***Class Testing From OCL Class Contract Specifications***

***Using Evolutionary Multi-Objective Genetic Algorithms***



***Supervised By***

***Mr. Atif Aftab Ahmed Jilani***

***Assistant Professor, FAST NU***

***Co-Supervised By***

***Mr. Syed Muhammad Saqulain***

***Assistant Professor, IIUI***

***Submitted By***

***Rehan Frooq***

***(119-FAS/MSSE/F06)***

**Department of**

**Computer Science and Software Engineering**

**International Islamic University, Islamabad**

A thesis submitted to

**The Department of Computer Science and Software Engineering,**

International Islamic University, Islamabad

as a partial fulfillment of requirement for the award of the degree of

**MS in Software Engineering**

Table of Contents

1. **Introduction1**
   1. Motivation2
   2. Introduction2
   3. Software Testing2
   4. Trends in Software Testing3
   5. Testing from Software Specifications3
   6. Model Based Testing4
   7. Multi-Objective Genetic Algorithms4
2. **Literature Review6**
   1. Test Sequences Generation8
      1. Test sequences from State Models 8
      2. Test sequences from UML Models 10
      3. Test Sequences from Software Specifications 11
   2. Test Sequence Optimization 14
      1. As Single Objective Problem 14
      2. As Multi Objective Problem 14
   3. Evaluation of Literature16
      1. Conformance to Standards17
      2. Lack of Automation17
      3. State-based Problems17
      4. The Need and Potential of Optimization18
3. **Problem Statement19**
   1. Problem Statement21
   2. Aims and Objectives21
   3. Research Questions22
4. **Proposed Approach22**
   1. Phases of Proposed Approach24
      1. Parsing of Class Contracts and Generation of Abstract Finite State Machine24
      2. Coding of Test Sequences in Chromosomes and Optimization through MOGA24
   2. Adopting to Standards25
   3. Automation26
   4. Optimization of Test Sequences26
   5. The Fitness Functions27
      1. Calculate Fitness By Coverage27
      2. Calculate Fitness By Validity 29
   6. A new Java based Tool for Research and Industry30
5. **Case Study and Experiment31**
   1. Introduction32
   2. Problems with Previous Approach33
      1. Deviation from Standard OCL Syntax33
      2. Inherent Problems of Exhaustive Finite State Machine Exploration35
   3. Application of our Approach35
   4. Mutation Analysis37
   5. Advantages37
   6. Results and Discussion38
   7. Conclusion and Future Work40
6. **References and Bibliography42**
   1. References43

**List of Figures and Tables**

**Fig. 5.1. A Chromosome of length n, in our coding scheme25**

**Fig. 6.1. OCL Class Contract that does not comply with standard OCL32**

**Fig. 6.2. Actual Parse able OCL Class Contract33**

**Fig. 6.3. Automated MOGA optimized, test sequence generation process36**

**Table 2.1. Summary of Literature Review 16**

**Table 6.1. Mutation Analysis of CoinBox, Stack and Circle Class37**

**Abstract**

Trend of Software has been towards building bigger, more complex and highly reliable systems. These trends turn Software failures into fetal and causing catastrophic damages to human life and wealth. It obviously, becomes extremely important that we must thoroughly test software systems, to be safe prior to being used. Testing of UML Class models from their semiformal OCL specifications can help identification of defects early in the software life cycle. Current approaches suffer from inherent problems of exhaustive exploration of finite state machines (infeasible paths, exponential number of test sequences and uncertainty of completions of testing). Evolutionary algorithms can greatly help by optimizing the test sequences to get optimal coverage and minimal cost.

Our newly devised approach can help improve the testing of UML model based software; by testing the conformance to semi-formal class operation contract specifications (specified in the form of OMG standard, OCL semiformal language). We achieve two main goals (1) Automation of testing process and conformance to standards, of current technique of test sequence generation, bridging the gap between the research and industry (2) Improvements in the state of the art technique through the application of Multi-Objective Genetic Algorithms (MOGA). Our Java based Testing tool, using our new approach, gives Test Engineers, choice of selecting better quality test sequences, optimized in terms of cost and coverage. Automation process makes possible the adaptation to changed class contract specifications in a dynamic environment.

Chapter 1

**Introduction**

## 

* 1. **Motivation**

Since the earliest development of computer program, software has come a long way and through many paradigms. It was journey form a few lines of computer instructions punched on machine readable cards to millions of lines of code to develop high end graphical user interfaces. The trends in the last three decades in software engineering have been to build bigger solutions to bigger and more complex problems, from a single user programs to multi-user, geographically distributed applications with multiple tiers and from an application affecting a few users to the application affecting the humans all over the world. Historical paper titled “No Silver Bullet” written by F.P. Brooks still holds even after 3 decades of its publication [20]. Brooks discusses inherent properties of software like Complexity, Conformity, Changeability and Invisibility.

As the software has grown, became more and more complex and ultra dependable; the need of finding and fixing problems, before the actual deployment and early in the Software Development Life Cycle has grown enormously. Over the past decades the trends in the software development have shifted from being considered as Art of individual programmer towards establishment of Engineering grounds and principles.

A very first consequence of the application of Engineering Principles to the world of Software Development was the thinking of software as a product as any other industry product. This raised questions about quality of the software and introduction of concept of quality.

Quality of software must be tested against the intended behavior as specified by the software requirement specifications. It raises major concerns Firstly, software requirements should be specified so that they could formally be tested against the actually developed software [1]. Secondly, requirement specification techniques should be understandable by software developers and should be close to programming syntax in order to be used in the industry (OMG’s like Object Constraints Language OCL [21]). Thirdly, some techniques should be devised to map these requirements to the actual functionally of the software. Finally, problem domain of testing has unlimited testing combinations for different input variable values so testing sequence should be figured out to the reveal most of the possible errors in the software implementation.

* 1. **Introduction**

This research targets testing of UML class Models from their Class contract specifications. It lies at the intersection of Model Based Testing, Specification Based Testing and Evolutionary Testing, which are subject-areas of Software Testing, area of Software Engineering. It specifically targets Optimization of Test sequences generated from state-based approach, for unit testing of class, form semiformal OCL class contract specifications. This introductory chapter gives the background of our work and its significance in the domain of software engineering and software testing.

* + 1. **Software Testing**

Software Testing is a discipline of Software Engineering which deals with the testing of the software to reveal any errors and indicate the quality of the software.

* + 1. **Trends in Software Testing**

Modern trends in software engineering directly affected the software testing process. Test engineers and test teams today face the challenge of testing large scale systems that might require exponential time and resources while being built and tested, change of requirement is quite often along with risk of wrongly elicited or documented requirements (the tacit knowledge), all these factors point towards a strong need of automaton and optimization for testing approaches. Due to large scale of built software and dynamic stake holder requirements; manual testing of software becomes impossible [17]. Testing of software in turn becomes strong candidate of automation along with a strong need to figure out the ways by which we can efficiently test the software keeping within the limited budgets of time and cost. Many authors have worked towards automation and optimization of testing process as discussed in the literature review section, but there are still may grey areas where we have questions that need to be addressed by the research.

* + 1. **Testing from Software Specifications**

Specification-based testing refers to the area of software testing where software is tested against its specification. It is a type of functional black-box testing where software is tested on its interfaces for the validation against the documented requirement specifications. This discipline deals with generating test suites from the software specification, executing the test case scenarios against the actual software and then checking the results against test oracles. One of the biggest plus of this type of testing is that it allows building of testing environment for the software even before the existence of the software [1], [11], [14] and [18].

* + 1. **Model-based Testing**

Model-based testing is a sub-area of Model-base development and Model drive engineering, where we represent software in terms of models. One of the famous modeling techniques is Unified Model Language (UML) where software is modeled in terms of static (e.g. class diagrams) and dynamic modes (e.g. sequence diagrams) structures. Model-based development lets the engineers to focus on the actual domain specific issue compare to technical issues of software development. A big advantage of Model-based development and Model-based testing is the presence of tools support. Tools are available that can help engineers model software, transform software models from one representation to another and generate abstract test case extracted from the software model. These abstract test cases can then be transformed into actual executable tests [11], [16].

* + 1. **Genetic Algorithms**

Genetic algorithms are random search based form of heuristics. They mimic the actual process of evolution; they are also referred as simulated genetic algorithms. As the theory of evolution states that living things get improved generation after generation and adopt better quality genes. While using genetic algorithm for a problem optimization the very first step is representation of the potential solutions in terms of chromosomes. Each chromosome consist of number of genes, genes are part of a solution. Together these genes and chromosomes form the population of possible solutions. MOGA tools execute Genetic algorithms, applying genetic operators on the input population.

The evolution process involves following steps:

* *Initialization* of the population, random or from some input.
* *Selection* of fittest individuals based on their calculated fitness values.
* *Reproduction* of the selected individuals.
* *Termination* of the evolutionary process based on selected criteria e.g. n number of generations or *f* target fitness values.

Reproduction involves application of one of the two genetic operators based on probability.

* *Crossover* is a genetic operator where two or more than two solutions are combined to form resulting child solution. A number of techniques for crossover are available in the literature. The simples is called “*single point crossover*”, a part of first chromosome and the remaining part of the second is taken and combined to produce resulting child chromosome.
* *Mutation* is the random change in part of a chromosome that results in a new individual with properties different from the parent. Depending upon the probability one or more of the genes can be change at random by the GA execution mechanism.
  + 1. **Multi-Objective Optimization through Genetic Algorithms**

Traditionally GAs has been used as a search heuristic for finding optimal set of solutions to problems involving single objective. Recent advances in the field suggested usage of GAs for multi objective optimization. In principle Multi Objective GAs are the same GA based tools, but the potential solutions are evaluated for multiple parameters and their fitness values are evaluated by multiple fitness functions. MOGA evolution process then involves comparison of the multiple fitness values of candidate chromosomes. Multi Objective optimization is particularly used with problems where no objective can be optimized without sacrificing the quality of the competitive objective(s). The solutions so generated are referred as Pareto-Optimal solutions. Test sequence optimization involves trade-off between testing cost and achieved test coverage; hence the process is a strong candidate of Multi-Objective Optimization [17].

* 1. **Amis and Objectives**

Thisthesis targets improvement of the class specification based test sequence generation process

1. Apply current state of the art test sequence generation technique to the industry standard OCL class contract specifications.
2. Automation of the current technique of test sequence generation from OCL class contract specifications.
3. Improvement of the current specs based test sequence generation technique by application of search-based techniques of Multi Objective Genetic Algorithms
   1. Optimizing the test Coverage achieved by the generated test sequences.
   2. Minimize the number of infeasible test sequences.

# 

**Chapter 2**

**Literature Review**

It is notable that the work is diverse in nature and spans across boundaries of the We have divided the review in sections as the work on test sequence generation, test sequences generation from formal specification (especially form operation contracts) and test sequence optimization.

## Test Sequences Generation

Generation of test sequences (synonymous to test cases) is one of the toughest tasks for a test engineer. This testing phase involves trade-offs between number of test cases and the desired test coverage, number of test cases and available resources, quality of test cases and achieved coverage etc. This kind of process can be tiresome if done manually.

## Test sequences from State Models

*Ruilian Zhao et al* aim to develop the infrastructure of automatic test data generation for EFSM models that produce real data to trigger feasible transition paths. It also provides empirical results on efficiency analysis of test data generation for a set of state-based models. in this paper, a GA-based system is presented to automatically generate test data for feasible transition paths in EFSM models [3].

*Karnig Derderian et al* present an approach for automated Unique Input Output (UIO) sequences generation for finite state models. They take sequence generation problem as a search problem and generate test sequences based on Genetic Algorithms (GA). They use 11 real and 23 randomly generated FSMs as proof of concept experiment. They also state that the problem of test sequence generation from an FSM is an NP-Complete problem. The presented experimental results show that GAs give result between the ranges of 62% better to at least as good as random search. They also propose a new fitness function for evaluating fitness values of UIO test sequences and claim that it is performance wise better than the previous approach. They suggest that at small FSMs random search seems to outperform GA but for bigger FSM models GA are a far better approximation [13].

## Test sequences from UML Models

UML diagrams model static and dynamic aspects of a system, techniques fond in the literature in general use one of the static diagrams to represent the static structure and one of the dynamic diagrams to represent dynamic behavior of the software; in order to generate test sequences / test cases.

*S. Asthana et al* have given an approach for generating test cases from class and sequence diagrams the claim is that this is the novel approach which uses test cases from class and sequence diagrams without transforming them into any intermediate model. The approach claims that use of any intermediate form is avoided by the approach from specification model to actual SUT, but XMI itself seems to be an intermediate form used for representation of the model [6].

*Chen Mingsong et al* present an approach of test case generation from UML activity diagrams. In their approach they compare the dynamic behavior of the activity diagram to the actual program execution and in this way the activity diagram behaves as a directed graph. They use three test selection criteria activity coverage, all transition coverage and simple path coverage. Code instrumentation is used for recording test data and the test logging statements are inserted into the program itself. This approach is a white box testing approach because it needs access to the program source for testing [12].

## Test Sequences from Software Specifications

*Atul Gupta* discusses an approach where class contracts are used to test class method interactions. The approach is state based approach. Using an abstract state configuration of class and initial abstract states, reachable states are incrementally generated by searching for the methods which can be invoked in the current state and resulting abstract states are computed. The thing which is lacking in the approach is that it lacks automation and syntax used does not conform to the industry OCL standards and fails even to get parsed by standard OCL parsers. The approach uses AFS traversal to generate test sequence paths, hence faces inherent problems of finite state traversal which we discuss in detail in our pitfalls section.

This is the core reference paper used by us and hence we give a brief over view of the approach here. Author has used the traditional searching approach for path traversal of finite state machines and all transition coverage is uses as sequence path generation. A specification based testing approach is proposed, which uses class contracts specified in the form of OCL constraints (class invariants, pre conditions and post conditions). They build an abstract state configuration for the class under test, for each initial abstract state, corresponding reachable states are incrementally generated by traversing and searching for the methods which are invoke-able in the current state and resulting abstract states are generated. Author argues that state of an object, being specified by values of its variables can lead to state explosion and hence notation of “abstract states” is introduced having abstract object variable values.

Applying Transition Tree Coverage

Abstract State Machine 🡪 Test Sequences

Testing sequences were generated using Transition Tree coverage and Modified Transition Tree coverage (by including additional test by for invalid inputs). The thing which is lacking in the approach is that it is still not automated (author himself mentions that in the conclusion section) and no tool has been suggested for automating the suggested process [1].

*T Miller and Paul Strooper* present a case study on specification based implementation testing frame work. They have used Possum animation tool and Sum specification language for modeling and specification of GSM 11.11 standard of mobile communication. They claim that the framework gives almost equal performance compared to BZ-Testing tools and more cost effective than manual testing. Authors suggest stepwise generation of a directed graph and then paths through that graph are the test sequences [11].

*Marie-Claude Gaudel* presents an approach for generation/ selection of test data from the formal specifications. An exhaustive test set based on the formal specifications and their correct implementation is proposed. After that selection of a finite test set is proposed based on domain specific selection hypothesis. Author presents result of case studies of application of the approach to algebraic specifications in the form of LOTOS based specifications of ISO OSI based protocol specifications. A big constraint in the application of this approach may be of manual work involved in order to decide to “Selection Hypothesis” that varies from domain to domain and specifications to specification [15].

Planning and execution of tests involves the analysis of the functionality of software (functional specs), what are the inputs and outputs of the software and its execution environment. This process is difficult, time taking and technically sophisticated. Role of a tester requires him/her to have programming skills, grip on formal languages like OCL, mathematical theory of graphs and good understanding and comprehension of computer algorithms. [10]

Literary survey reveals that most of state of the art research targets test sequence generation using UML static diagrams(class diagram) and UML dynamic diagrams (Sequence diagram and State Charts). UML diagrams are not sufficient enough for specifying complete class behavior, most accurate details of a class are revealed from the OCL class specifications in the form of OCL Class Contracts [1].

Test sequences generated using the OCL class contract specifications using state-based technique suffer from their inherent problems including infeasible-paths and exponential number of generated test sequences. In this research we try to figure out solution to these problems besides have automated and optimize the test sequences generated from the OCL Class contracts specifications. Multi Objective Genetic Algorithms are used to overcome the issues by their power of search based multi objective optimization as discussed in [2], [4], [7] and [9].

## Test Sequence Optimization

*Shukatl Ali et al* preset a systematic review of search-based test case generation techniques. The plus is a comparison of different Meta Heuristic Search (MHS) algorithms being employed in search-based testing of software. They have assessed 450 papers out of 6 research repositories. They conclude that Genetic Algorithms are promising for problem solving in the domain of software testing [7].

## Single Objective Optimization

*Mark Harman et al* propose three search-based algorithms for test data generation and preset the result of a case study for the application of their approach. The claim made by authors is that their approach can maximizes the coverage and minimizes the number of test cases generated. The size of the software considered for case studies is as big as 144 lines of code, which might be good for a proof of concept [4].

*Andrea Arcuri et al* focus on comparison of 3 test automation strategies namely Random Testing, Adaptive Random Testing and Search-based testing using Genetic Algorithms and present their results. They present a comparative analysis of the approaches and present the results of experiment on 3 SUTs [5].

*S.K. Prasad et al* present GA based approach for test data generation and they present their algorithm that takes the user input variables and using GA generates test data. They claim that GA outperforms random testing on time measures [7]. In another paper S.K. Prasad et al present another search-based test sequence generation technique using Ant Colony optimization algorithm where “Ants” are used to explore CFG to find optimized test sequences [8].

Compared to competitive optimization techniques, GAs, instead of searching a solution by heuristic search methods, start with a random set of possible solutions and then improve the solutions by simulation of evolutionary processes of crossover, mutation and selection. This process is repeated generation after generation. That way an optimized set of solutions is guaranteed, which can always be improved further by subsequent GA implementation, as the optimization techniques give optimal solution(s) because exact solution is not available [3].

GA techniques are independent from the problem domain; this is quite helpful for general purpose optimization of the problem, because the GA implementation takes encoded representation of the problem and yields the optimized results irrespective of the problem at hand. Being random search algorithms, they avoid convergence to local minima and the solutions are quite evenly distributed across the problem domain.

## Multi-Objective Optimization

*Thaise Yano et al* present an approach of test sequence generation using Evolutionary Algorithms. The claim that search based approaches till then had been mostly proposed for white-box testing. The paper presents, an evolutionary approach for test sequence generation from a behavioral model, in particular, EFSM. A multi-objective evolutionary algorithm, M-GEOvsl adopted from M-GEO is used, that can consider two objectives: to search for a test sequence that covers a target transition, as well as to minimize the length of this test sequence [2]. Authors present an approach of test sequence generation using Evolutionary Algorithms. They claim that search based approaches till then had been mostly proposed for white-box testing. The paper presents, an evolutionary approach for test sequence generation from a behavioral model, in particular, EFSM. Problem of Infeasible paths generations is covered by executable model. Transition of interest coverage criterion is applied using Evolutionary Algorithm. System is modeled in form of EFSM. Challenges listed by the authors while generating test for EFSM. An Evolutionary Algorithm is also proposed, based on Pareto optimality. Each solution is non-dominating, that is, it can’t be improved in any objective without causing degradation in at least one other objective. Future work of the authors suggest improvements like addressing the limitation of the approach when there are no slices of a model are found and validation of the approach is demonstrated by an experiment but they sate that they are carrying out further experiments for the validation of the approach [2].

Multi Objective Genetic Algorithm (MOGA) go one step further, they support optimization for multiple objectives, in our case optimization for two objectives, minimize the number of test sequences and maximize achieved test coverage of the test sequences is required[2].

MOGAs have a very good support by open source tools like Java Genetic Algorithm Package (JGAP), JMetal (a multi-objective GA implementation tool) and Java API for Genetic Algorithms (JAGA) [12],[13] and [14]. These and similar tools, being used in the industry and research, it makes them more practical to be used for the practical test sequence optimization for industry usage.

* 1. **Literature Evaluation**

Approaches and techniques in the literate has different problems here we discuss these identified problems as found the literature.

* + 1. **Conformance to Standards**

Class contract based test sequence generation technique found in the literature [1] does not conform to industry standard OCL syntax so it makes the process impractical, while being adopted by industry practitioners. Due to the same reason current technique lacks automation. The first phase of the research focuses on adopting the technique to work on standard OCL syntax. We take standard OCL syntax specs and apply the test sequence generation technique to get the output test sequences. This kind of test sequence generation approach is state-based as discussed [1].

* + 1. **Lack of Automation**

Approaches found in the literature either don’t provide any automation at all (assuming the input in a predefined state) or Fail to comply with the state of the art industry standards like e.g. deviation for the standard syntax It makes it hard for test engineers to used these techniques Software requirements and hence specification are quite often volatile, automation can be a great help to regenerate the test sequences from new specifications.

* + 1. **State-based problems**

A large number of possible test sequences may require exponential time and effort for the testing process itself. Unfortunately resources and time are limited for the Software Development Lifecycle (SDLC). Many of the state-based generated test sequences might be Infeasible, repetitive, reoccurring possibly several times or might not be required at all. It is not practical and, in general, impossible to asses all the possible test sequences of program flows due to effort and time required for execution. There is always a tradeoff between number of generated test sequences (cost) and the achieved test coverage (coverage). It is quite difficult for a machine to evaluate all test sequences within a reasonable amount of time. Exhaustive testing of all the test sequences is impossible.

* + 1. **The Need and Potential for Optimization**

Being state-based the technique suffers from inherent problems of state-based test sequence generation techniques [2], [3] and [4], they can be improved by applying search-based optimization techniques. Multi Objective GA’s are promising for the improvement where we can remove infeasible test sequences using multiple fitness functions to achieve maximum test coverage in minimum number of test sequences.

The next phase is to optimize the generated test sequences using Evolutionary Genetic Algorithms using a multi objective approach where we have two conflicting objectives first to minimize number of test sequences and second to maximize test coverage of generated test sequences.

Approach discussed by [2] for test data generation using GA but a similar approach can be used in our case for generating test sequences using Multi Objective Genetic Algorithms.

Table 2.1 Summary of Literature Review

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Authors** | **Automation** | **Specification Based** | **Coverage** | **State Based** | **Optimization** | **Multi Objective** |
| Atul Gupta [1], Springer (2010) | Automation is hard due to non-standard OCL syntax | Yes, from OCL specifications | -Transition Tree Coverage | Yes, suffer from State-based problems | X | X |
| Marie-Claude Gaudel [14], Springer (2001) | Semi Automation | Yes, LOTOS based proof of concept on Possum. | -All paths coverage | Yes, test sequences are generated from directed graph | X | X |
| Thaise Yano et al [2], ICSTW, IEEE (2010) | Partial automation does not discuss in which form the model will be taken. | X | -Target Transition Coverage | EFSM is used as an intermediate form | Yes, an Optimization Algorithm, which is strictly not GA based. | Yes |
| Mark Harman et al [4], ISCTW, IEEE (2010) | X | X | X | Yes an EFSM based representation is used | Three Test data optimization algorithms are proposed | X |
| S. Asthana et al [6], Springer (2010) | Automation without using an intermediate model. | X | X | Yes , and claim to have avoided state space explosion because their model is executable | X | X |
| S.K. Prasad et al [8], ICISTM, Springer (2009) | Claim automatic approach for generating test data | X | X | X | Yes optimization through single objective GA | X |
| S.K. Prasad et al [9], ICISTM, Springer (2009) | Automation of test sequence generation process is claimed. | X | -All state coverage | Yes, FSM is generated. | Yes ,Ant Colony based Optimization | X |
| M. Prasannan and K.R. Chandran [10], ICSRS (2009) | Automation of test case generation is claimed. | X | X | General Tree and Tree Structure are built and depth-first search gives test sequences | Cross Over of GA is used, but optimization is not mentioned. | X |
| Chen Mingsong et al [12], ACM (2006) | Yes from UML Activity Diagram | X | -Activity  -Simple Path  -All Transition | Yes, UML Activity Diagram as Directed Graph. | X | X |
| K. Derderian et al [13], ACM (2006). | Yes | X | X | Yes, Approach is specifically for FSMs | Yes, GA based optimization | X |

# Chapter 4

# The PROBLEM STATEMENT

# 

## Problem Statement

Model Based Testing involves automation of testing process. Building a model of System Under Test (SUT) and then generation, execution and evaluation of Test cases for SUT. Operation contracts specify the class behavior in terms of invariants, pre and post conditions, these class contracts are bindings that SUT must conform to. An obvious advantage of using class contracts is that they can be written in form of semiformal OCL constructs which are more precise compared to the natural language specifications and also can be easily converted to a machine readable form. The survey of literature reveals that class contracts have potential of revealing the test sequences for the unit testing of classes [1], but to-date very little work has be done in this direction. State of the art approaches also lack automation and conformance to industry standards.

Search based optimization algorithms on the other hand have been employed widely in the field of MBT but to date there is no evidence of their application for test sequence identification from class contract specifications. Optimization techniques are promising for optimization of number and quality of test sequences by overcoming the state space explosion problem.

## Research Questions:

This research answers the following questions:

1. **How we can improve the Unit Testing of Class Models using OCL class Contract specifications in terms of compliance to industry standards and automation of the Test Sequence Generation Process?**
2. **How state of the art techniques of Optimization (Evolutionary Genetic Algorithms) can be applied to the problem of determination of Test Sequences based on OCL Class Specifications to achieve reduction in number of infeasible test sequences and improvement in test coverage?**

# Chapter 5

# Proposed Approach

* 1. **Phases of Proposed Approach**

Our approach improves the previous approach in a number of ways; here we explain the actual functionality of our approach and the advantages achieved.

* + 1. **Parsing of Class Contracts and Generation of Abstract Finite State Machine**

We use standard OCL syntax and build the test sequences from the generated Abstract Finite State Machine. We use standard OCL parser [22] for generating OCL pares tree of input class contracts. This parser is frequently used with Eclipse IDE for Java [25] for parsing of OCL constraints on UML Models.

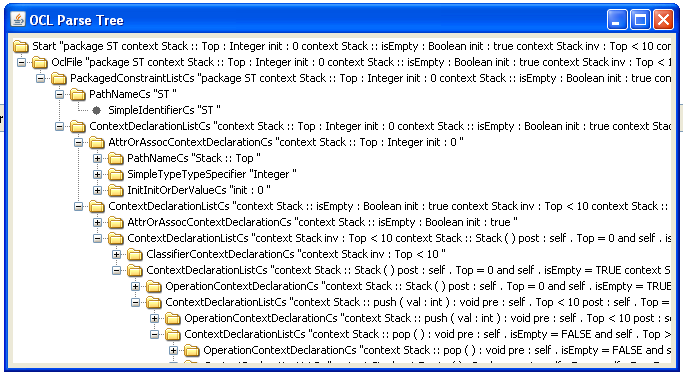
****

Fig. 5.1. Sample Partial Parse-Tree of OCL Operation Contract for Stack Class

After generation of parse tree is the process of semantic analysis of the output parse tree and construction of domain specific objects in Java. Our OCL parse tree processor transverses the parse tree and extract the Objects corresponding to the domain concepts of OCL semantics.

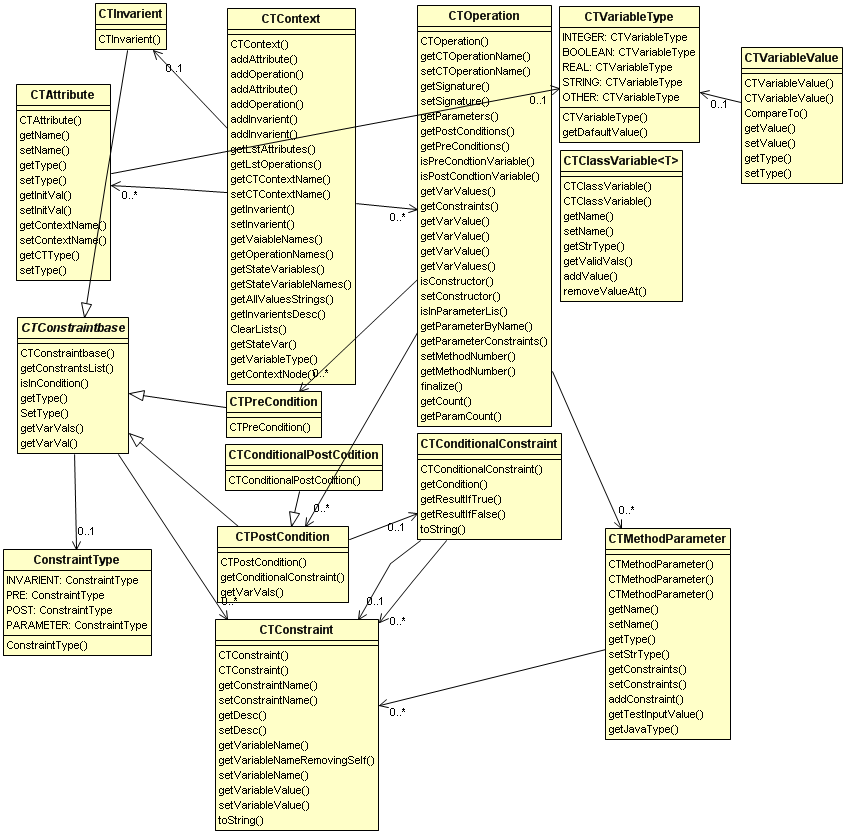


Fig. 5.2. The Class Diagram of Mapping Objects of OCL Operation Contracts

After generation of parse tree the next step is of constructing the abstract finite state machine applying the rules used by [1]. The abstract state model of the software from specification is created starting from the class constructors. For each constructor a new initial state in the Abstract Finite State Machine is created. We then dynamically create all the resulting state form the initial state onwards.

A deviation here is that previous approach suggest, using transition tree coverage criterion i.e. test sequences are identified along with the simple paths. Simple paths coverage misses the self reference transitions and it is quite possible that a method might fail on subsequent invocations as the subsequent calls might bring the object in as state (due to implementation faults) that it may behave anomalously; even the specifications may suggest some other behavior.

But in case we have self transitions to a state then it might skip a valid step in the sequence of method calls. So it is better if we have row test sequences from exhaustive search of the AFSM. The test sequences generated in this step are used as an initial population for the MOGA optimization.

* + 1. **Coding of Test Sequences in Chromosomes and Optimization through MOGA**
    2. **The Multi Objectives**

The test sequence generation process should be efficient enough to reveal the problems in the implementation. In order to get quality test sequences we use two objectives they are not totally in a conflict but optimization for one might decrease the fitness of the other objective our two optimization objectives are

* + - 1. **Optimize Transition Coverage**
      2. **Test Sequence Validity Optimization**
    1. **Coding of Solutions in Genes and Chromosomes**

We suggest a coding scheme where a potential solution (Chromosome) comprises of Transition (Method Call) from the built Abstract Finite State Machine. Eeach Method Call represents a transition in the Abstract Finite State Machine with additional feature to be automatically executable on a class by calling the method represented by this transition.

We user the name method call so that it is more close to the testing domain and is quite easily understood by test engineers, compare the FSM domain specific word ‘Transition’. This is not a binary coding were each gene is coded in terms of a binary representation. So a Chromosome of length n will have n method calls objects (genes). The JGAP tool used by us allows specifying a mechanism of returning custom random genes values while population is evolved for the purpose of mutation. To tell the MOGA implementation system how to get random values we devise a mechanism which returns random transitions from the generated finite state machine. Each chromosome in our solution set can be visualized as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MC1 | MC2 | MC3 | .... | MCn |

Fig. 5.3 A Chromosome of length n, in our coding scheme.

Where n is the length of the chromosome and MCi is the ith method call in the test sequence and i = 1,2,3,…n. during the MOGA optimization mutation and cross over is applied on the genes. While during mutation the changed gene value is randomly selected transition from the built Abstract Finite State Machine.

* 1. **Expected Benefits**

Proposed approach gives following benefits over current approach of test sequence generation and optimization.

* + 1. **Adopting to Standards**

The Object Management Group (OMG) has clearly defined specification for standard OCL syntax. As this thesis is written the current version is 2.3.1 as of January 2012, which is available on OMG’s website for download. When OCL class contract, presented in the literature, is compared to the industry standard OCL syntax; it is revealed that, current approach deviates from the state of the art OCL used in the industry. As a matter of fact no standard OLC parser accepts the syntax used in the literature. Current syntax is more C++ like which is not acceptable as OCL according to the OMG OCL specifications. Since the approach deviated from the standards, it was quite unlikely to be adopted by the industry practitioners. In order to build our testing tools we observed failure of parsing for the current syntax. So we took the OCL Class contracts in standard OCL form and then applied current approach to it for building Abstract Finite State Machine (AFSM). Then we generated the test sequences by traversal of AFSM, beginning from the start state. We call these test sequences as raw test sequences because they suffer from the state-based path search problems. Now our approach is able to generate test sequences directly from the OCL.

* + 1. **Automation**

Conformance to standards provides the benefit of automation for the process of test sequence generation. Reading OCL class contract specifications, we automatically construct OCL parse tree. After that our tool does semantic analysis of constructed OCL pares tree and applying the rules defined in the literature build corresponding AFSM automatically. Next step is automatic generation of the raw test sequences from exhaustive search of AFSM, these test sequences can be directly used by the test engineers if they think raw test sequences test sequences are good enough and can be used without optimization. In case when test engineers decide to go for Multi-Objective GA based optimization for the test sequences, our tool automatically run MOGA over the raw test sequences selecting a random population out of them. This way the process of generating test sequences from standard OCL is automated all the way to the MOGA optimized test sequences.

* + 1. **Optimization of Test Sequences**

The newly suggested approach is a novel approach that uses multi-objective GA for test sequence optimization, using an initial population of randomly selected exhaustive search test sequences. State of the art approaches found in the literature use random stochastic initial populations. By nature of MOGAs, use of complete random sequences, gives a big chance of getting the population evolved in a negative direction, because while optimizing test sequences it is quite important to have a valid sequence of method invocations. Due to random nature of GAs, such a population for test sequence generation might be disaster. Starting with valid set of values and applying MOGA using our fitness functions yield more useful test sequences.

* 1. **The Fitness Functions**

In order to optimize the test sequences through MOGA, the role of efficiently defined fitness functions is critical. The MOGA based tools used these user defined fitness to assess the quality of solutions (the chromosomes). The simulated genetic process of evolution, assigns the fitness values to the chromosomes for each generation and after application of genetic operator only fittest chromosomes are selected for subsequent generations. In order to be used with our optimization we have devised the following fitness functions:

* + 1. **Calculate Fitness By Coverage**

Calculates the coverage of current chromosome by the number of transitions covered and assigns the fitness value according to the following algorithm

Description of calculation of coverage weights is

* If a transition is covered once chromosome is given additional positive weight-age, it rewards a chromosome for covering a transition.
* If a transition is not covered at all by a chromosome it is given additional negative weight-age, it reward a chromosome negatively for not covering a transition.
* If a transition is covered more than once by a chromosome it is given additional negative weight-age, it rewards negatively due to repetition.

The fitness value for a chromosome by coverage can be calculated by the following pseudo code:

*Initialize CF:=0, wCoveredOnce, wCoveredTwice, wCoveredMoreThanTwice*

*For each Chromosome c in the current population*

*For each Gene g in c*

*If g occurs once*

*CF = CF + wCoveredOnce*

*End*

*If g occurs twice*

*CF = CF + wCoveredTwice*

*End*

*If g occurs more than twice*

*CF = CF + wCoveredMoreThanTwice*

*End*

*End*

*Set coverage fitness of c equals CF*

*End*

*wCoveredOnce, wCoveredTwice, wCoveredMoreThanTwice are, problem specific arbitrary weight-ages, for at least one state coverage, a state covered twice and a state covered more than two times.*

* + 1. **Calculate Fitness By Validity**

To be a valid test sequence the chromosome has to have method calls in valid sequence of invocation. We get fitness value as weighted sum of all individual fitness values of each gene of a chromosome.

Mathematically the fitness by sequence validity for a chromosome is calculated as

Description of the weight calculation for validity fitness is

* Initial state weight,if the first gene of the chromosome has an initial state of AFSM as from state then this weight is added else skipped.
* Sequence weight for call sequence,we calculate the quality of chromosome by the sequence of method calls and reward each chromosome by following formula
  + If any of the method calls (genes) is in a valid sequence then a positive weight is added to the second fitness value.
  + If any of the method calls (genes) is not in a valid sequence then a negative weight is added

The fitness value for a chromosome by validity can be calculated by the following pseudo code:

*Initialize VF:=0, wSState, wInSeq, wNotInSeq*

*For each Chromosome c in the current population*

*If c starts with an initial state*

*VF = VF + wSState*

*End*

*For each Gene g in c*

*If g is in sequence*

*VF = VF + wInSeq*

*Else*

*VF = VF + wNotInSeq*

*End*

*Set validity fitness of c equals VF*

*End*

*wSState, wInSeq and wNotInSeq are, problem specific arbitrary weight-ages, for starting with initial state, being in sequence and not being in sequence respectively.*

* 1. **Java based Tool for Research and Industry**

While working on the research we have came up with a new tool which can be used as baseline for research in FSM based testing. The tool is now open source and freely available for subsequent researchers. This tool can build, save and load FSMs and run MOGA with custom fitness functions for generating optimized test sequences.

# 

# Chapter 6

# Case Study and Experiment

* 2. Introduction:

Generation of test sequences is a critical part of testing phase of software development life cycle. It is show by [1] that test sequences for unit testing of a class can be generated from OCL class specifications, that is by mapping class specifications (OCL class contracts) to the Class Model (specifically a Class in the class diagram).

Current test sequence generation process when applied to actual testing reveals some critical issues, these issues and our proposed solution is presented in this case study. CoinBox class is picked from a Drink Vending Machine’s class diagram; this class is responsible for keeping record of number of available drinks and number of quarters entered by the customer. We used this class because it was used by the reference paper; it helps us to present a comparison. Two more classes Stack and Circle were tested.

Current approach deviates from the actual OCL standards in terms of syntax and semantics. Due to the lack of conformance to standards of OCL the approach lacks the ability of automation. Due to state space exhaustive search the technique has inherent problems of the approach.

## Problems with previous Approach:

When we applied the previous approach to the generation of test cases from CoinBox class we observed the following problems

## Deviation from standard OCL Syntax:

The example OCL code used by an author [1] is not according to the Industry standard OCL syntax and hence none of OCL parsers used in the industry accepts this syntax e.g. the OCL example used is in the following format [OCL Specs 11]:

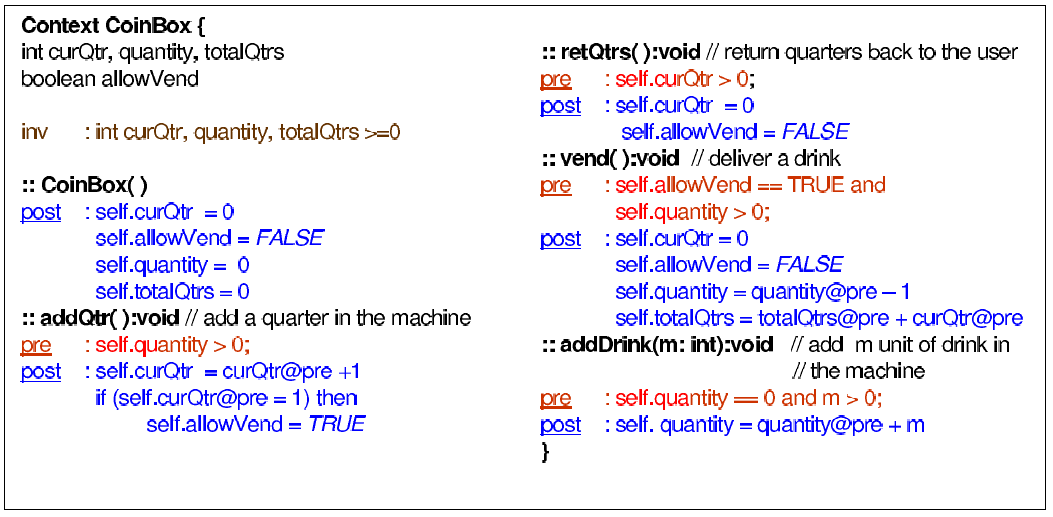


Fig.1. OCL Class Contract that does not comply with standard OCL

The above example was not according to the OCL standards syntax and after modification/adaptation we get the following OCL class contract that is acceptable according to the OCL 2.0 standard:

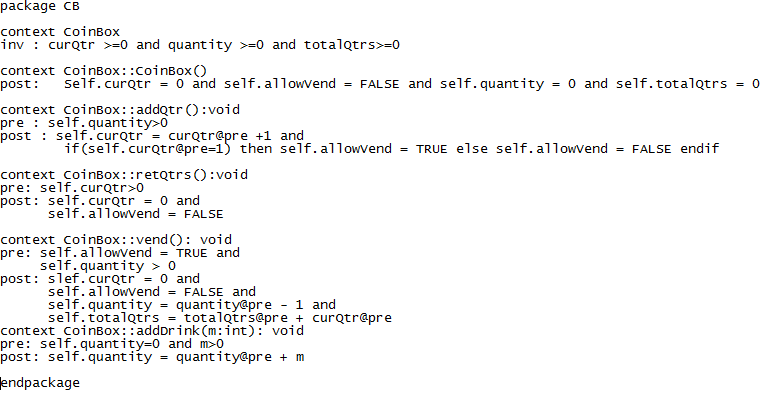


Fig.6.2. Actual Parse able OCL Class Contract.

As the used syntax deviates from the standard OCL in many aspects like e.g. [11]:

* Each statement in each pre and post condition is joined by a logical operator e.g. ‘and’, which is missing in the example.
* Standard OCL syntax does not allow the use of curly braces ’{}’ around the context declarations.
* All the OCL contexts (equivalent to Class) must be declared inside a package and endpackage statement.
* Each constraint in the Invariant declaration must be separated by ‘and’ instead of ‘,’.
* Writing just ‘::’ operator while declaring a method signatures is not enough, it should be fully qualified with the context name being referred by the method.
* Each if must have an accompanying else in order to valid OCL statement.

## Inherent Problems of Exhaustive Finite State Machine Exploration

Test sequences generated using the OCL class contract specifications using state-based technique suffer from their inherent problems including infeasible-paths and exponential number of generated test sequences. So we might get exponential number of test sequences which might also be of indefinite length. Use of these sequences might take exponential time for execution and even them we might ne be sure if they cover even all the states of the object, along with that it is quite possible that a state is covered indefinite number of time e.g. if a state has a method loop (transition to itself with a method) or if a state is revisited again and again.

## Application of our approach

Our approach works in following steps:

1. Generation of OCL Parse Tree: In our approach we take OCL class contract in the form of .ocl (a text file) and generate the parse tree for that passed file using an Industry standard OCL parser. At the moment we use Dresden OCL Parser [11]. This is a popular tool available both as standalone distribution and as an Eclipse integrated plug-in.
2. Semantic Analysis and Generation of AFSM: From the constructed parse tree, by semantic analysis of the tree and applying rules of the previous approach [1], an Abstract Finite State Machine is constructed.
3. Creation of initial population of Test Sequences by Exhaustive Search: From the constructed AFSM Using these “Raw Test Sequences” we select an initial population for Multi-Objective Genetic Algorithm. It is observed that the MOGA performance is highly dependent on the fitness functions used. The detailed process is shown in Fig.3.

Standard OCL Parser

OCL Class Contracts

OCL Semantic Analysis

OCL Parse Tree

Abstract Finite State Machine

Exhaustive Search

Multi Objective Genetic Algorithm

Raw Test Sequences

Optimized Test Sequences

Fig.6.3. Automated MOGA optimized, test sequence generation process.

## Mutation Analysis

We used mutation analysis for bench marking the performance of our approach. We used Mu Java for seeding faults in the classes under test.

## 6.1. Mutation Analysis of CoinBox, Stack and Circle Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Under Test** | **Total Faults Seeded** | **Faults identified by Previous Approach** | **Faults Identified by New Approach** |
| CoinBox | 117 | 81 | 97 |
| Stack | 63 | 49 | 49 |
| Circle | 98 | 55 | 73 |

In this analysis predefined number of faults was seeded in the compiled class files. These faults were based on predefined mutation operators. The experiment reveals that our approach either out-performs the previous approach or at least gives equal fault revealing efficiency .

## Advantages

Our improvements give following benefits to the research and industry community:

* Automation of the test sequence generation process, now test sequences can be generated directly from the OCL specifications of a class automatically.
* Test sequence generation even before the implementation of the software is ready.
* Helpful visual representation of generated Abstract Finite State Machine.
* Improved test sequences with specified length and number, we produced optimized test sequences of a certain length and having the maximum coverage of the states of the class.
* Fine Tuned fitness functions, fine tuned specifically for Test Sequence Optimization process.
* Less time and few resources required due to optimized test sequences, more reliable results because in exhaustive searching of class states we may never know how effective our testing is and when to stop.

## Results and Discussion

As observed in the experimental case study exhaustive state space search generated 872 test sequences of maximum length 26 with redundant test sequence loops. Unnecessary effort needs to be spent on executing all these test sequences. Application of MOGA with population size 10 and sequence length 15 gave 10 Test sequences of length 15 each which were optimized for all sate coverage and valid sequence paths over 500 MOGA generations. Since we used a random population out of the search based sequences, it minimizes the chances of bad genes and evolution in negative direction

By Nature, as of all optimization techniques, we are never expecting that we might have exact solution, but we get optimized solutions. MOGA being a subset of evolutionary algorithms start with a possible set of solutions and try to optimize the set of solutions generation after generation. Evolution as a mimicry of the natural process of evolution might not find suitable chromosomes (e.g. due to mutation) and might give some useless test sequences. This is obvious because the in nature if wrong genes gets to the next generations then the individuals may suffer from defects. have After generation of AFSM we can:

* Either generate a stochastic random population where each chromosome is a constituted out of a completely random set of genes
* Or get a random population out of the population of test sequences generated from state-based test sequence generation approach

Second option seems to give far better results.

While specifying MOGA Fitness Functions for Test sequence Optimization we must take into account the sequence of Genes while calculating fitness values. Our approach gives improvement in terms of Automation of test sequence generation process. MOGA are quite effective while being used for test sequence optimization process but we recommend use of raw test sequences as initial population. MOGA optimized test sequences give optimized coverage (all state coverage) within limited test sequence length and numbers. Caution should be taken while devising Fitness functions for Test Sequence optimization because random nature of GAs (especial mutation) might deviate from the sequence.

# Conclusion and Future Work

In this research we have devised an automated and optimized approach for test sequence generation from OCL class contract specification. The new approach gives us benefits of optimization of test sequences in terms of minimum number and higher quality along with automation of test sequence generation process and conformance to industry practiced OMG standard OCL syntax,. It can save the time and resources spent on a part of testing process where selection of test sequences is done.

We also presented a tool which can be used with any type of finite state machine while applying GAs and MOGAs not just limited to the Class testing from OCL specifications. Our approach gives improvement in terms of Automation of test sequence generation process.

MOGA are quite effective while being used for test sequence optimization process and our use of raw test sequences as initial population appears to give far better results compared to the random selection of test sequence genes. MOGA optimized test sequences give optimized coverage (maximum transitions coverage) within limited test sequence length and numbers.

Caution should be taken while devising Fitness functions for Test Sequence optimization because random nature of GAs (especial mutation) might deviate from the sequence. By Nature, as of all optimization techniques, we are never sure that we have exact solution, but we get optimized solutions. Evolution as a mimicry of the natural process of evolution might not find suitable chromosomes (e.g. due to mutation) and might give some useless test sequences. After generation of AFSM we can: Generate a random population or get a population from raw sequences, second option seems to give far better results. While specifying MOGA Fitness Functions for Test sequence Optimization we must take into account the sequence of Genes while calculating fitness values.

# 

# References

**References**

1. Atul Gupta, “An Approach for Class Testing from Class Contracts”, Springer 2010.
2. Thaise Yano et al, “Generating Feasible Test Paths from an Executable Model Using a Multi-Objective Approach”, *ICSTW*, IEEE (2010).
3. Ruilian Zhao et al, “Empirical study on the efficiency of search based test generation for EFSM models” *3rd International Conference on Software Testing, Verification, and Validation Workshops*, IEEE (2010).
4. Mark Harman et al, “Optimizing for the Number of Tests Generated in Search Based Test Data Generation with an Application to the Oracle Cost Problem”, *ISCTW*, IEEE (2010).
5. Andrea Arcuri et al, “Black-Box System Testing of Real-Time Embedded Systems Using Random and Search-Based Testing”, IFIP International Federation for Information Processing (2010).
6. S. Asthana et al, “A Novel Approach to Generate Test Cases using Class and Sequence Diagrams”, *IC3*, Springer (2010).
7. Shaukat Ali et al, “A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation”, *IEEE Transactions on Software Engineering* (2010).
8. S.K. Prasad et al, “Optimization of Software Testing using Genetic Algorithm”, *ICISTM*, Springer (2009).
9. S.K. Prasad et al, “An Ant Colony Optimization Approach to Test Sequence Generation for Statebased Software Testing”, *ICISTM*, Springer (2009).
10. M. Prasannan and K.R. Chandran, “Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm”, *Int J. Advanced Soft Computing* ICSRS (2009).
11. T. Miller and P. Strooper, “A case study in model-based testing of specifications and implementations”, Wiley Internet Science (2007).
12. Chen Mingsong et al, “Automatic Test Case Generation for UML Activity Diagrams”, ACM (2006).
13. K. Derderian et al, “Automated Unique Input Output sequence generation for conformance testing of FSMs”, *The Computer Journal*, ACM (2006).
14. Konak et al, “Multi-objective optimization using genetic algorithms: A tutorial”, *Reliability Engineering and System Safety,* Elsevier, available at www.sciencedirect.com.
15. Marie-Claude Gaudel, “Testing from Formal Specifications, a Generic Approach”, Springer (2001).
16. Ying Gao, “Study on multi-objective genetic algorithm”, IEEE (2000).
17. James A. Whittaker, “What Is Software Testing? And Why Is It So Hard?”, IEEE Software (February 2000).
18. S. R. Dalal et al, “Model-Based Testing in Practice”, Proceedings of ACM ICSE (1999).
19. Murata, T., “MOGA: multi-objective genetic algorithms”, IEEE (1995).

# F. P. Brooks, “No Silver Bullet Essence and Accidents of Software Engineering”, IEEE (1987)

1. Object Management Group (OMG)’s Formally Released versions of Object Constraint Language (OCL) available at <http://www.omg.org/spec/OCL/>
2. The Dresden OCL parser version 3.1. available at http://www.dresden-ocl.org/
3. Java Genetic Algorithm Package (JGAP) available at <http://jgap.sourceforge.net>
4. Metaheuristic Algorithms in Java (jMetal) an API for GA and MOGA based optimization, available at <http://jmetal.sourceforge.net/>
5. Eclipse IDE for Java Developers, available at <http://www.eclipse.org/downloads/>
6. JUnit a Unit Testing Framework for Java, available at <http://www.junit.org/>
7. Java Reflection API for assessing runt time properties of Java Objects java doc available online at <http://docs.oracle.com/javase/tutorial/reflect/index.html>