CHAPTER 1

INTRODUCTION

1.1 GENERAL

This chapter discusses about the concepts of software testing, test data generation, test case selection and model-based testing techniques. It also tends to present the motivation for the research, problem definition, and objectives of the research work, research methodology adopted and organization of the thesis. In recent years, search based software engineering has evolved as one of the major research field in the software engineering community (Afzal et al. 2009). Search Based Software Engineering (SBSE) has been applied successfully to many software engineering activities ranging from requirements engineering to software maintenance and quality assurance (Harman 2007). Figure 1.1 shows the five phases of software engineering. Among the five phases of software testing, the current research work concentrated on testing is a verification and validation process.

![Figure 1.1 Phases of Software Development Life Cycle](image-url)
1.2 SOFTWARE TESTING

Software testing involves identifying the test cases which detect errors in the program. Collection of test data inputs are termed as test case and collection of test cases are called as test suite or test set (Beizer 1990). As software grows and evolves, new test cases are generated and added to a test suite to exercise the latest modifications to the software. However, exhaustive testing of software is more time consuming and it requires more number of test cases which makes testing process more complicated. The testing effort can be optimized by executing only a selected subset of the test cases that are believed to have a better change of revealing faults (Bertolino 2003). In object-oriented programming, the number of methods and conditions to be tested is more. So, test automation will be one of the solutions for reduction in execution time. Automation process involved in testing consists of test case generation, test case minimization, test case selection and prioritization (Ting Shu et al. 2012). Several researches have been carried out to address the problem encountered in software testing using different techniques. However the problem continues giving scope for further research in test case generation issues.

1.3 TESTING METHODS

Test cases are developed using various test techniques to achieve more effective testing (Myers 1979). Test cases provide software completeness and testing conditions which gives better results in finding errors. So, testers do not guess which test case and test technique have to be chosen to enable them to design testing conditions in a systematic way (Ould and Praxis 1991). Moreover combination of test techniques can give better results than use of single test technique (Lu Luo 2011). Software Testing can be broadly classified into two techniques (Srinivas and Jagruthi 2012):

- Black-box testing and
- White-box testing
In addition to the above two techniques the other important testing methods are evolutionary testing and search based testing.

**Black-box testing**

Black-box testing treats the software as a ‘black box’, examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it. The main advantage of black box technique is that no prior knowledge is required and the disadvantage is limited coverage since only a selected number of test scenarios are actually performed (Laurie Williams 2006). The commonly used black box techniques are shown in Figure 1.2. Among the different techniques, model based testing technique is useful for analyzing system functional behavior and large complex systems (Ina Schieferdecker et al. 2012). Hence the concept of model based testing is chosen for the present research work to develop enhanced testing methods.

**White-box testing**

White-box testing is also known as structural testing. It is carried out to test internal structures or working of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. It can test paths within a unit, between units during integration, and between subsystems during a system-level test (Sapna Varshney and Monica Mehrotra 2014). The various white-box testing techniques used are shown in the above Figure 1.2. White-box testing consists of Path, Loop and Control structure. White box testing is code based and carried out on specific models and for more structural behavior analysis. The concept of code based testing is considered in this research work for test case minimization.
Figure 1.2 Classification of Testing Techniques

**Evolutionary Testing**

Evolutionary testing (ET) is a technique coming under evolutionary computation through which test data can be generated automatically by the use of optimization search techniques. The search space is the input domain of the software under test. ET has been shown to be successful for generating test data for many forms of testing, namely specification testing, extreme execution time testing and structural testing (Phil McMinn and Mike Holcombe 2003). One of the approaches to automate test case design for various test objectives is evolutionary testing. ET refers to the use of meta-heuristic search techniques for test data generation. The concept of ET is used in this research work to optimize the test case generation.
Search-Based Testing (SBT)

Search-based testing is one of the automation approaches in which the task of generating test data is transformed into an optimization problem which is solved using meta-heuristic search techniques. SBT is investigated for use in code-level testing in which testing is done for both functional and non-functional properties by considering certain structural coverage criteria (Kalaji et al. 2011). Search based testing is popular nowadays due to various advantages like less computation time, more accurate testing etc. Hence the concept of search based optimization technique is used in this research work and has been considered to optimize the test suite size.

1.4 OPTIMIZATION TECHNIQUES

The several optimization techniques that have been successfully applied to test data generation are categorized and shown in Figure 1.3. Search-based techniques (Phil McMinn 2004) are used to generate the input test-data, as the input space of the program can be large and quite complex. Optimization techniques produce better results in sampling large search spaces.

1.4.1 Meta-heuristic Optimization Search Techniques (MOST)

Metaheuristics are strategies that guide the search process to find near-optimal solutions. Meta-heuristic Optimization Search Techniques (MOST) are high-level frameworks which utilize heuristic to seek solutions for combinatorial problems (Phil McMinn 2004) within a large multi-modal search space at a reasonable cost. There are two types of search techniques, local and global.
Local Search Techniques (LST)

Local Search Techniques (LST) are based on the iterative exploration of a solution space: at each iteration, a Local Search algorithm steps from one solution to one of its ‘neighbors’ i.e., solutions that are close to the starting one.

Local Search meta-heuristics are an emerging class of methods for tackling combinatorial search and optimization problems, which recently have been shown to be very effective for a large number of combinatorial problems (Luca Di Gaspero 2003). One major drawback of this family of techniques is the lack of robustness on a wide variety of problem instances. In fact, these methods assure finding good results in reasonable running times, whereas in other cases local search techniques are trapped in the so-called local minima. In order to resolve this drawback, local search techniques combined with...
other optimization techniques named hybrid approaches or global search methodologies would be the solution for local minima.

**Global Search Techniques (GST)**

Global search techniques aim to overcome the problem of local optimum in the search space and find more globally optimal solutions. The commonly used global search techniques are evolutionary computation, genetic algorithms, and particle swarm optimization. These techniques are population-based approaches which produce multiple candidate solutions for search problems.

1.4.2 **Evolutionary Search using Genetic Algorithms**

Genetic Algorithms (GAs) belong to the larger class of Evolutionary Algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection and crossover. GAs search for optimal solutions by sampling the search space at random and creating a set of candidate solutions called a ‘population’. These candidates are combined and mutated to evolve into a new generation of solutions within the population (Back 1997).

GAs operates on a population of individuals (often called chromosomes) each of which has an assigned fitness. An advantage of GA is that the solution domain is fully sampled using Fitness Function (FF). FF plays an important role in finding best solution in the search space. It helps to differentiate good solutions from poor ones, there by guiding the search. Because often the first population does not have the final or “good enough” solution, there is a need for keeping an artificial diversity in the population. Diversity can be maintained by using the crossover and mutation operations.
The mutation is performed by applying a random change to the individual’s chromosomes (David Goldberg 2008).

A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection* and guarantees that the solution quality obtained by the GA will not degrade from one generation to the next.

1.4.3 Single and Multi-Objective Optimization

In single-objective optimization problem only single optimal solution is required whereas multi-objective optimization problem produces a set of solutions which are superior to the rest of the solutions with respect to one or more objectives. These solutions are called ‘pareto’ optimal solutions or non-dominated solutions (Kalyanmoy 2005).

Software test case optimization problems such as classification, minimization, selection, and prioritization have been focuses in software testing by considering multiple objectives. Instead of considering single objective, multiple objectives of test cases optimization such as maximum number of fault detection capability, minimum test design efforts/cost, minimum testing cost/execution cost, maximum coverage of client requirements/codes, minimum test execution time/effort, and maximum mutant killing score are generally considered as parameters for assessing the fitness of test cases (Anon Sukstriendwong 2011).

Most real world problems have multiple objectives to achieve; this situation creates a set of problems in Operation Research (OR) called Multi-Objective Optimization Problems (Abdullah 2006). A Multi-Objective Optimization Problem has a number of objective functions, which are to be
minimized or maximized. The general form for Multi-Objective Optimization Problems (MOOP) can be expressed as (Dun Wei Gong and Yong Zhou 2006),

\[ f_m(x), \text{ where } m = 1, 2, 3 \ldots M \text{ number of objective functions and} \]

\[ x = \text{single candidate solution}. \]

Multi-Objective Optimization is sometimes referred as vector optimization, because a vector of objectives, instead of a single objective is optimized. For example, try to achieve maximum path coverage with a minimum execution time. MOOP can be linear or non-linear type (Ghiduk 2010). Genetic Algorithms are the most popular heuristic technique to solve Multi-Objective Optimization Problems and hence an attempt is made to model a GA based optimization testing technique in the present research work (Coello et al. 2007).

1.5 TEST SUITE SIZE MINIMIZATION

Software testing and retesting occurs continuously during the software development lifecycle. As software grows and evolves so do the accompanying test suites. Over time, some test cases in a test suite may become redundant as the requirements executed by them are also executed by other test cases in the test suite (Ostrand and Balcer 1998). Due to time and resource constraints for re-testing the software every time it is modified, it is important to develop techniques that keep the test suite size manageable by removing those test cases that may have become redundant with respect to the coverage of program requirements and this process is known as Test Suite - Size Minimization (TSM) (Balcer (1989). Since test suite minimization
removes test cases, minimized suites may be weaker at detecting faults in software than their minimized counterparts (Hutchins et al. 1994).

Previous work on test suite minimization has shown some conflicting results. Wong et al. (2014) reported that minimizing test suites while keeping all-uses coverage constant could result in little to no loss in fault detection effectiveness. However, the empirical study conducted by Rothermael et al. (1998) suggests that minimized test suites can severely compromise the fault detection capabilities of the test suites. There are two implications of this conflict: first, there are situations where minimization can achieve high suite size reduction without significantly decreasing fault detection effectiveness; second, there are also situations where minimization can achieve high suite size reduction at the expense of significant loss in fault detection effectiveness.

A concept of selectively retaining test cases by considering multi-criteria for test suite size reduction is the initial step of TSM. In addition to number of test cases, fault detection effectiveness is also considered while doing optimization process. Removal of certain redundant test cases reduces the test suite size significantly. Test suite size reduction is required for all types of applications including internet (Alireza Khalilian and Saeed Parsa 2012).

Reduction techniques help in effective testing by reducing test suite size, execution time and cost. By changing an existing heuristic technique new approaches can be explored (Shin Yoo and Mark Harman 2012). Hybrid optimizations perform better in test suite size reduction at the same time retaining the quality of test suite. Usually GA is combined with local search technique which gives improved results in test optimization (Dharmalingam et al. 2010).
Existing test suite minimization techniques are defined in terms of test case coverage as they attempt to minimize the size of a suite while keeping some coverage requirement constant (Garey and Johnson (1979)). A simple greedy algorithm for the test suite minimization problem is described in Chvatal (1979). An algorithm developed by Harrold et al. (1993) based on a heuristic to select a minimal subset of test cases that covers the same set of requirements as the unminimized suite. Agarwal (1999) used the notion of mega blocks to derive coverage implications among the blocks to reduce test suites such that the coverage of statements and branches in the reduced suite implies the coverage of the rest. Sampath et al. (2004) used concept analysis for incrementally creating and maintaining a reduced test suite for web applications. A number of empirical studies have been available for traditional test-suite reduction. Some empirical studies do not consider the lowering of fault-detection capability caused by test-suite reduction.

An algorithm based on a heuristic to select a representative set of test cases from a test suite, providing the same coverage as the entire test suite, was developed by Harrold, Gupta and Soffa (1993). Jeffrey and Gupta (2007) reported that test suite minimization with respect to a given testing criterion can significantly diminish the fault detection effectiveness (FDE) of suites. Hence a new approach was presented for test suite reduction that attempts to use additional coverage information of test cases to selectively keep some additional test cases in the reduced suites that are redundant with respect to the testing criteria used for suite minimization, with the goal of improving the FDE retention of the reduced suites. They implemented their approach by modifying an existing heuristic for test suite minimization and concluded that it significantly improve the FDE of reduced test suites without severely affecting the extent of suite size reduction.
The work presented by Heimdahl and George (2004) uses a greedy technique for suite reduction in the context of model-based testing. This work showed that while suite sizes could be greatly reduced, the fault detection capability of the reduced suites was adversely affected. A new model for test suite minimization has been developed by Black et al. (2004) that explicitly consider two objectives: minimizing a test suite with respect to a particular level of coverage, while simultaneously trying to maximize error detection rates with respect to one particular fault. A limitation of this model is that fault detection information is considered with respect to a single fault (rather than a collection of faults), and therefore there may be limited confidence that the reduced suite will be useful in detecting a variety of other faults.

Techniques for test suite minimization that are specifically tailored to consider the complexity of the modified condition/decision coverage criterion was developed by Jones and Harrold (2003). But the limitation of this approach is the fault detection loss of suites reduced under these techniques may vary greatly depending upon the particular program and test suites used. Suite size and fault detection effectiveness are opposing forces in the sense that more suite size reduction would intuitively imply more fault detection effectiveness loss. Because throwing away more test cases, in effect, throws away more opportunities for detecting fault. Thus, there seems to be an inherent tradeoff involved in test suite reduction: one may choose to sacrifice some suite size reduction in order to increase the chances of retaining more fault detection effectiveness. Some new algorithm may provide a framework for testers to have more flexibility in determining the conditions of this tradeoff.

Recent developments on test suite size minimization focuses on hybridization of evolutionary approaches where GA is combined with local search technique. Hybrid optimizations perform better not only in test suite
size reduction but also in retaining the quality (Isha Mangal et al. 2014; Jeya Mala Dharmalingam et al. 2014; Ekta Tyagi and Yashika Sharma 2015).

Hence, the existing techniques for test suite minimization do not consider keeping any redundancy with respect to the given coverage criterion during the suite minimization process. Any test suite minimization algorithm addressing the above problem can be modified to incorporate some approach to generate reduced test suites that selectively retain some of the test cases that are redundant with respect to the given coverage criterion. Hybrid algorithms suitably address the above issues in test suite size minimization.

1.6 TEST CASE SELECTION

Test case selection is a method of selecting a subset of test cases from a test suite to reduce the time, cost and effort in software testing process. It is very much similar to test suite minimization technique. The test suite minimization technique is based on metrics like coverage measured from a single version of the program under test. The difference between these two techniques depends upon the changes made in system under test. The test cases are selected according to the changes made between the previous and the current version of the system under test (Biswal and Baikunth 2010).

From the minimized test suite, selection process plays a major role and then prioritization. The process of choosing a subset of test cases from the test suite is performed in minimization as well as selection. The key difference (Shin Yoo and Mark Harman 2012) between these two approaches is selection focuses on the changes in the software under test. Test suite minimization is often concentrated on metrics such as coverage measured from a single version of program under test. By contrast, in test selection, test cases are selected because their execution is relevant to the changes between the previous and the current version of the software under test. Many
approaches have been used for regression test case selection namely Integer programming, Data-flow analysis, Symbolic execution, Dynamic slicing-based, Graph-walk, Textual difference, SDG slicing, Path analysis, Modification-based, Firewall, and UML design-based. Similar to minimization technique, hybrid approaches are used for selection as well as for prioritization (Rothermel 2001). Most of the existing methods are restricted to specific data representations (i.e., notations) as well as not suited for multi-objective optimization. Hence, there is a need for further research work to focus on methods for optimizing the test case selection process.

1.7 TEST CASE PRIORITIZATION AND EVALUATION

The basic difference between test suite minimization and test case prioritization is given here. The test suite minimization techniques attempt to remove test cases from the suite whereas the test case prioritization techniques only re-order the execution of test cases within a suite with the goal of early detection of faults (Elbaum et al. 2002; Rothermel et al. 2001 and Srivastava and Thiagarajan 2002).

Test case prioritization is a method of scheduling and ranking the test cases from multiple test suites of software. There are many approaches to schedule and rank the test cases. Each and every test case is assigned some priority but sometimes there may be some issues arise when multiple test cases have the same priority or the weights. Sometimes problem occurs in prioritizing these multiple test suites (Catal and Mishra 2012).

Test case prioritization seeks to find the ideal ordering of test cases for testing, so that the tester obtains maximum benefit, even if the testing is prematurely halted at some arbitrary point. Early Fault Detection (EFD) capability is the main metric considered for test case prioritization. In addition
to EFD cost-effectiveness could also be considered during prioritization. Class dependence analysis is one of the methods for test case selection and prioritization of object-oriented programs. Criteria-based approaches namely distribution-based, human-based, probabilistic, history-based, requirement-based, and model-based are available. Rather than that other approaches like mutation-based, user session-based, component-based, property relevance-based etc., Coverage-based and Cost-aware techniques also exist for prioritization (Siripong Roongruangsuwan and Jirapun Daengdej 2010).

The other approach is based on designed model, where the relevant test cases are assigned into high and low priority test cases based on the designed model. Supriya and Ilavarasi (2015) describes an experiment’s design, measurement metrics and results in order to determine the most recommended test case prioritization method. The test cases are evaluated to assess and compare the suitable test cases to test the software. These methods help in finding the minimum number of test cases for testing software.

**Multiple Test Suite Prioritization (MTSP)**

Siripong (2010) describes the Multiple Test Suite Prioritization (MTSP) method, which is used to prioritize the test cases from multiple test suites. The entire program is divided into number of test suites and these test suites contains multiple test cases. The test cases are prioritized according to weight and rank that are used for testing the program. Rothermal (2001) considered nine approaches for prioritizing a set of test cases and reported results. He also presented different approaches for revealing the faults to improve the software quality. Regression testing is the process of testing the software against those changes in the existing software. The four methods of regression testing are reset method, regression test selection method, test suite reduction and test case prioritization method.
Shin Yoo and Harman (2010) presented various test case prioritization approaches that are based on some criteria. The first approach is Distribution-based approach, in which the test case profiles are distributed based on the dissimilarity metric. Using this metric, the clusters of these test cases are prepared according to their profiles. The test cases having similar profiles get clustered into one redundant group of test cases and other groups indicate the unusual conditions that cause the test case failures. The second approach is Human based approach which is based on Case-Based Reasoning (CBR) in which a Rank shoot algorithm is taken that selects test cases according to their ranks provided.

Cagatay Catal and Deepti Mishra (2012) contributed some other approaches where the test cases get prioritized according to the probability of test case selection methods. The test cases are selected based on some factors like cost, length of test cases etc. The history based approach is associated with the clusters based on some pervious artifacts that are obtained by matrix analysis. Some test cases are prioritized according to the software requirements of the customers.

**Cluster-based Approaches for Test Optimization**

By incorporating expert knowledge through Machine Learning (ML) techniques namely cluster-based approaches are widely used for test case selection and prioritization. An improvement in test case prioritization through clustering approach has been proven with a help of industrial case study (David et al. 2014).

A hybrid meta-heuristic is one which combines a meta-heuristic with other optimization approaches such as algorithms from mathematical programming, constraint programming, and machine learning. A genetic algorithm-based clustering technique has been introduced for cluster
optimization with artificial and real-life data sets (Sangeeta et al. 2011). A combination of various optimization techniques at multiple levels is theoretically introduced as Multi-Objective Regression Test Optimization (MORTO) in which test suite size reduction is considered as major criteria.

Though several techniques are used for test suite prioritization some limitations are present in these approaches. Hence enhanced model is required to be developed to optimize test cases in best available manner to generate most effective test cases, evaluate them and further minimize and select best test cases using a firm mechanism. One such approach is group wise clustering algorithm.

1.8 SYSTEM-LEVEL TESTING TECHNIQUES

System-level testing approach relies on a formal model built to support the testing activity. This approach can offer a number of benefits, such as high fault detection ratio, reduced cost and time, traceability, and ease of handling requirements evolution. As models play a crucial role in MBT, a major testing challenge moves to the modeling of system under test itself. Recently test models have been combined with test purposes and properties for test generation (Ali et al. 2010).

Two main types of models are currently considered in the testing context, holistic models and scenario models. A number of test scenarios describing various aspects of the expected behavior are scenario-based models. Scenarios are usually described using message sequence charts and UML sequence diagrams, including a variety of timed scenario notations. Holistic models describe the behavior of the system under test with some level of detail, trying to capture the features that are relevant to testing (Blackburn et al. 2002). Again they can be divided into sequential and true-concurrency models in which sequential models include Extended Finite State
Machines (EFSMs), Input/ Output Transition Systems (IOTSs), Unified Modeling Language (UML) and Specification Definition Language (SDL).

The true-concurrency models such as Petri nets and Event-driven are used for generation and minimization of test cases. The parameters considered for optimization are test suite size, coverage, execution time, early fault detection and weight per test case. The advances in software engineering which include the use of executable specification languages, the pattern-based detection of faults in source code, or the inference of program behavior from runtime observations, contribute to a renaissance of MBT approaches (Bringmann and Kramer 2008).

1.8.1 System-Level Test Case Selection

In model based technique, the subset of the requirements gets modeled by using formal notations such as specifications of the subset of the requirements. Leon and Podgurski (2003) presented an empirical comparison of four different techniques for filtering large test suites-test suite minimization, prioritization by additional coverage, cluster filtering with one cluster sampling, failure pursuit sampling. Rothermal (2001) also define a technique for test suite minimization where the size of the test can be reduced by eliminating redundant test cases from the test suite. Therefore minimization method is also called as test suite reduction. Leung and White present the first systematic approach to regression testing and test cases. Tallam et al. (2006) performed two types of reduction on the lattice. Lattice is a natural representation that supports the identification of the test cases. Test suite optimization problem is solved by implementing the model based test suite optimization technique. In this technique the evolutionary based algorithms are used for optimizing the test suite. Another technique used to reduce the total number of test cases are Extended Finite State Machine (EFSM). It is basically used to reduce the regression test suites.
1.8.2 System-Level Test Prioritization (SLTP)

The commonly used technique to perform MBT is Dynamic Dependence Analysis (DDA). The combined concepts of dynamic and interaction-based testing are used for finding dynamic behavior of a system. The process of test case optimization by identifying similar test patterns is used to eliminate redundant patterns. Test prioritization is performed for different model-based systems.

Recent developments in model-based test prioritization for various systems (Sanjukta Mohanty et al. 2011; Cagatay Catal and Deepti Mishra 2012; Yuen Tak Yu and Man Fai Lau 2012; Ajeet Kumar Pandey and Vivek Shrivastava2011 and Hema Srikantha and Sean Banerjee 2012) motivates us to do experimentation on model-based test prioritization. In existing method test suite minimization is performed using Dynamic Interaction Pattern (DIP) (Selvakumar and Ramaraj 2011; Tahat et al. 2011) by removing redundant patterns. However the concepts of assigning priorities to interaction patterns are attempted in the earlier studies. The concept of assigning priorities to interaction patterns for test case prioritization using DIP is taken as a proposal in this research work.

Instead of considering single test case, an entire test pattern is considered for eliminating the similar patterns (Rachna and Arvind 2012). Interaction among test patterns are found using data and control dependencies in states. Interaction patterns found at runtime is generally known as Dynamic Interaction Pattern (DIP); early fault detection ability is tried to improve using DIP. When compared to static, dynamic patterns are more advantageous.
1.8.3 System-Level Security Testing (SLST)

System-level security testing (SLST) is a relatively new field and especially dedicated to systematic specification and documentation of security test objectives, security test cases and test suites, as well as to their automated or semi-automated generation. In particular, the combination of security modeling and test generation approaches is still a challenge in research and of high interest for industrial applications. SLST is a combination of four approaches namely security testing, risk-oriented testing, model-based testing and test automation (Mitrabinda and Durga Prasad 2013).

Security testing techniques allow specification of test cases at a higher level of abstraction, for enabling guidance on test identification and specification as well as for automated test generation. Risk Analysis (RA) is the quantitative analysis of risk present in a system based on the threat modeling results. RA is performed to find the vulnerable states that need to be tested (El Far and Whittaker 2002).

Security testing does not provide due importance to threat modeling and risk analysis simultaneously that affects confidentiality and integrity of the system. In order to enhance the performance of test models a security testing mechanism based on risk analysis results using STRIDE threat modeling approach has been proposed for identifying highly risk states and the proposed model is termed as Model-Based Security Testing (Dianxiang et al. 2012).

Risk-driven Security Testing (RST) and Test-driven Security Risk analysis (TSR) are the two commonly used risk analysis approaches. Security risk analysis is a specialized RA approach in which information security risk associated with the potential threats will be evaluated. In RST, security testing is supported by security risk assessment in order to make ST more effective.
by executing only the selected test cases. RST comparatively performs better in improved test suite reduction rate with better coverage results. Risk-oriented testing uses risk analysis results in test case identification, selection and assessment to prioritize and optimize the testing process (Bouquet et al. 2014).

For test optimization, test cases covering high risk states are selected for execution. There are different types of threat modeling processes which are used to identify threats and to identify stakeholder's risk. The two different Microsoft threat modeling processes are STRIDE and DREAD. STRIDE is an acronym of six types of threats; Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. It is used to identify both technical and non-technical threats (Dianxiang et al. 2012).

DREAD stands for Damage, Reproducibility, Exploitability, Affected users and Discoverability. It is used for rating threats and also for quantifying, comparing and prioritizing the amount of risks associated with each threat. Applicability of combinatorial testing to web-based application security is one of a recent trend used in testing internet applications (Bernhard et al. 2014). It is important to identify the threats associated with a system which identify vulnerabilities. The concepts of security are applicable to real-time systems and so models of the system are needed for better testing which indeed leads to Model-Based Security Testing (MBST).

1.8.4 Domain Specific System-Level Testing (DSSLT)

Integrating MBT with specific domain features is called Domain Specific System-Level Testing (DSSLT) (Eddy et al. 2014). Models use different abstractions and notations for different specifications. Generally the system model is represented using Unified Modeling Language (UML) which
provides a better way for deriving Domain Specific Language (DSL). MBT enables test automation which is used for testing functional behavior of a system. Domain Specific Modeling (DSM) involves systematic use of DSL to represent various aspects of a system. DSM Languages support higher-level abstractions which require less effort to specify a given system. In existing system, the specific domain considered for MBT is Voice Recorder in which code generation automates the creation of executable source code directly from DSM models. Enhancements in model-based testing approaches for industrial applications are developed and applied for similar case studies (Endo et al. 2013).

In MBT although several attempts have been made for test case optimization, the concept of combining two dimensions namely risk identification and automated test generation are not explored much. Hence, in this research work an enhanced model of combining risk analysis using STRIDE threat modeling and automated test generation are developed.

1.9 MOTIVATION FOR THE RESEARCH

The motivation for the research is to minimize the test cases by eliminating the redundant test cases from a given test suite. Because test case minimization is an important problem in software testing as it involves lots of software organizations money, time and resource. Nowadays software system failures are frequent, widespread and serious. According to Gallop (2009), the top ten software failures of 2009 accounts to £34.99 billion as a result of software crashes of Amazon, UK border and immigration department etc. Hence software testing is more important nowadays and test case size optimization problem equally attracts researchers to develop enhanced models.
Optimization testing techniques attempted by earlier researchers on test case size minimization, test case selection and test case prioritization had several limitations such as considering single parameter for optimization, testing without eliminating redundant test cases, without considering changes between before and after software under test etc. Due to time and resource constraints for retesting the software system on modification, it is important to develop techniques that keep test suite sizes manageable by periodically removing redundant test cases.

However optimization techniques to minimize test cases which can eliminate redundant test cases from a given test suite are not developed to a major extend. Hence optimization testing technique based on multiple objectives is investigated in the present research and several models have been developed.

1.10 PROBLEM DEFINITION

Software testing is an important component of software quality assurance and many software organizations are spending nearly 40% of their resources on testing. The testing procedure and cost varies from one application to other. Hence several studies have been made to study about risk analysis in software testing. The normal problems encountered during the software testing are schedule delays, cost overruns, outright cancellations and many other factors. In general software testing involves a number of test cases known as test suite. When the size of a test suite is large, it is difficult to manage the test cases which resulted in increased usage of computation resources and execution time requirement. Hence there is a need for optimization techniques to minimize test cases which can eliminate redundant test cases from a given test suite.
Multi-objective optimization (MO) also known as multi-criteria optimization is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. Test cases optimization is the problem of finding the best subset or class of test cases from a pool of test cases to be audited. It will meet all the objectives of testing concurrently. Several researches have been carried out in optimization testing techniques to evaluate the fitness of test cases through the single parameter such as either coverage or fault detecting capability. But very limited researches have been carried out considering multiple parameters such as execution time, test suite size, fault detection ability in addition to coverage for test case optimization. Hence it is felt that optimization testing technique based on multiple objectives would be a worthwhile attempt.

In the present research multiple parameters such as execution time, faults detection ability, test suite size and coverageability are considered simultaneously. This multi objective optimization would result in reduced testing time and search based testing technique would improve the quality of testing.

The objective of test cases optimization is to reduce the number of test cases in suite to be audited and improve the effectiveness/fitness of test cases. So, test cases fitness evaluation, classification and selection of test cases should be treated as multi-faceted concept. It will result in reduced cost and efforts of software testing and improve the quality of testing by reducing the total number of test cases executed.

1.11 OBJECTIVES OF THE RESEARCH WORK

Several researches have been carried out to tackle the problem of software testing by considering optimization techniques. Almost all such attempts were based on considering the single parameter for test case optimization. Hence in the present research work detailed investigation on
development of enhanced model for search based optimization testing techniques such as code based testing and model based testing concepts are carried out. Accordingly, the specific objectives of the present research work are as follows:

- To develop enhanced models using search based optimization testing techniques for:

  i. Test Suite Size Optimization – To develop techniques that keep test suite sizes manageable by using Genetic Algorithm combined with Class Based Elitist approach which eliminates the redundant test cases without comprising fault detection effectiveness

  ii. Test Case Selection – To propose Cluster identification technique for effectively performing the test cases selection with increased condition coverage.

  iii. Test Case Prioritization – To propose Group Wise Clustering Algorithm is to improve the test case prioritization by assigning priorities to the selected test cases with two set of values as high and low priorities

  iv. Model-Based Test Suite Size Prioritization – To develop Dynamic Interaction Patterns approach for improving the early fault detection capability and reducing the overall testing time in a system model.

  v. Model-Based Security Test Execution – To determine the risks present in the system states and protect them from vulnerabilities two concepts namely risk analysis using STRIDE threat approach and automated test generation are combined and called as Model-Based Security Testing (MBST) is introduced.
vi. Domain-specific model-based test generation – applying model-based test suite generation for a specific domain with selected features

- To develop various software test models with suitable optimization techniques and improve the effectiveness of testing.
- To compare the results of newly developed above models with the existing methods.

1.12 RESEARCH METHODOLOGY

In the current research work multi objective optimization technique is developed and implemented. Combination of suitable optimization techniques at multiple levels has been attempted. Genetic Algorithm combined with Class Based Elitist approach is introduced to improve the test suite size reduction with increased fault detection ability. Hybrid algorithm has been developed and proposed in order to achieve the test suite size minimization with reduced execution time.

Cluster identification technique is proposed to effectively perform test cases selection with increased condition coverage. Group Wise Clustering Algorithm is proposed in this research work to improve the test case prioritization. Dynamic Interaction Patterns approach is introduced to achieve Model based test case prioritization in the proposed work. STRIDE threat approach and automated test generation are combined and called as Model-Based Security Testing (MBST) is introduced in the proposed work to determine the risk present in the system states.

Object-oriented programs in JAVA and C# are used in this investigation for carrying out experimentation part. Different sets of system
models are considered for comparing the experimentation values of proposed models with existing methods.

The scope of the research work is to enhance software models for real-time system applications.

1.13 ORGANIZATION OF THE THESIS

This thesis is organized in six chapters. Chapter 1 gives a preface to the proposed research work, motivation that led to the research, problem statement and a brief description of the basic concepts underlying this research work.

Chapters 2 describes the theoretical concepts related to this research work, the review of literature in optimization techniques used for testing and recent developments in related approaches.

Chapter 3 discusses the proposed research work with different combinations of optimization techniques applied to code-based testing and analyses the performance by considering various coverage criterions and contains the results which are compared with existing methodologies.

Chapter 4 describes the model-based testing techniques used for optimization by identifying dynamic behavior of a system and performs model-based test prioritization using dependence analysis with the help of system models using C# language. An attempt on application of model-based testing for a mobile domain with some features is presented in this chapter.

Chapter 5 presents security mechanism in model-based testing using STRIDE threat modeling approach and the results are compared with existing risk assessment method in which the proposed combination predicts more number of faults.

Chapter 6 concludes the research work and highlights the scope for future work.