is not able to generate ordering of test cases that can detect all faults in the application.
2. The proposed work is not extendable to support future development.

Treatment:

The algorithm to reorder test cases through PSO.

Experimental Design:

Simple Design

Experiment Operation:

1. The experiment will be executed in following steps.
2. An application will be created.
3. Test cases will be reordered using the application.
4. The results of the execution will be analyzed.

Technique	Projects		
PSO	Sales Intranet	IT Procurement Suite	Risk Management
Random	Sales Intranet	IT Procurement Suite	Risk Management

Table 5.1 Simple Design Table

Experimental Steps:

1. Problem identification
2. Formulate hypothesis.
3. Application Development.
4. Execution of the application.
5. Comparing the results of test case reordering by applying it on industrial project.

Experiment objects: The test case reordering will act as an experiment object.

Independent Variable: prioritize technique will be independent variable during test case prioritization

Dependent Variable: The dependent variables in our experiment will be rate of fault detection.

Control Variable: project complexity, size.

Internal Validity: Biasness of factor values provided by different stakeholders. These values will be validated from project expertise like project manager etc. This will increase our confidence on results attained.

External Validity: object program representativeness will be external threat to our findings. The experiment will perform on mid-size projects so we cannot generalize the results gathered from our experiment. Repeating the experiment at different complex projects will ensures that the results are due to our technique used rather due to the fatigue of continuous being involved in it.

Construct Validity: The possibility of difference in costs of faults and test cases are not accounted by APFD.

The algorithm for fault detection analysis has been implemented in MATLAB. The algorithm has been applied on various projects to determine its effectiveness. Three industrial projects were selected to experiment with.

5.3 Project Description

The detail of these projects is as follows.

Project Description	Project 1	Project 2	Project 3
Project name	Sales Intranet	ITProcurement Suite	Risk Management
Company Name	Ovex technologies	Ikonomi	Ovex technologies
Nature of Project	Web based	Web based	desktop
No. Of modules	14	9	23
No. Of Test Cases	40	21	47
Complexity level	Medium	Medium	Medium
Team size	9	6	5

Table 5.2 Project Description

<u>Project Name: ITProcurement Suite</u>

Description: It is a web based solution that helps buyer to post online projects, bidders submit bids against buyer's project, buyer selects one sellers bid and assign contract to them, both can communicate and interchange files etc. buyer made payment to the seller by payment module. Three most important modules of this application include buy or make decision, bid no bid decision and seller evaluation module.

Project Name: Risk Management

Description: I.T. Projects are some of the riskiest projects executed today. This application is windows based that helps to manage risks of any type of project that can be either web based or desktop based, small or large. It helps project managers in minimizing the risks associated with project by risk mitigation methods and understanding the impact of risk to customers and develop plans to identify their risk tolerance.

Project Name: Sales Intranet

Description: The main purpose of this project is to provide an interface to the Sales Team to process Sales Quotes/Orders, Create New Customers and Track Sales Order Status. Other interface is provided to the purchase team for purchasing software and hardware from different vendors, credit team processes invoices, customers can submit online orders and check the status of their orders. Data management team manages inventory and price profiles into the system. Development team was asked to provide values of these projects for following factors.

Factors	Values	Stakeholders
Customer Priority	1- 10	Developer
Implementation Complexity	1- 10	Developer
Requirement Volatility	1- 10	Business Analyst
Requirement Traceability	1- 10	Maintenance Engineer
Execution Time	1- 10 sec	Developer
Fault Impact of requirement	1-5	Test Engineer

Table 5.3 Data Collected from Stakeholder

5.4 Performance Measure

To quantify the goal of increasing a subset of the test suite's rate of fault detection, APFD metric has been used. This metric is developed by Elbaum et al. [29] that measures the average rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the number of faults detected during the run of the test suite.

Formula:

$$\text{APFD} = 1 - \frac{(TF1+TF2+ \dots +TFm)}{nm} + \frac{1}{2xn}$$

Where,

T -> The test suite under evaluation

m -> the number of faults contained in the program under test P

n -> The total number of test cases and

TFi -> The position of the first test in T that exposes fault i.

5.5 Experimental Results

Experimental results show that the proposed algorithm was able to achieve more fault detection rate than the random technique. Furthermore it is depicted by fault detection rate there is still room for improvement. However, achieving such a high fault detection rate proves the competiveness of our technique as compared to other existing approaches. Following table shows the details of parameters used in experimentation during testing each project.

Table 5.4 Parameters used for Project 1

Parameters	Values
Population size	40
Number of iterations	30
Termination criteria	Constant results or iterations=30

Table 5.5 Parameters used for Project 2

Parameters	Values
Population size	21
Number of iterations	30
Termination criteria	Constant results or iterations=30

Table 5.6 Parameters used for Project 3

Parameters	Values
Population size	47
Number of iterations	30
Termination criteria	Constant results or iterations=30

5.5.1 Project 1 Results

Following graph depicts the fault rate comparison between PSO and random techniques.

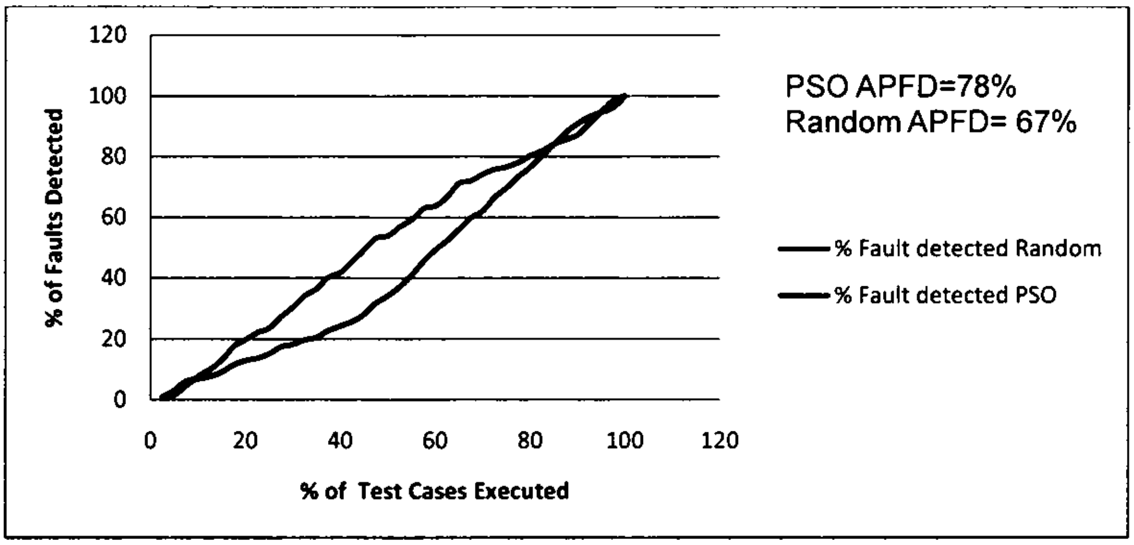

Figure 5.1 Comparison of Results for P#1

We can see that after executing 40% test cases we obtained 42 % fault detection rate through PSO and 24% fault were detected through random technique. Furthermore we have also validated our results through APFD metric. APFD calculation results shows that PSO detects 78% faults while random ordering produces 67% of faults which again shows significance of our findings.

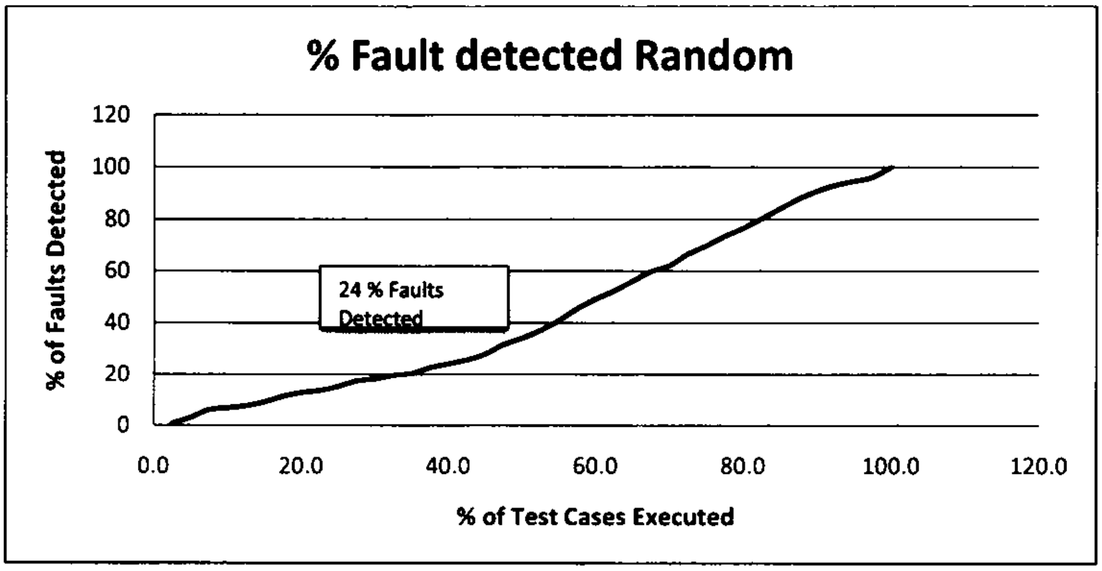

Figure 5.2 Random Technique Results P#1

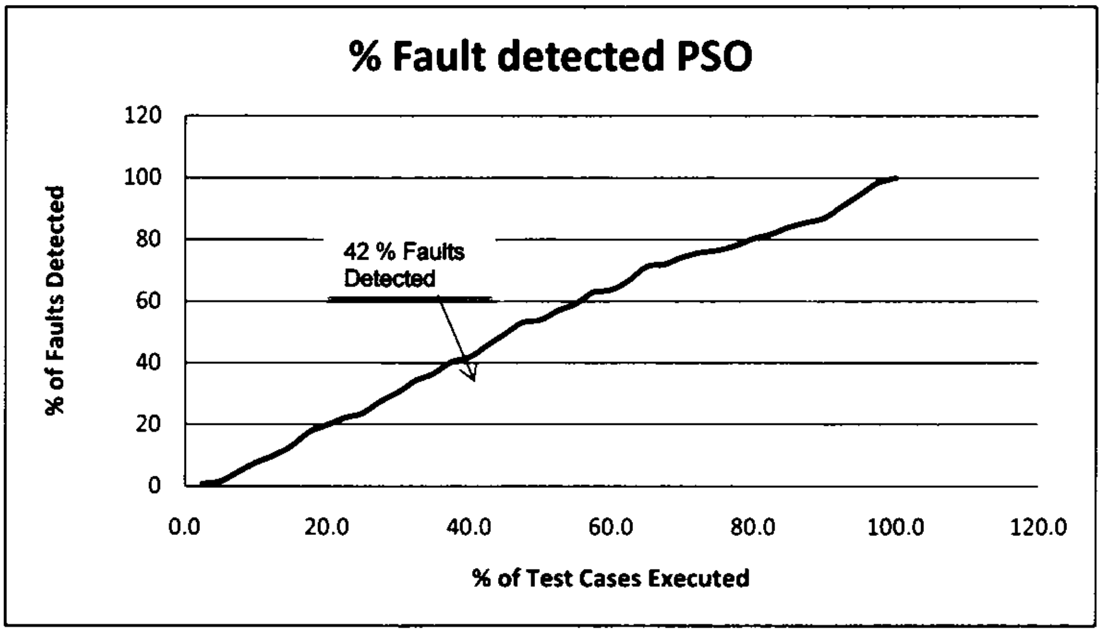

Figure 5.3 PSO Results P#1

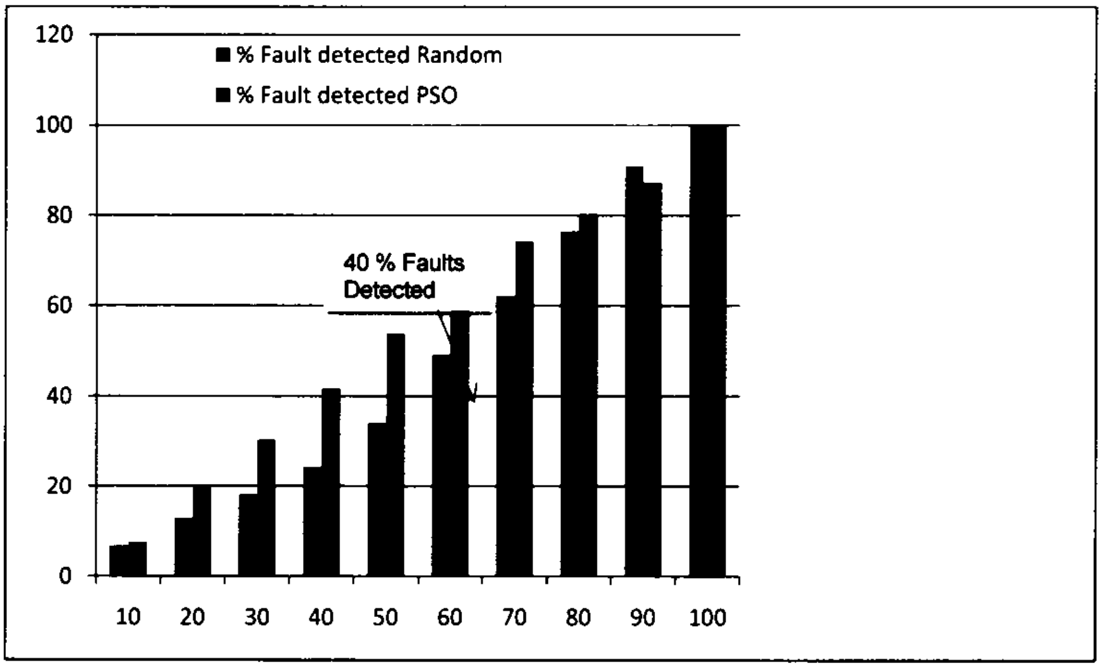

Figure 5.4 Comparison Bar Chart for P#1

5.5.2 Project 2 Results

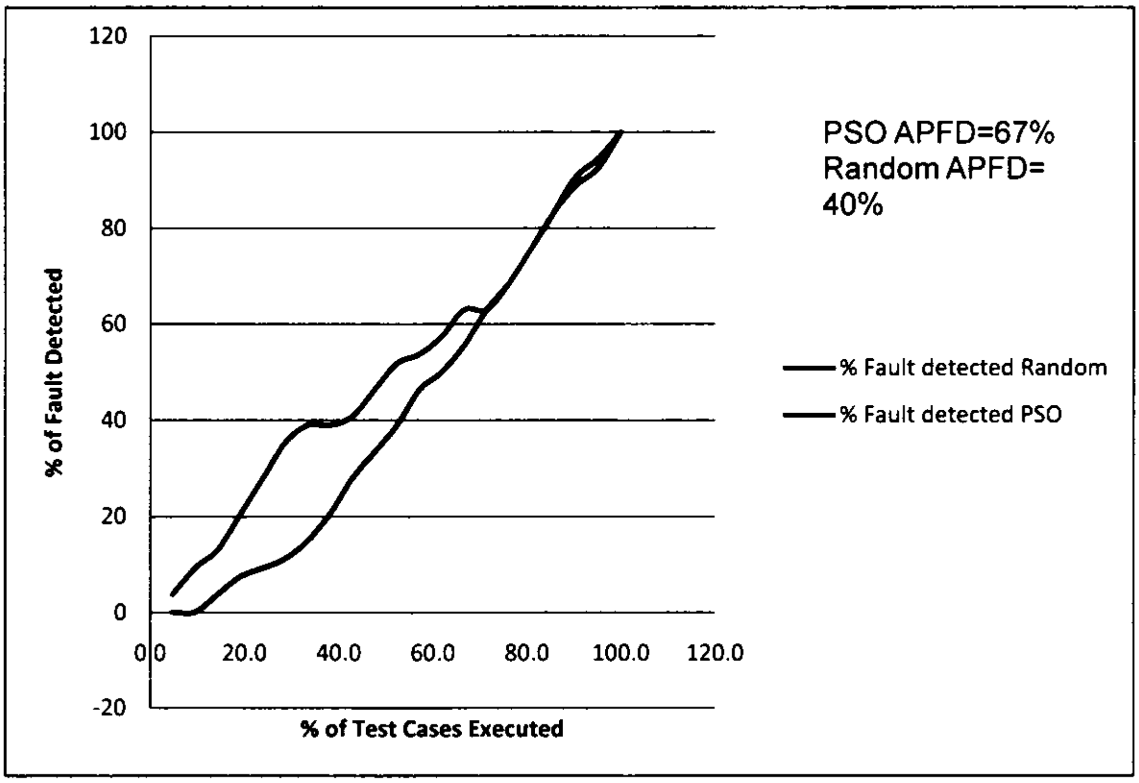

Figure 5.5 Comparisons of Results for P#2

We can see that after executing 40% test cases we obtained 39 % fault detection rate through PSO and 20% fault were detected through random technique. Furthermore we have also validated our results through APFD metric. APFD calculation results shows that PSO detects 67% faults while random ordering produces 40% of faults which again shows significance of our findings.

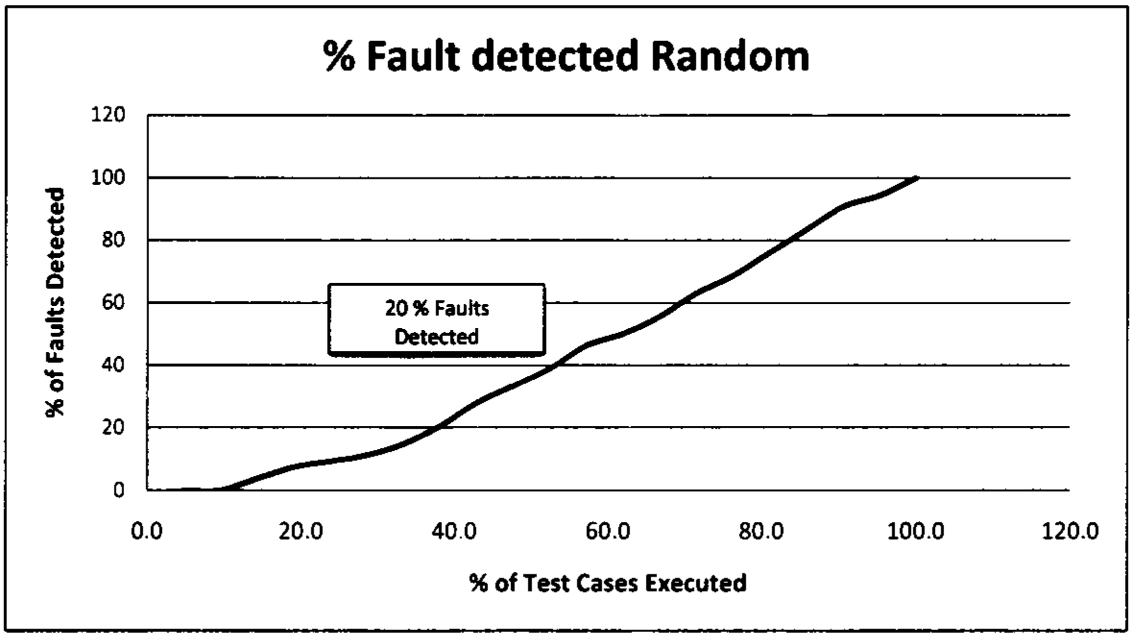

Figure 5.6 Random Technique Results P#2

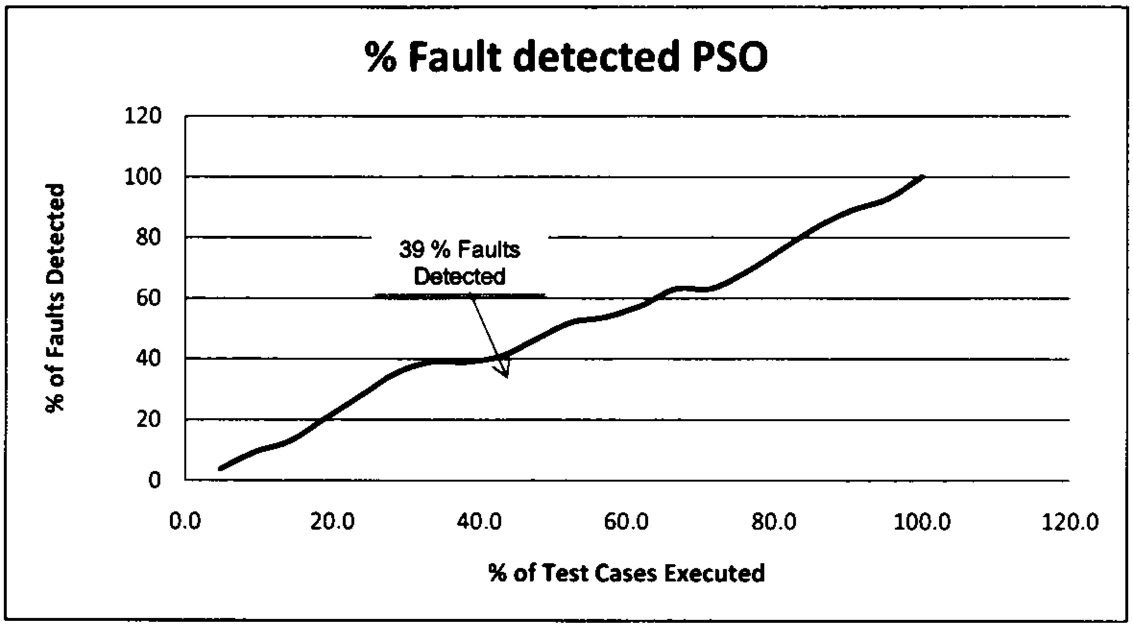

Figure 5.7 PSO Results P#2

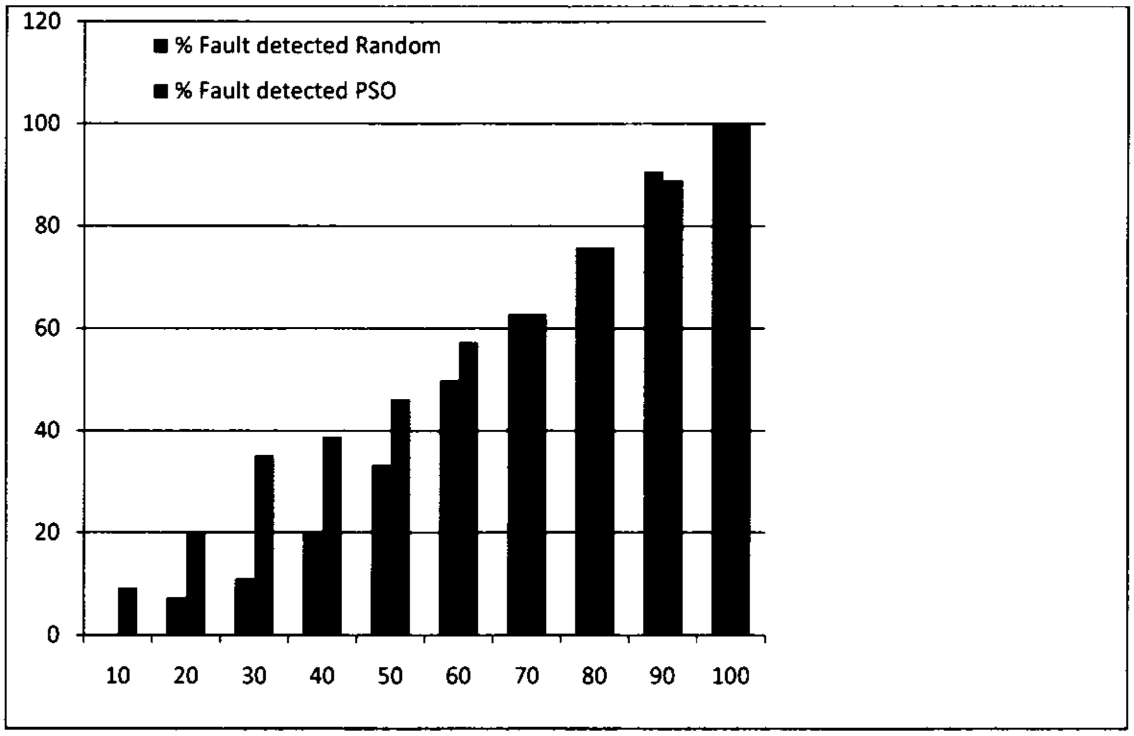

Figure 5.8 Comparison Bar Chart for P#2

5.5.3 Project 3 Results

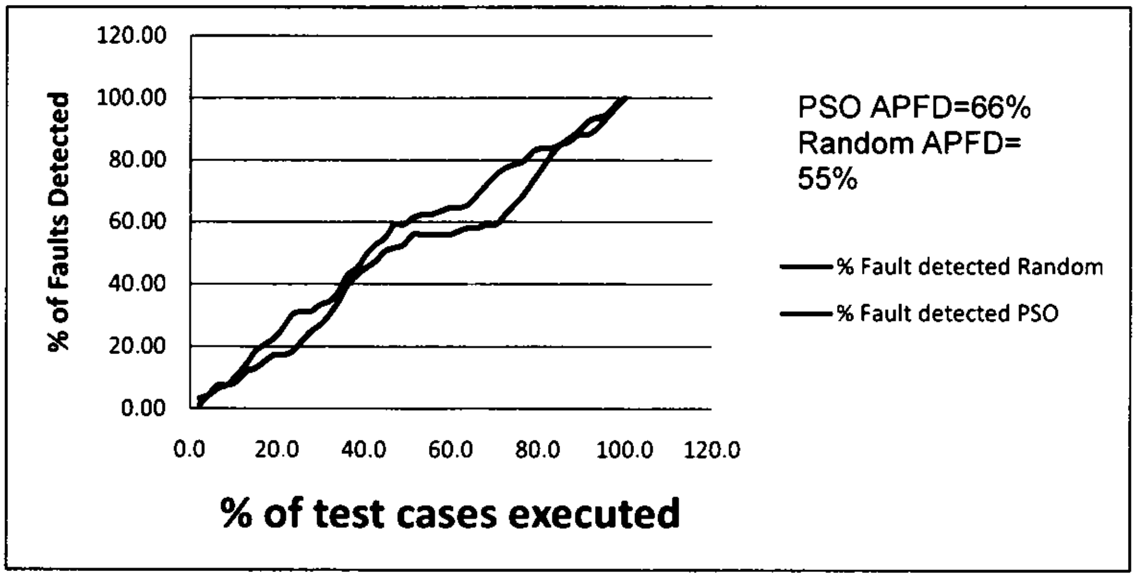

Figure 5.9 Comparisons of Results P#3

75 % fault detection rate has been obtained while executing 40% of test cases through PSO and 59% fault were detected through random technique. We have also validated our results through APFD metric. APFD calculation results shows that PSO detects 66% faults while random ordering produces 55% of faults which again shows significance of our findings.

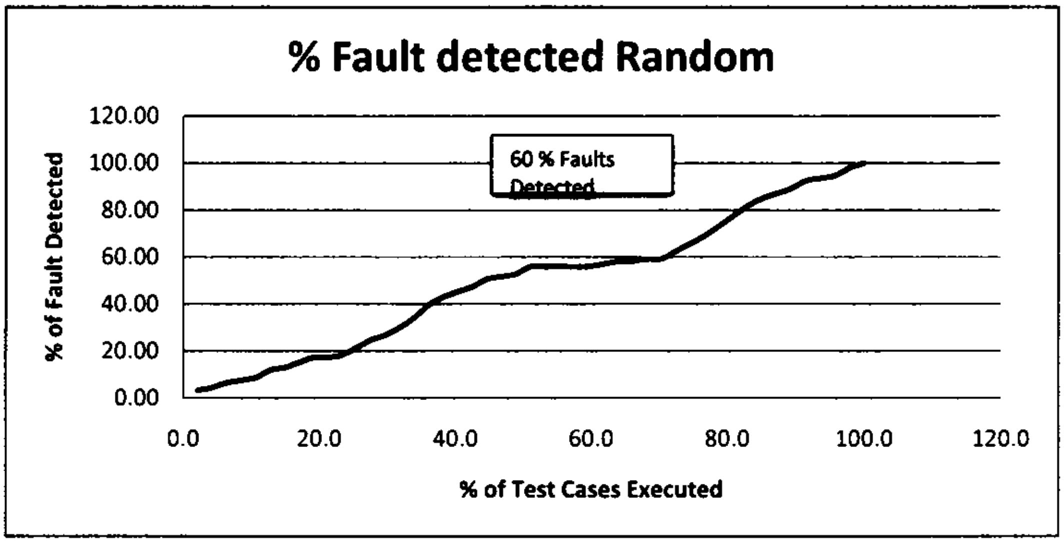

Figure 5.10 Random Technique Results P#3

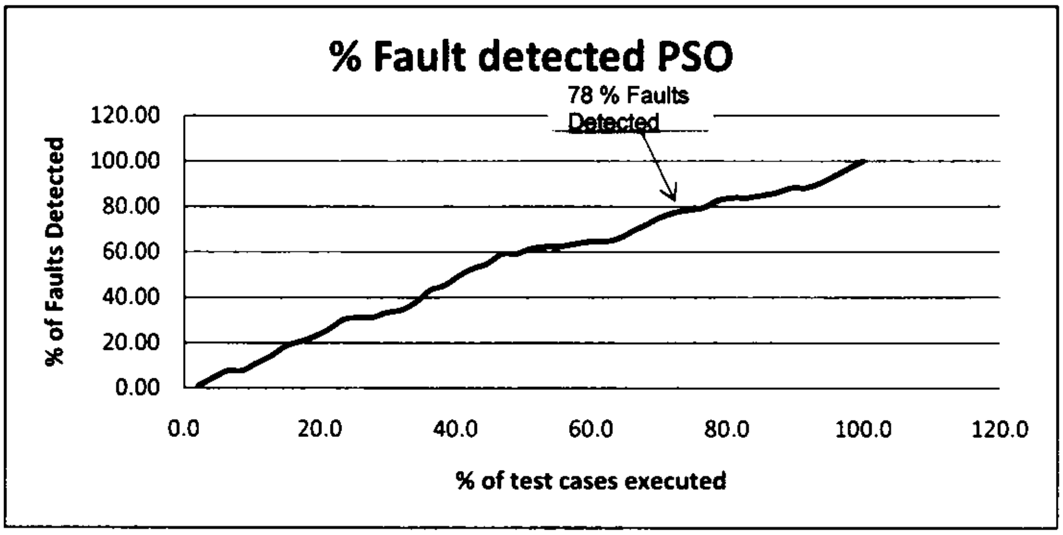

Figure 5.11 PSO Results P#3

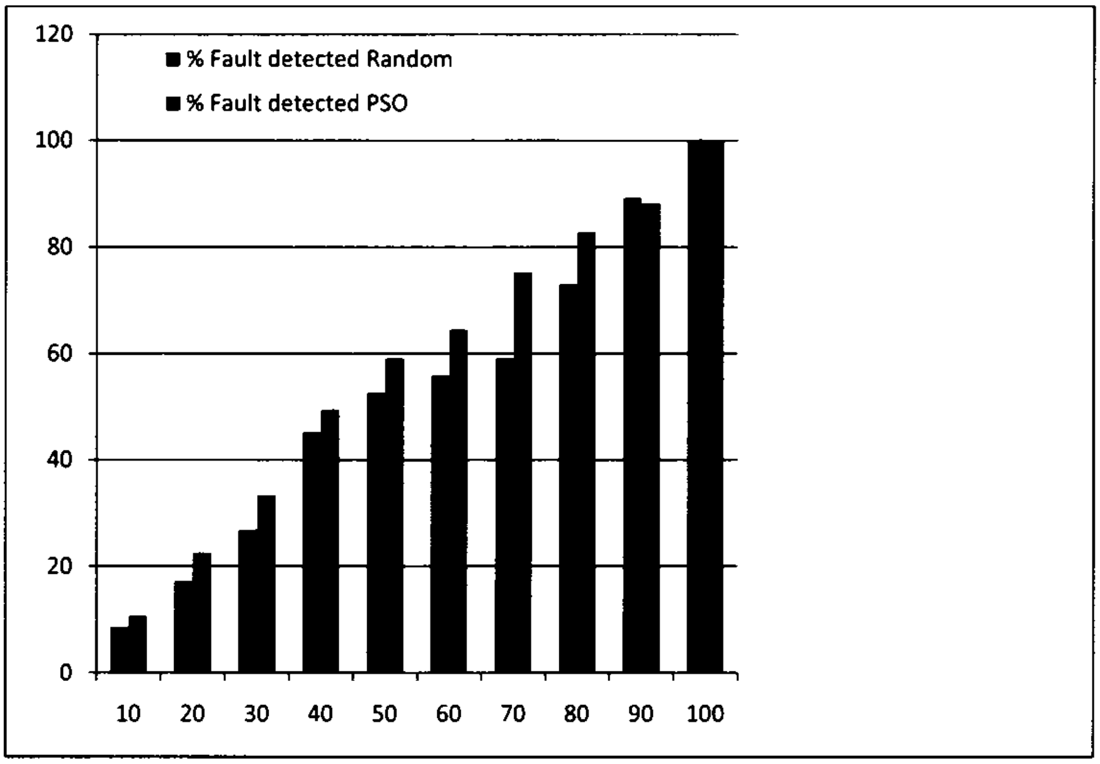

Figure 5.12 Bar Chart Comparison for P#3

The results of these three projects have shown that our proposed algorithm is more effective and efficient in earlier fault detection analysis.

Table 5.7 Summary of Results

% of Test Cases Executed	Experiment 1		[redacted]		Experiment 3	
	% Fault detected Random	% Fault detected PSO	% Fault detected Random	% Fault detected PSO	% Fault detected Random	% Fault detected PSO
10	7	8	0	9	9	11
20	13	20	7	20	17	23
30	18	30	11	35	27	33
40	24	42	20	39	45	49
50	34	54	33	46	53	59
60	49	64	50	57	56	65
70	62	74	63	63	59	75
80	77	80	76	76	73	83
90	91	87	91	89	89	88
100	100	100	100	100	100	100

5.6 Summary

This chapter shows the effectiveness of our proposed technique in early fault detection rate. There is no proof from literature that prioritization of test cases have been done through PSO incorporating value considerations before this research work. Thus proposed algorithm offers an exciting new area of research for test case prioritization. This problem can be solved using different other value based artificial intelligence algorithms. The results have shown the overall worth and improvement that our proposed algorithm has gained in competent early fault detection. This innovative idea to work on maximizing early fault detection will be a huge reduction in terms of time.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

Software testing is a critical area of software engineering which plays a significant role in success or failure of software developments. However, software engineering fails to meet its desired objectives due to several reasons. One major reason is that software testing in general operates in traditional environment. Consequently it is very hard for software testing practitioners to cope up with the significant changes in software development environment. This creates a lot of problem in the ensuring the error free and quality software products. Also this creates extra operating cost to overwhelm software tester errors.

In this thesis, effort has been made to apply the principles of VBSE on intelligent test case prioritization to make it more reasonable and worthwhile for software engineering community and practitioners. We have presented a value based PSO algorithm for automation of the test case prioritization process. PSO based test case prioritization algorithm can be used to found a effective and efficient solution. Our proposed technique can be used to remove the manual effort required in detection of faults in software testing. In this research, the experiments have shown very inspiring results. The results have revealed improvement in analyzing earlier rate of fault detection in comparison of random technique. The proposed algorithm offers a new area of research for test case prioritization. This problem can be solved using different other value based algorithms of artificial intelligence.

6.2 Recommendations for Future Work

As a future work the same experiment could be performed on complex project to generalize our findings. Comparison of the results obtained by PSO can also be compared by using other evolutionally techniques like Genetic Algorithm. Moreover we can apply the same approach to test case prioritization through hybrid PSO with GA. This approach of hybrid PSO with GA is expected to have merits of both techniques. It avoids premature convergence of PSO by using mechanism of GA. Therefore mutation is applied to PSO to increase the diversity of the population and to avoid the local maxima.

References

[1] F. Basanieri, A. Betolino and E. Marchetti, "*The Cow_Suite Approach to Planning and Deriving Test Suites in UML Project,*" Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, Dresden, Germany, pp. 383-397, September 2002.

[2] B. Boehm and L. Huang, "*Value-Based Software Engineering: A Case Study*" *IEEE Computer*, vol. 36, pp. 33-41, March 2003.

[3] B. Boehm, "*Software Engineering Economics*" Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[4] S. Elbaum, A. Malishevsky and G. Rothermel, "*Test Case Prioritization: A Family of Empirical Studies*" IEEE Transactions on Software Engineering, vol. 28, pp. 159-182, February, 2002.

[5] IEEE, "*IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology*" 1990.

[6] J. Karlsson and K. Ryan, "*A Cost-Value Approach for Prioritizing Requirements*" IEEE Software, vol. 14, pp. 67-74, Sep-Oct 1997.

[7] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "*Test Case Prioritization*" IEEE Transactions on Software Engineering, vol. 27, pp. 929-948, October, 2001.

[8] G. Rothermel, R. Untch, C. Chu and M. Harrold, "*Test Case Prioritization: An Empirical Study*" International Conference on Software Maintenance, Oxford, UK, pp. 179 - 188, September 1999.

[9]. Tao Xie. "*Improving Automation in Developer Testing: State of the Practice*"North Carolina State University Department of Computer Science Technical Report TR-2009-6, February 20, 2009.

[10] G. Tassey, "*The Economic Impacts of Inadequate Infrastructure for Software Testing*", RTI Project 7007.011, U.S. National Institute of Standards and Technology, Gaithersburg, Md, USA, 2002.

[11] G. J. Myers. "*The Art of Software Testing*" John Willey & Sons, Inc., New York, USA, 1976.

[12] B. A. Myers, M. B. Rosson, "*Survey on User Interface Programming*". Proc. CHI'92 Human Factors in Computing Systems (May 1992), ACM Press, 195-202.

[13] A. M. Memon, M. E. Pollack and M.L. Soffa, "*Automated Test Oracles for GUIs*" Proceedings of the ACMSIGSOFT 8th International Symposium on the Foundations of Software Engineering (FSE-8), 8–10 November 2000. ACM Press: New York, 2000; 30–39.

[14] Q. Xie and A. M. Memon, "*Using a Pilot Study to Derive a GUI Model for Automated Testing*" ACM Transactions on Software Engineering and Methodology Volume 18 , Issue 2 (November 2008) Article No. 7 Year of Publication: 2008 ISSN:1049-331X

[15] B.Beizer, "*Software Testing Techniques*" International Thomson Computer Press, 1990.

[16] M. Lowry and R. Duran."*Knowledge-based Software Engineering.The Handbook of Artificial Intelligence*" Vol. 4.Addison-Wesley Publishing Company, Inc. 1989.

[17] J. S.Shirabad, Ph.D. thesis, "*Supporting Software Maintenance by Mining Software Update Records*" School of Information Technology and Engineering, University of Ottawa, May 2003.

[18] Craig, D. Rick and S. P. Jaskiel, "*Systematic Software Testing*", 536, Artech House Publishers, Boston. 2002

[19] IEEE Std. 829, IEEE "*Standard for Software Test Documentation*" 1998.

[20] W. C. Hetzel, "*The Complete Guide to Software Testing*" 2nd ed. Publication info: Wellesley, Mass.: QED Information Sciences, 1988. ISBN: 0894352423

[21] A. Ahmed, "*Software Testing as a Service*" Auerbach Publications, New York: 2009.

[22] W. E. Howden, "*Functional Program Testing and Analysis*" McGraw-Hill, 1987.

[23] M. Fewster, "*Common Mistakes in Test Automation*", Grove Consultants, 2001.

[24] S.Berner, R.Weber and R.K.Keller, "*Observations and Lessons Learned From Automated Testing*" in Proceedings of the 27th International Conference on Software Engineering (ICSE '05), pp. 571–579, St. Louis, Mo, USA, May 2005

[25] M. Ramzan, M. A. Jaffar, A. A. Shahid, "*Value based Intelligent Requirement Prioritization (VIRP): Expert Driven Fuzzy Logic based Prioritization Technique*", International Journal of Innovative Computing, Information and Control (IJICIC) Vol.6, No.12, December 2010

[26] A.Windisch, S. Wappler and J. Wegener "*Applying Particle Swarm Optimization to Software Testing*",GECCO'07, July 7–11, 2007, London, England, United Kingdom.

[27] A. Rauf, S. Anwar, N. Kazim Khan, A. A. Shahid, "*Evolutionary based Automated Coverage Analysis for GUI Testing*", Communications in Computer and Information Science (Springer) ISSN: 1865-0929

[28] X. Chen, Q. Gu, J. Qi and D. Chen "*Applying Particle Swarm Optimization to Pairwise Testing*",34th Annual Computer Software and Applications Conference IEEE 2010

[29] Z. Li, M. Harman, and R.M.Hierons "*Search Algorithms for Regression Test Case Prioritization*",IEEE Transaction on Software Engineering, VOL. 33, NO. 4, APRIL 2007

[30] R. Ramler, S. Biffl and P. *Grunbacher "Value-Based Management of Software Testing*", Book Chapter

[31] M. Amram and N. Kulatilaka, " *Real Options* ", Harvard Business School Press, 1999.

[32] "*Value*".Dictionary of Canadian Economics. 2006

[33] "*Value*". New Oxford Companion to Law, ed. Walker, David M. (Oxford: Clarendon Press. 1980)

[34] "*Value*". "Dictionary of Sociology", 2005

[35] J. McTaggart "*The Value Imperative*" (The Free Press, 1994)

[36] D. Ahern, A.Clouse and R. Turner, CMMI Distilled, Addison Wesley, 2001.

[37] "*value.*" Merriam-Webster Online Dictionary. 2008. Merriam-Webster Online. 23 October 2008,

http://www.merriam-webster.com/dictionary/value

[38] B. Boehm, "*Software Engineering Economics*", Prentice Hall, 1981.

[39] Favaro, "*When the Pursuit of Quality Destroys Value*". IEEE Software (May 1996)

[40] Favaro, Favaro, K. R., Favaro, P. F "*Value-based Reuse Investment, Annals of Software Engineering*", (1998)

[41] B. Boehm, B. W "*Value-Based Software Engineering. Software Engineering Notes*", 28(2):2003

[42] B. Boehm and Sullivan, "*Software Economics: A Roadmap in The Future of Software Engineering*", 22nd International Conference on Software Engineering, June 2000.

[43] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "*Using the WinWin Spiral Model: A Case Study*", IEEE Computer, July 1998, pp. 33-44.

[44] Wiegers, K. E. "*Software Requirements*", 2nd ed. Redmond, WA: Microsoft Press, 2003

[45] R. Krishnamoorthi, S.A. Sahaaya and Arul Mary "*Incorporating varying Requirement Priorities and Costs in Test Case Prioritization for New and Regression testing*", 2008

[46] X. Zhang, C.Nie, B. Xu and B.Qu "*Test Case Prioritization based on Varying Testing Requirement Priorities and Test Case Costs*", 2007

[47] H. Srikanth, L. Williams and J. Osborne "*System Test Case Prioritization of New and Regression Test Cases*", 2005

[48] R. Krishnamoorthi and S.A. Mary "*Factor oriented requirement coverage based system test case prioritization of new and regression test cases*",2009

[49] A. M. Smith, G. M. Kapfhammer "*An Empirical Study of Incorporating Cost into Test Suite Reduction and Prioritization*", 2009

[50] K. H. S. Hla, Y. Choi and J. S. Park "*Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting*",2008

[51 H. Park, H. Ryu and J. Baik "*Historical Value-Based Approach for Cost-cognizant Test Case Prioritization to Improve the Effectiveness of Regression Testing*", 2008

[52] S. Amland, "*Risk Based Testing and Metrics,*" 5th International Conference EuroSTAR '99, Barcelona, Spain, pp. 1-20, 1999.

[53] B. Boehm, "*Value-Based Software Engineering,*" *ACM Software Engineering Notes*, vol. 28, pp. 1-12, March 2003.

[54] B. Boehm and L. Huang, "*Value-Based Software Engineering: A Case Study,*" *IEEE Computer*, vol. 36, pp. 33-41, March 2003.

[55] J. Karlsson and K. Ryan, "*A Cost-Value Approach for Prioritizing Requirements,*" *IEEE Software*, vol. 14, pp. 67-74, Sep-Oct 1997.

[56] F. Moisiadis, "*Prioritizing Use Cases and Scenarios,*" 37th International Conference on Technology of OO Languages and Systems, Sydney, NSW, pp. 108-119, 2000.

[57] J. C. Munson and S. Elbaum, "*Software reliability as a function of user execution patterns and practice,*" 32nd Annual Hawaii International Conference of System Sciences, Maui, HI, pp. 255-285, 1999.

[58] J. Musa, "*Software Reliability Engineering*". New York, NY: McGraw-Hill, 1999.

[59] F. Shull, V. Basili, B. Boehm, W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "*What We Learned about Fighting Defects,*" IEEE Symposium on Software Metrics, Ottawa, Canada, pp. 249-258, June 2002.

[60] H. Srikanth and L. Williams, "*On Economic Benefits of System Level Test Case Prioritization,*" International Conference on Software Engineering, St. Loius, MO, pp., 2005.

[61] E. Wong, J. Horgan, M. Syring, W. Zage, and D. Zage, "*Applying design metrics to predict fault-proneness: a case study on a large-scale software system*," Software Practice and Experience, vol. 30, pp. 1587-1608, 2000.

[62] Standish.Group, "*CHAOS.*" http://www.standishgroup.com/chaos.htm.

[63] G.Mogyorodi, "*Requirements-Based Testing: An Overview*," 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, Santa Barbara, California, pp. 286-295, August 2001.

[64] A. Hutchings and S. Knox, "*Creating Products Customers Demand*," *Communications of the ACM*, vol. 38, pp. 72-80, 1995.

[65] C. Jones, "*Software Challenges: Strategies for Managing Requirements Creep*," *IEEE Computer*, vol. 29, pp. 92 - 94, June 1996.

[66] Y. K. Malaiya and J. Denton, "*Requirements volatility and defect density*," 10th Intl' Symposium on Software Reliability Engineering, Boca Ratan, Florida, pp. 285-298, November 1999.

[67] T. Ostrand, E. Weyuker and R. Bell, "*Where the Bugs Are*," Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Boston, MA, pp. 86-96, July 2004.

[68] B.Korel "*Application of System Models in Regression Test Suite Prioritization*" 2008

[69]Y. Fazlalizadeh, A. Khalilian, M. Abdollahi Azgomi and S. Parsa "*Prioritizing Test Cases for Resource Constraint Environments Using Historical Test Case Performance Data*" *IEEE*2009

[70] K. R.Soffa "*Time Aware Test Suite Prioritization*"ISSTA'06, July 17–20, 2006, Portland, Maine, USA.

[71] L. Zhang, S. S Hou, C. Guo, T. Xie and H. Mei "*Time Aware Test-Case Prioritization using Integer Linear Programming*",ISSTA'09, July 19–23, 2009, Chicago, Illinois, USA.

[72] Elbaum "*Incorporating Varying test costs and fault severities into test case prioritization*" 2001

[73] G. Rothermel "*Prioritizing Test Cases For Regression Testing*", IEEE Computer Society, 2001

[74]J. Nocedal and S. J. Wright "numerical optimization" book chapter

[75] P. NagahawateandH.Do "*Effectiveness of Regression Testing Techniques in Reducing the Occurrence of Residual Defects*" Third International Conference on Software Testing, Verification and Validation2010

[76] K.H .S Hla, Y. Choi and J. S. Park "*Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting*", 8th International Conference on Computer and Information Technology Workshops IEEE 2008

[77] M. A. Fayoumi, P. Mahanti and S. Banerjee "*OptiTest: Optimizing Test Case Using Hybrid Intelligence*". World Congress on Engineering 2007.

[78] A. G. Sanchez "*Prioritizing Tests for Software Fault Localization*" 10th International Conference on Quality Software, 2010

[79] G. Kapfhammer and M. Soffa "*Using Coverage Effectiveness to Evaluate Test Suite Prioritizations"ACM,2007*

[80] B. Boehm, "*Value-Based Software Engineering*", ACM SIGSOFT, March 2003

[81] M. G. H. Omran "*Particle Swarm Optimization Methods for Pattern Recognition and Image Processing*" November 2004

[82] "*Global Optimization Algorithms Theory and Application* "Thomas Weise Version: 2009-06-26

[83] S. Mirarab and L.Tahvildari "*An Empirical Study on Bayesian Network-based Approach for Test Case Prioritization*" International Conference on Software Testing, Verification, and Validation 2008

[84] A.Kaur and B.bhatt "*Hybrid Particle Swarm Optimization for Regression Testing*"International Journal on Computer Science and Engineering (IJCSE) Vol. 3 No. 5 May 2011

[85] B. Boehm and Ross, R. "*Theory-W Software Project Management: Principles and Examples.*" IEEE Transactions on Software Engineering 15, 4 (July 1989): 902-916

INDEX 1 (Risk Management Data)

RISK MANAGEMENT		UC Ra ng 1-10	Implimenta o n Complexity (1-10)	Requirem ent Vola lity (1-10)	Traceable (1-10)	Execu on Time (Sec)	Coverage (Func on al)	Fault Impact of Req	
								faults (d)	sever y (v) (1-5
	TC1	3	2	2	10	3	2	3	8
	TC2	3	2	2	10	3	2	3	8
UC01	TC3	3	2	2	10	3	2	3	8
	TC4	5	5	4	4	10	3	3	32
	TC5	5	5	4	4	10	3	3	32
UC02	TC6	5	5	4	4	20	3	3	32
	TC7	5	8	5	5	15	3	4	32
	TC8	5	8	5	10	8	3	2	32
	TC9	5	6	3	8	35	3	3	32
	TC10	5	6	1	10	20	3	0	0
UC03	TC11	5	7	8	4	15	5	1	32
	TC12	4	8	3	10	10	[illegible]	[illegible]	32
UC04	TC13	4	8	3	10	40	[illegible]	3	32
	TC14	6	8	2	9	30	1	2	32
UC05	TC15	6	7	5	10	20	2	3	16
	TC16	5	8	5	5	5	2	3	32
	TC17	5	8	5	10	45	7	5	32
UC06	TC18	5	7	5	2	5	5	3	32
	TC19	7	9	5	5	10	2	2	32
UC07	TC20	7	9	5	10	15	2	2	32
UC08	TC21	5	8	1	7	20	4	3	8
	TC22	8	10	8	6	5	1	1	32
UC09	TC23	8	10	8	6	10	1	1	32
UC10	TC24	9	10	8	3	15	4	3	32
	TC25	8	5	1	4	2	0	0	0
	TC26	8	5	1	9	3	0	0	0
UC11	TC27	8	5	1	5	4	0	0	0
	TC28	10	6	1	10	8	0	0	0
UC12	TC29	10	6	1	8	12	1	1	8
UC13	TC30	9	6	1	10	15	1	1	8
	TC31	7	6	1	1	10	2	0	0
UC14	TC32	7	6	1	10	8	7	1	8
	TC33	8	6	1	8	10	2	0	0
UC15	TC34	8	6	1	6	5	7	1	8
UC16	TC35	7	5	1	4	10	2	1	8
	TC36	9	5	1	10	5	1	1	8
	TC37	9	5	1	10	5	3	1	32
UC17	TC38	9	5	1	3	10	3	0	0

	TC39	9	5	1	7	15	1	1	4
	TC40	9	5	1	10	15	1	1	16
UC18	TC41	8	8	5	8	10	2	1	32
UC19	TC42	9	5	2	10	5	2	0	0
UC20	TC43	7	8	1	6	8	1	0	0
[illegible]	TC44	8	8	2	10	10	[illegible]	[illegible]	[illegible]
[illegible]	TC45	8	8	2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
UC22	TC46	7	8	4	10	10	5	3	16
UC23	TC47	8	5	4	10	28	5	2	16

INDEX 2 (IT Procurement Suite Data)

IT Procurement Suite		UC Ra ng 1-10	Impliment Complexity (1-10)	Req Vola lity (1-10)	Traceable (1-10)	Execu on Time (Sec)	Coverage (Func on al)	Fault Impact of Req	
								faults (d)	severity (v) (1-5)
	TC1	3	4	3	10	6	2	0	0
	TC2	3	3	4	8	7	2	0	16
UC01	TC3	3	5	5	9	5	2	2	8
	TC4	7	8	5	9	7	3	2	16
	TC5	4	6	3	8	5	3	1	32
[illegible]	TC6	4	8	5	9	6	3	[illegible]	[illegible]
[illegible]	TC7	4	9	4	8	6	[illegible]	2	16
[illegible]	TC8	4	8	6	10	4	[illegible]	[illegible]	4
	TC9	5	7	4	10	7	3	4	32
	TC10	5	8	5	7	8	3	3	4
UC05	TC11	5	6	7	9	8	3	3	8
	TC12	6	7	5	10	7	3	4	0
	TC13	6	8	6	9	9	3	2	16
UC06	TC14	6	9	8	8	9	3	3	4
	TC15	7	5	3	10	7	1	4	32
UC07	TC16	7	4	2	7	6	1	3	16
UC08	TC17	7	9	4	10	7	2	4	32
	TC18	7	10	7	9	5	2	4	8
	TC19	7	7	3	10	4	2	4	32
	TC20	7	8	2	8	8	2	2	16
UC09	TC21	7	9	9	8	9	2	3	8

INDEX 3 (Sales Intranet Data)

Sales Intranet Data		UC Ra ng 1-10	Implimenta on Complexity (1-10)	Req Vola lity (1-10)	Traceable (1-10)	Execut Time (Sec)	Coverage	Fault Impact of Req	
								faults (d)	severity (v) (1-5)
UC01	TC1	1	2	3	10	3	3	0	[illegible]
	TC2	3	5	3	10	5	2	2	[illegible]
	TC3	3	6	4	9	6	2	3	[illegible]
	TC4	3	4	5	10	5	2	2	[illegible]
	TC5	3	6	6	8	7	2	4	[illegible]
	TC6	3	7	3	9	4	2	2	[illegible]
UC02	TC7	3	6	8	7	5	2	5	[illegible]
	TC8	3	6	3	10	6	2	1	[illegible]
	TC9	3	6	4	8	5	2	2	[illegible]
UC03	TC10	3	5	6	7	4	2	3	[illegible]
	TC11	4	7	3	9	6	[illegible]	[illegible]	[illegible]
	TC12	4	7	4	8	5	[illegible]	[illegible]	[illegible]
UC04	TC13	4	6	6	7	4	[illegible]	[illegible]	[illegible]
	TC14	3	8	5	10	3	1	1	[illegible]
	TC15	3	8	7	9	3	1	3	[illegible]
	TC16	3	7	4	8	4	1	2	[illegible]
	TC17	3	6	6	9	5	1	4	[illegible]
	TC18	3	8	3	9	3	1	3	[illegible]
UC05	TC19	3	7	4	8	3	1	5	[illegible]
	TC20	10	10	4	10	10	1	0	[illegible]
UC06	TC21	10	9	6	9	9	1	2	[illegible]
	TC22	9	9	5	10	10	2	1	[illegible]
UC07	TC23	9	9	4	9	10	2	4	[illegible]
UC08	TC24	9	9	6	9	3	2	1	[illegible]
	TC25	10	10	6	10	3	2	1	[illegible]
UC09	TC26	10	9	7	9	4	2	3	[illegible]
	TC27	9	9	5	10	3	2	1	[illegible]
	TC28	9	9	7	8	4	2	3	[illegible]
UC10	TC29	9	8	6	7	5	2	3	[illegible]
	TC30	8	8	5	9	3	2	1	[illegible]
	TC31	8	9	7	10	3	2	2	[illegible]
	TC32	8	8	6	9	4	2	3	[illegible]
	TC33	8	8	4	8	5	2	2	[illegible]
	TC34	8	7	6	9	4	3	1	[illegible]
UC11	TC35	8	7	5	8	6	2	2	[illegible]
UC12	TC36	8	8	5	10	3	2	1	[illegible]

	TC37	8	9	6	9	3	2	2	[redacted]
UC13	TC38	8	8	4	9	3	2	1	[redacted]
	TC39	8	8	3	10	5	1	1	[redacted]
UC14	TC40	8	9	5	9	4	2	1	[redacted]