Regression Test Case Generation Technique Using Mobile Agents

A Thesis submitted

in partial fulfillment of the requirements for the award of degree of

DOCTOR of PHILOSOPHY

By

Pardeep Kumar Arora

(951003009)

Under the Guidance of

Dr. Rajesh Bhatia

Professor

Department of Computer Science and Engineering

Punjab Engineering College (Deemed to be University), Chandigarh

India

DEPARTMNET OF COMPUTER SCIENCE AND ENGINEERING

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY

(Deemed to be University), PATIALA – 147004

March 2018

turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Rajesh Bhatia
Assignment title:	kkk
Submission title:	Agent Based Regression test case..
File name:	finaldoc.docx
File size:	4.61M
Page count:	121
Word count:	25,544
Character count:	147,961
Submission date:	01-Jul-2018 10:04PM (UTC+0530)
Submission ID:	979698646

9%	3%	8%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

%

Bozkurt, Mustafa, Mark Harman, and Youssef Hassoun. "Testing and verification in service-oriented architecture: a survey : TESTING AND VERIFICATION IN SOA: A SURVEY", Software Testing Verification and Reliability, 2012.

Publication

Exclude quotes	Off	Exclude matches	< 4 words
Exclude bibliography	On		

Pardeep Arora

Certificate

This is to certify that the thesis entitled **Regression Test Case Generation Technique Using Mobile Agents,** submitted by Pardeep Kumar Arora (Roll No 951003009) to Thapar University, Patiala, embodies the work carried out under my supervision; and that is worthy of consideration for the award of the degree of Doctor of Philosophy.

Dr. Rajesh Bhatia
Professor,
Department of Computer Science and Engineering,
Punjab Engineering College (Deemed to be University), Chandigarh
India

Declaration

I certify that

a. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the Institute in writing the thesis.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

Pardeep Arora

(Pardeep Kumar Arora)

Acknowledgement

First and foremost, I would like to express my sincere gratitude to my supervisor Dr. Rajesh Bhatia for his patience, motivation, enthusiasm and immense knowledge. His guidance, continuous support helped me in all the time of research and writing of this thesis. He not only molded my mind to come up with a good research study but also supported me with his insightful comments at all stages of my study. I could not have imagined having better adviser and mentor than him for being continuously supportive and convincingly conveyed a spirit of adventure in regard to research study. Without his persistent help I could not have formed this concept and pursued this study.

Besides my adviser, I would like to thank our doctoral committee panelists Prof. Inderveer Channa, Dr. Shalini Batra, and Dr. Sanjay Sharma for their encouragement, insightful comments and support during progress evaluation and defense of my research work. I would like to acknowledge the gratefulness of Dr. Maninder Singh (Head, Dept. of Computer Science and Engineering), Prof. O.P Pandey (Dean Research) and Prof. P. Gopalan (Director, Thapar University) for providing the academic and technical support along with research facilities. The good advice, support and friendship of my fellow researchers, has been invaluable on both academic and personal level, for which I am extremely grateful.

Last, but by no means least, I would like to thank my son Yaksha Arora and my wife Shiwani Arora for their personal support and great patience at all times. My parents, brother and sister have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

List of Publications

SCI/SCIE Publications

Arora, P.K. & Bhatia R, (2017) "Mobile Agent Based Regression Test Case Generation using Model and Formal Specifications", IET Software, DOI: 10.1049/iet-sen.2016.0203

Impact Factor 0.733

Arora, P.K. & Bhatia R, (2017) "A Systematic Review of Agent-Based Test Case Generation for Regression Testing", Arabian Journal for Science and Engineering, 2018, Volume 43, Issue 2, pp 447–470, DOI https://doi.org/10.1007/s13369-017-2796-4

Impact Factor 0.865

Papers Published in Conference Proceedings

Arora, P.K., Bhatia, R., Agent based Regression Testing framework, Proceedings of International conference on Signal propagation and Computer Technology, IEEE, 411-414 (2014)

Agent-Based Regression Test Case Generation using class diagram, use cases and activity diagram accepted in Elsevier, Science Direct (Procedia Computer Science), International Conference International Conference on Smart Computing & Communications (ICSCC 2017):

(Procedia Computer Science Elsevier Journal)

(Scopus Indexed)

List of Figures

List of Tables

List of Abbreviations

Agent Based Software Engineering	ABSE
Agent Based Software Testing	ABST
Agent Oriented Software Testing	AOST
Agent-Based Test Management System	ATMS
Artificial Neural Networks	ANN
Autonomous Unit Level	AUL
Belief Desire Intention	BDI
Business Process Execution Language	BPEL
Combinatorial Interaction Testing	CIT
Cyclomatic Complexity	CC
Depth of Inheritance Tree	DIT
Deterministic Finite Automaton	DFA
Document Object Model	DOM
Extended Control Flow Graphs	ECFG
Extended Finite State Machine	EFSM
Extensible BPEL Flow Graph	XBFG
Foundation of Intelligent Physical Agents	FIPA
Global Platform	GP

Graphic User Interface	GUI
High Level Petri-Nets	HLPN
Home Appliance Control Systems	HACS
Human Robot Interaction	HRI
Inter-Procedural Level	IPL
Java Agent Development Environment	JADE
Java Application Template	JAT
Junit Axiom	JAX
Lack of Cohesion in Methods	LCOM
Line of Code	LoC
Model Versioning and Evolution	MoVE
Multi Agent Software Engineering	MASE
Multilayer Perception	MLP
Natural Language Processing	NLP
Object Constraint Language	OCL
Online Analytic Processing	OAP
Orthogonal Array Testing	OAT
Pair-Wise Testing	PWT
Particle Swarm Optimization	PSO
Regression Test Case Generation	RTCG

Research Questions	RQ
Semi-Supervised K- Means	SSKM
Software Under Test	SUT
Support Vector Machine	SVM
Systematic Literature Review	SLR
Telling TestStories	TTS
Test Development Environment	TDE
Total Change Impact	TCI
UML Testing Profile	UTP
Unified Modelling Language	UML
Unit Testing Framework for Java Programming	UTFJP
Verification Based Testing	VBT
Web Ontology Language	WOL
Web Service Definition Language	WSDL

Abstract

Regression testing involves testing modified software to expose faults if any introduced by the updates. It ensures that the functionality of previous code is not affected by the updates in the modified code. The focus of regression test case generation is to generate test cases for changed functionality and is a very time consuming and labor intensive task in regression testing. Most of the regression testing systems are using localized environment for test case generation, but that alone seems to be inadequate in today's complex and rapidly changing scenario. The potential alternative to address the issues of agility and complexity seems to be distributed systems. Agent-based technology seems to be an alternative for testing multifaceted, large, complex and heterogeneous distributed systems. Agent is a portion of code that perform on behalf of a user or other program. It possess additional properties like autonomy, intelligence, proactive mobility etc. and interacts in multi-agent system.

In our work, we conducted systematic literature review of existing test case generation approaches for regression testing and agent-based software testing systems. The emphasis is articulated on agent-based regression test case generation. Based on our inclusion and exclusion criteria, we identified 123 potential research papers on test case generation in regression testing and agent-based software testing and categorized them under model based testing, formal specifications based testing, structural testing, functional testing, mutation testing and random testing. The data extracted from our study are classified into seven broader areas of agent-based software testing. Based on our systematic literature survey, we recognized available techniques, approaches, platforms as well as methodologies for regression test case generation and developing agent-based software testing systems.

In the first approach, we used multi-agent systems for regression test case generation on distributed environment using standard unified modelling language (UML) models and formal specifications. Different agents are designed to perform model comparison, behavior comparison, specifications comparison, impact analysis, and regression test case generation. Agents designed in Java Agent Development Environment (JADE) framework perform these tasks by using XML files of UML class diagram, sequence diagram and formal specifications

based on Object-Z and OCL. In proposed method, all the three comparison agents: model comparison, behavior comparison, specification comparison work concurrently to save time as compared to other existing techniques.

The second approach is based on the comparison of UML class diagram, use cases, activity diagram and formal specifications OCL to identify changes at both syntax and semantics level. We compared UML class diagrams, use cases and activity diagrams of old and modified code to identify these changes. It is found that the use of UML class diagram, use cases and activity diagram results in better identification of changes and hence results in efficient test case generation. Additionally, agents developed in JADE are used to collect these changes from different stake holders in the distributed environment. The distribution of testing tasks among mobile agents reduces the average time required for generation of test cases in regression testing.

The third approach is based on agent based machine learning approach for identifying changes in the code for regression test case generation. These agents use object oriented metrics to identify changes within the software. In a first of its kind study, we integrated JADE with WEKA tool to design learning agents. We observe that Random Forest and Random Tree are classifiers giving best results for change impact analysis.

Table of Contents

Chapter - 1

Introduction

This chapter contains a short introduction to the thesis. Section 1.1 contains regression test process, 1.2 contains regression testing techniques. Motivation for work is provided in section 1.3. Section 1.4 contains agent-based technology. In section 1.5, we listed research gap, 1.6 contains problem formulation, 1.7 explains the research objectives, 1.8 provides contribution of the thesis and 1.9 contains the organization of thesis.

Software testing is a significant phase in software development life cycle. According to IEEE, software testing is "a process of analyzing software with the intent of finding any errors". High testability is the desirable goal and can be accomplished by identifying what needs to be tested and how well in advance. The significance of verifying the quality of software has been unanimously recognized. Significant research has been achieved to improve the testing process and the efficiency of software testing. One of the important, resource intensive and active field of software testing is regression testing. It involves testing of modified software to expose faults (if any) introduced by the updates. It includes execution of test cases repeatedly after the software has been updated with the aim to expose defects introduced by modifications.

Regression testing also known as "Program Revalidation" is derived from the word "Regress" which means degenerate i.e. return to a previous, generally worse state. It contains testing of software for newly added functionality and ensures that the changes do not affect the functionality of previous code [1].

Regression testing is a vital subject which is conducted by executing test set on a modified software to expose faults occurred due to modification. It is one of the most addressed problems in software testing due to time consuming and resource intensive field. Regression testing involves test case generation, prioritization, minimization and execution [2].

Test case generation is an activity that generates test cases for the Software Under Test (SUT) to detect errors and make testing more cost effective as well as efficient. During test case generation the program under testing is executed against the test cases to determine whether it

conforms to the requirements. It is a difficult and time consuming process and depends entirely on the tester. Many approaches have been put forward by the researchers for regression test case generation. So, regular increase in the level of development of these approaches has broadened its horizon.

1.1 Regression Test Process

Consider development cum test release process for an existing in-use program P. Let this program P be changed to P' due to addition of some features or may be due to changes required in the code to improve its performance. This program P' is tested for new functionalities; it is also tested for previous functionalities present in P. Regression testing refers to the test process which is performed to ensure that apart from the newly present code in program P' works correctly but also the code present in P works correctly. Regression testing is recommended after program P has been modified to program P' and any newly added functionality has been tested and found correct.

Consider a regression test process assumes that P' is available for Regression Testing. Regression testing involves following steps:

Regression test selection process is divided into the following set of test cases [3]:

- **Reusable Test Cases**
- **Retestable Test Cases**
- **Obsolete Test Cases**

Reusable Test Cases (Treuse): It refers to the set of test cases which can be reused to test the unmodified part of the program. There is no need to re-run these test cases as they return the same results on the new version as applied to the previous version.

Retestable Test Cases (Tretest): It represents a set of test cases that are required to be run to test the modified part of the program.

Obsolete Test Cases (Tobs): It represents a set of test cases that are no longer required because of the changes in the code.

The regression testing problem is to find a set of tests cases Treg on which program P' is to be tested to ensure that the code that implements functionality taken from program P works correctly.

Treg, set of regression test cases is a subset of T i.e. set of all possible test cases used for testing program P. Apart from it program P' is also tested for new test cases Tn in order to check new functionality added it it. Thus the test case required to test program p' leading to set of test cases T'=Treg U Tn required to test P'. While P is executed against the entire T, P' is executed only against the regression set Treg and the new set of test cases. Thus, regression test case generation problem is to find a set of minimum of non-obsolete test cases.

1.1.1 Test Revalidation, Selection, Minimization and Prioritization

Test Revalidation refers to the task of checking which test meant for P remains valid for P'. Revalidation is necessary to ensure that tests that are only applicable to P' are used during regression testing.

Test Selection/Generation refers to identification and generation of tests that traverse modified portions of P' from a set of reusable, retestable and obsolete test cases.

Test Minimization discards tests appearing redundant with respect to some criteria. The purpose of minimization is to reduce the execution of number of tests cases required for regression testing. e.g. if there are two tests say T1 and T2 are meant for testing function F in P, then one might decide to discard T2 in favor of T1 or vice-versa.

Test Prioritization refers to the task of prioritizing tests based on some criteria and then the test can be selected from a prioritized list.

1.2 Regression Testing Techniques

Grave *et al.* [4] presented classification of techniques of regression testing into Retest All, Minimization. Various other authors Rothermel *et al.* [5] suggested techniques like Prioritization which includes test case prioritization and other have suggested novel methods of Hybrid technique, which includes the combination of Minimization and Prioritization. Broadly regression testing can be divided into four categories as given below:

- **Retest All**
- **Regression Test Selection**
- **Test Prioritization**
- **Hybrid**

Retest All

This is one of the traditional and simplest methods of all of the available regression testing techniques. In this method, the user has to run all the possible test cases required to validate the modified program. It is the safest technique as the tester is not ready to take any risks and test the modified program P' in all existing test suites. The main drawback associated with this technique is that running all test a cases takes very long time so it leads to problem when time for release of software is very less.

Regression Test Selection

Regression test selection technique is preferred when time for release of build is less. Instead of running all the test cases it is preferred to run selective test cases out of the test case suite. In this technique, tests cases are selected randomly from the set of test suite. Based on the information about the modified program, the tester can decide a subset of test cases depending upon the level of confidence as well as the time required.

Test Prioritization

In test Prioritization technique, each test case is assigned some priority. Suitable metric is used to assign rank to every test in T. A test with the highest rank has the highest priority; the one with the next highest rank has the second highest priority and so on. This approach does not eliminate any test from T but leaves it on the tester to decide which test to select based on their relative priority. Code coverage can be used to assign the priority to test cases [5].

Hybrid

As the name implies, this technique is a combination of both regression test selection and test case prioritization. Many researchers are working on this technique and have suggested many algorithms and approaches for the same.

Test case generation is a very time consuming and labor intensive task in regression testing. Over the past years, the complexity of software has increased manifold due to its variety of applications. As the software is designed and developed in distributed environments, the identification of changes or updates in the software has become complex and difficult. Therefore regression test case generation has become cumbersome. Many approaches have been put forward by the researchers for regression test case generation. So, regular increase in the level of development of these approaches has broadened its horizon.

We have observed that most of the systems that perform software testing are using localized environment, but that alone seems to be inadequate in today's complex and rapidly changing scenario. The potential alternative to address the issues of agility and complexity seems to be distributed systems. So, there is a need to shift towards dynamically changing and efficient distributed systems. With the convergence of many technologies like object oriented, distributed and artificial intelligence, agents seem to be a natural tool for developing large, complex and heterogeneous distributed systems. Agent is a portion of code which perform on behalf of a user or other program. It possess additional properties like autonomy, intelligence, proactively, mobility etc. and interacts in multi-agent system. Agents offer high level software abstractions and can manage multifarious applications [6]. These properties have proved the significance and effectiveness of agents in automated testing. Over the past few years, agent-based approach has been integrated in software testing, a significant stage of software development life cycle.

An agent is an autonomous, intelligent system that interacts with the environment to achieve its goal. Agent-based technology seems to be an alternative for testing multifaceted, large, complex and heterogeneous distributed systems. Over the past few years, agent-based technology has emerged as a significant field in software testing. Thus, promising developments in regression test case generation using agent-based technology have raised the need to develop solutions for distributed systems to reduce time and make it cost effective. Although various researchers have advocated the use of agent-based technology for software testing, till this date no standard agent-based technique has emerged for regression test case generation.

1.3 Motivation for Work

Regression test case generation is labor intensive task in regression testing. It helps to improve the efficiency and productivity of the software. Multi agent-based software testing provides a better solution as compared to conventional manual software testing. It may aid in reducing the time and cost connected with testing to a greater extent.

After analyzing the latest developments in agent-based software testing; we have outlined the existing developments based upon a broader, more systematic investigation and reported the research gaps for analysis.

1.4 Agent-based Technology

Agent-based technologies are the trusted means to be adopted in industry and wide-reaching applications, normally working on heterogeneous and distributed systems. Agent-based structural design supports sophisticated logic, learning, planning and knowledge representation. Agents offer high level software abstractions and can manage multifarious applications because of their ability to support distributed systems. Agents are considered as next generation model for engineering complex, heterogeneous and distributed systems [2]

Software Agent

Software agent is a piece of code that works on behalf of a user or other program, functions independently and continuously in a specific environment [7]. Agents possess properties like autonomy, adaptivity, mobility, persistency, goal oriented communicative /collaborative, flexible, actively/proactively etc. Thus, software agent is a computer program that perform for a user or program in autonomous fashion and performs some level of proactivity and reactivity [7].

1.4.1 Types of Software Agents

There are many types of agents available in literature as follows:

Collaborative agents: These communicate and collaborate with each other to perform the task assigned to them.

Interface agents: Interface agents provide necessary interface to the users for accomplishing their tasks. They possess properties like autonomy and learning.

Mobile agents: Mobile agents possess features like mobility, autonomy and learning. They can migrate from one machine to another in heterogeneous distributed system. They can change its state from one environment to other without altering its data, collecting information on behalf of their owner and return to its host machine.

Information agents: Information agents are used for managing information. They can retrieve and apply different operations on information.

Reactive agents: Reactive agents receive input, process it and produce an output. They react on various situations occurring in the system due to which they are also known as intelligent agents.

Hybrid agents: Hybrid agents are a combination of the properties of two other agents: reactive and deliberative. Apart from reacting to external events they usually follow their own plans.

1.5 Research Gaps

The aim of the study was to make a review of existing literature and identify the potential techniques of regression testing, along with the relationship among them and their scope area in agent oriented technology. We made analysis of the findings and summary of existing regression testing techniques and their relevant research areas. The existing analysis and studies are diverse. We have identified some studies in the field of regression testing through agents. They have proposed it through case studies but still there lies a wide gap to meet real implementations. The agent-based technologies are still in early stages due to which very less actual agent-based systems have been built.

We observed that most of the systems that perform software testing are using localized environment, but they alone seems to be inadequate in today's complex and rapidly changing scenario. The potential alternative to address the issues of agility and complexity appear to be distributed systems.

With the convergence of many technologies, agents seem to be a natural tool for developing large, complex and heterogeneous distributed systems. To carry out automation of test case generation for regression testing, there is a need to identify the test case generation approaches in regression testing as well as agent-based software testing studies. This has directed for assessment and integration of approaches of test case generation for Regression Testing and Agent-based Software Testing (ABST).

1. Regression test case generation is a significant but ignored subjects in the industry, may be due to the cost of software maintenance is very high.

2. Poor choices in identifying change impact patterns and selecting testing technique during regression test case generation process may affect regression testing.

3. Regression testing is a tedious work and may require some human Intelligence. Agents are the best candidates for automated regression testing.

4. Many techniques for regression test generation have been proposed and evaluated but no broad solution based on agent has been proposed as existing agent-based test case generation techniques could not reply into the complexity of the problem and requirements in the software systems.

5. There is a need to develop Agent-based regression testing tools so as to get the benefit of distributed environment.

1.6 Problem Formulation

Based on the findings of the literature available and trends in regression testing, there is very little work done on regression testing through agents. Moreover, normal tools require much time to execute all the selected test cases. Therefore, a need for agent oriented and intelligent technology based tool is required which could replace the role of a tester, plans the test, selects the test cases from the test suite and then perform regression testing. It should also possess the properties of Intelligence like autonomy, adaptivity and decision making etc.

The importance of verifying the accuracy and reliability of software has been universally recognized. The impact analysis of modifications in a program has become challenging and

time consuming. Thus, there is a requirement for an effective technique to perform desired impact analysis and test case generation. Very little research work has been undertaken to provide valuable tools to support testing activities through agent systems, even the available work done is at very early stages.

1.7 Objectives of the Proposed Work

1. To explore various mobile agent-based regression test case generation techniques and to carry out their comparative analysis.

2. To study various mobile agent best platform/technologies for their possible use in regression testing.

3. To design and develop mobile agent-based regression test cases generation technique.

4. To verify and validate the proposed technique.

1.8 Contribution of Thesis

1. To achieve the first objective, we conducted systematic literature review of test case generation approaches/techniques in regression testing and agent-based software testing. We categorised studies on regression test case generation along with their techniques/algorithms used. We also identified different studies related to agent-based software testing.

2. To achieve the second objective, we conducted extensive systematic review of various different methodologies and platforms used in developing agent-based systems and agent-based software testing systems. On the basis of systematic literature review, we identified and recommended platforms and methodologies used for developing agent-based software testing systems.

We examined all identified studies for various platforms used by researchers for developing agent-based software testing systems. JADE seems to be the most popular platform among various researchers because of its open standard and compliance with IEEE Foundation of Intelligent Physical Agents (FIPA).

3. To achieve the third objective, we proposed two techniques for regression test case generation using agents to gather change identification in the code using UML models, formal specification and machine leaning techniques.

4. To achieve the fourth objective, we compared our both techniques with existing relevant studies using open and diverse subject systems. We empirically evaluated and compared our results against various subject systems with related studies and in terms of number of test cases generated using model and formal specifications. We have further enhanced the subject systems with Object-Z and OCL specifications to improve semantic analysis.

1.9 Organization of Thesis

The thesis is organized into six chapters. A brief outline of these chapters is given below:

First chapter introduces the concepts of regression testing with emphasis on major issues and complexities in regression test case generation. This chapter provides introduction to agents, their properties and characteristics. It also includes motivation behind the proposed research that worked as a driving force to pursue research in the area of agent-based regression test case generation.

In **second chapter**, a comprehensive systematic review of literature on agent-based test case generation for regression testing is presented. We have discussed various stages involved in performing the systematic review, review method, planning the review, research questions, search criteria sources of information and inclusion and exclusion criteria. We have presented a study of the existing literature on techniques, approaches and agent-based test case generation techniques for developing agent-based regression test case generation system. We have explored existing studies for identifying suitable methodologies, platforms and subject systems for developing agent-based regression test case generation systems.

Third chapter presents the experimental setup of agent-based regression test case generation technique. It uses UML model and formal specifications for regression test case generation. A comparative analysis of the proposed technique with other related studies have been presented in this chapter.

In chapter four, we have provided extension of agent-based regression test case generation using class diagram, activity diagram and use cases. Results show that our approach covers in-depth code coverage for change identification and takes lesser time as compare to other existing approaches.

In Chapter five, we have presented agent-based machine learning approach for identifying changes in the code. The details and working of all the four agents is presented in this chapter. These agents use object oriented metrics to identify changes within the software. We integrated JADE with WEKA tool to design machine learning based agents. The results of the proposed technique are put forward in this chapter.

Chapter six, this chapter concludes the thesis by highlighting the contributions made by the proposed research work. It also mentions the useful directions for future work.

Chapter - 2

Literature Survey

In this chapter, Section 2.1 contains background and describes various steps adopted in the review method, research questions, sources of information and analysis of data. Section 2.2 presents inclusion and exclusion criteria, Section 2.3 contains meta collection and data analysis. Section 2.4 contains studies related to test case generation approaches in regression testing approaches adopted and various methods used. Section 2.5 contains subject systems available for test case generation approaches in regression testing. Section 2.6 contains time based map of test case generation approaches in regression testing. Section 2.7 contains studies related to agent-based software testing. It includes discussions of methodologies, platforms, approaches for developing agent-based software testing systems. It also contains various areas of agent-based software testing and their time based evolution. Section 2.8 contains studies related to agent-based regression testing. In section 2.9, we have discussed the results and discussions of our findings. Section 2.10 contains summary of the work.

2.1 Background

Agent-Based Software Testing (ABST) use agents to automate various processes in software testing. These agents constantly monitor code for changes and perform test case generation, prioritization and execution as per the roles assigned to them. Multi-agents in the distributed environment communicate with each other and perform task to automate the various processes in regression testing.

Several languages, platforms and methodologies have been proposed for designing agent-based systems. The language used for developing agents should have high level abstraction and constructors to support agent properties resembling human like features: Beliefs Desire Intentions (BDI), plan, goals and communication [6]. Some of the agent programming languages having BDI properties are AgentSpeak, AF-APL, 3APL, 2APL, JADEX and Jason. Goal oriented agent language bridges the gap between agent logistics and agent programming model [6]. Some of the goal oriented agent languages are Golog, Claim and Flux. Some popular agent development platforms are JADE, Aglets, Madkit, Jafmas, Cougaar, AgentScapa,

Tacoma, Grasshopper, Voyager, Springs, Concordia Zeus and agentTool [6]. This study provides an organized summary and use of agents in software testing, which motivates recent efforts in adapting concepts, methodologies and platforms for Agent-Oriented Software Testing (AOST) to complex systems, which has not been done earlier.

2.1.1 Review method

We adopted the guidelines proposed by Kitchenham and Brereton [8] to evaluate and interpret all available research questions related to software engineering. Our work is motivated by systematic reviews of Engstorm *et al.* [9] on regression test selection techniques and Rattan *et al.* [10] to study the concept of software clone detection. Systematic reviews are very time consuming and require effort to identify current and past research related to a particular topic.

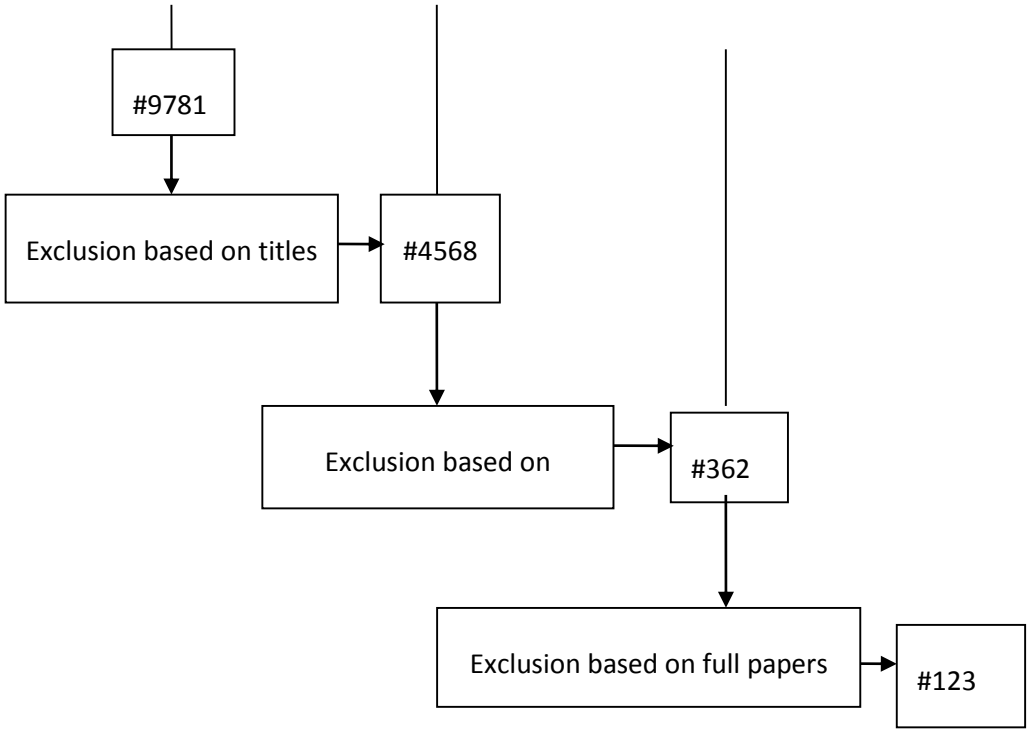


Fig. 2.1: Exclusion and inclusion criteria

After conducting various steps like planning of review, performing review, analysis of outcome, results and discussions, an effort was made to analyze and identify the approaches,

methodologies and platforms related to test case generation in regression testing and ABST. Different foras were searched to collect and compare existing evidences for this purpose.

2.1.2 Planning the Review

The outcome of every survey is based on the access to good quality research papers, original work and authentic references. Therefore, we explored various recognized research forums for journals, proceedings and databases. We divided our review process into three main parts. Research questions are framed in the first part. The second part contains the searching and studying of journals, conference proceedings and databases for required articles, whereas the third part contains the selection criteria of research papers for systematic review.

2.1.3 Research Questions

The aim of Systematic Literature Review (SLR) was to recognize and categorize the existing studies on test case generation approaches for regression testing, agent-based software testing and suitable platforms/methodologies for developing agent-based software testing systems. Identification of test case generation approaches in regression testing and agent-based regression testing are the final outcome of this research study.

We framed a set of research questions to carry out the SLR. Table 2.1 contains questions, sub questions and their motivations.

Table 2.1: Research questions

Research Questions (RQ)	Motivations
1. Research studies in test case generation in Regression Testing 1.1 What are the various studies in test case generation for regression testing and their comparative analysis?	Regression testing has been a crucial issue in the software industry as well as academia over the years. Regression testing being labor-intensive and expensive, several approaches and techniques have been put forward by researchers for effective regression test case generation. Thus, there is

1.2 What approaches have been used in these studies for regression testing? How many times have they been cited? 1.3 What subject systems are used to evaluate these studies? What is the size of subject systems and programming language used for their development? 1.4 How have these studies evolved over time? 2. Research studies in agent-based software testing 2.1 What are the key areas in ABST, number of studies in each area and their outcomes? 2.2 What approaches are available for ABST? What type of coverage criteria have they used? 2.3 Which agent-based methodologies and platforms have been used in software testing and how many times? 2.4 Which subject systems are used to evaluate the ABST systems? What is the size of these subject systems and their programming language? 2.5 How has agent-based software testing evolved over time?	a need to review the existing studies to identify the active research and find the current state of the art issues with a view on future development. Software changes/upgrades give way to regression testing. To assess the impact of software changes is a very difficult process. Automation in identifying changes and their coverage/impact will help in reducing the time and effort required for regression testing. In the age of distributed software development, agent-based software testing can be a promising alternative to trace changes from geographically distributed teams. ABST is a promising field, so there is a need to identify various ABST frameworks and technologies. There is a need to explore various methodologies as well as platforms suitable for designing agent-based software testing systems, so as to recommend the best suitable methodologies as well as platforms. It is significant to identify the areas related to ABST and number of studies for each sub area, so as to identify the areas for further research. We also aim to investigate all existing approaches of agent-based regression testing. This SLR will help in

3. Research studies in agent-based regression testing and their comparative analysis 3.1 What is the current status of agent-based regression testing? What are various studies, test coverage and their results? 3.2 What approaches have been used for agent-based regression test case generation? 3.3 What subject systems have been used to evaluate these approaches? What is the size of these subject systems, type of subject system-open source or commercial?	moving forward towards determination and standardization of studies related to ABST. It is important to the know various subject systems being used in existing ABST approaches, their size, number of their usage and programming language used. These subject systems can be useful for future research and empirical studies in agent-based software testing and test case generation in regression testing.

2.1.4 Search Criteria

We performed a very thorough search to find the studies related to SLR. Despite our best efforts, still few recognized research papers were left unidentified in earlier search approach because of various reasons like unusual title or keywords in search strings not present in abstract etc. The final search keyword was revised as test case generation in regression testing, regression test case generation, agents based testing/regression testing/web testing/distributed testing/software testing/mobile application testing/network testing.

2.1.5 Sources of Information

We applied the above mentioned pre-search criterion for finding relevant research papers from journals, conferences and workshop proceedings. For this reason, information gathered from the initial study in the specific area of test case generation in regression testing and ABST was applied and due consideration was given to the views of well-known software engineering researchers. It was realized that most of the references of articles consulted during the

preliminary search phase were leading towards reputed electronic libraries such as IEEE, Springer, ACM, Science Direct and Elsevier, Scopus etc. So, we explored these libraries for our research.

2.1.6 Additional Sources

Apart from these, we explored references of primary studies, technical reports and websites relevant to our work. We also considered a Ph.D. thesis related to our study. The publications considered in this study range from the year 2000 to 2017.

2.2 Inclusion and Exclusion Criteria

In the first stage papers were eliminated on the basis of their titles. In the second stage, detailed study of the abstracts for selected papers was made, so as to reject the papers not conforming to our research questions. We included only the studies focused on test case generation in regression testing and ABST. Apart from the selected papers, we considered some technical reports and research papers containing studies related to agent-based methodologies and platforms. References of significant studies were used to identify missing papers. In the third stage, exclusion was based on full text. The complete analysis of full text was performed on selected papers. We identified a total of 9781 papers in the first stage. In the second stage, these were narrowed to 4568 on the basis of titles and 362 papers based on their abstracts. In the third stage, we made detailed reading of papers shortlisted in second stage and finally selected 123 papers conforming to our research questions. Research papers with informal literature study and those not conforming to the defined research questions were excluded from the review.

2.3. Meta Collection and Analysis

A questionnaire was designed for extracting the data from the selected studies. We applied criterion to include the significant papers, also a quality assessment was applied to the remaining papers as per screening questions. Data related to our research questions was collected from the selected papers. The data collected from each selected paper was the source, full reference, authors and their affiliations, main topic area and its relevant fields,

classification of the study related to regression test case generation, agent-based software testing and outline of the study with focus on main research questions.

2.3.1 Data Analysis

Data analysis was made to explore the primary studies related to regression test case generation, agent-based software testing and their relevant areas published each year, research trends in regression test case generation, agent-based regression testing and their scope.

The objective of this research was to recognize suitable literature on regression test case generation and ABST. We identified 123 potential research papers on regression test case generation and ABST. Out of these 123 research papers, 66 studies are on regression test case generation; whereas the remaining 57 are on ABST. This includes 27 papers from reputed journals, 89 from international conference, 6 from workshops and a thesis.

2.4 Test Case Generation Approaches in Regression Testing

Out of the 66 studies of regression test case generation, 15 are from reputed journals and 51 are from various conferences and workshops. We found three secondary studies on test case generation viz. McMinn [11], Ali *et al.* [12] and Anand *et al.* [13]. McMinn [11] presented a survey of search-based software test data generation. Ali *et al.* [12] presented a systematic review of the various approaches in search-based test case generation. Anand *et al.* [13] presented a survey of the important approaches like model based testing, structural testing, random testing, combinatorial testing and search-based testing for automatic generation of test cases. Some of the studies related to our work are: Garg *et al.* [14] proposed ranked software engineering metrics in accordance with the value of permanent function on their criteria matrix. Kumar *et al.* [15] compared the performance assessment of genetic algorithm and classifiers based on decision tree in the presence of diverse datasets. Parashar *et al.* [16] presented a maximum fault detection technique based on prioritization of test cases based on their time-fault ratio. Singh *et al.* [17] discussed the issues and challenges in testing of software with non-functional requirement. Owens [18] presented an approach that focused on risk controlled and a framework for promising the development and life cycles of project management in software

quality assurance. Das [19] used mobile agents to mark the nodes with tokens to solve the problem of network failure.

There exist a number of approaches and algorithms proposed by researchers for test case generation in regression testing viz. model based testing, formal methods, structural testing, functional testing, mutation testing and random testing. But as per literature, we could not identify standard categorization for testing approaches related to the RTCG. This section summarizes the detail of these 66 available studies related to regression test case generation. Some studies have used more than one approach for test case generation in regression testing. We discussed these studies with all related techniques.

2.4.1 Model Based Testing

These approaches use model of the SUT for testing. Many techniques and algorithms have been proposed for regression test case generation through model-based testing viz. Unified Modeling Language (UML) class diagram, activity diagram, use case, sequence diagram, scenario-oriented test case generation etc.

Ricca and Tonella [21] proposed test case generation based on path expression by using node reduction algorithm. Definition-use and all-uses testing are performed by taking into account data dependence for assessment of the web sites quality. Bai *et al.* [22] proposed scenario based testing to capture system functionality. Hierarchical definition and organization of scenarios at various abstraction levels is provided to improve scenario based testing.

Chen *et al.* [23] presented model based regression testing and test case generation through Extended Finite State Machine (EFSM) with the specifications of SUT. The proposed approach constructs regression test suite from given EFSM to cover all occurrences of new dependence per modifications and reduces manual effort. Meyer *et al.* [24] proposed auto test unit framework for test case generation using Eiffel contracts as test oracles, generating objects and routine arguments. Classes are considered as contracts specifying expected behavior, run-time objects and their features. This approach reduces the size of bug-reproducing cases by generating scenario that triggers the failure. Gorthi *et al.* [25] used behavioral slicing to extract behavior of affected paths from UML use case activity diagram. Chen *et al.* [26] used Java

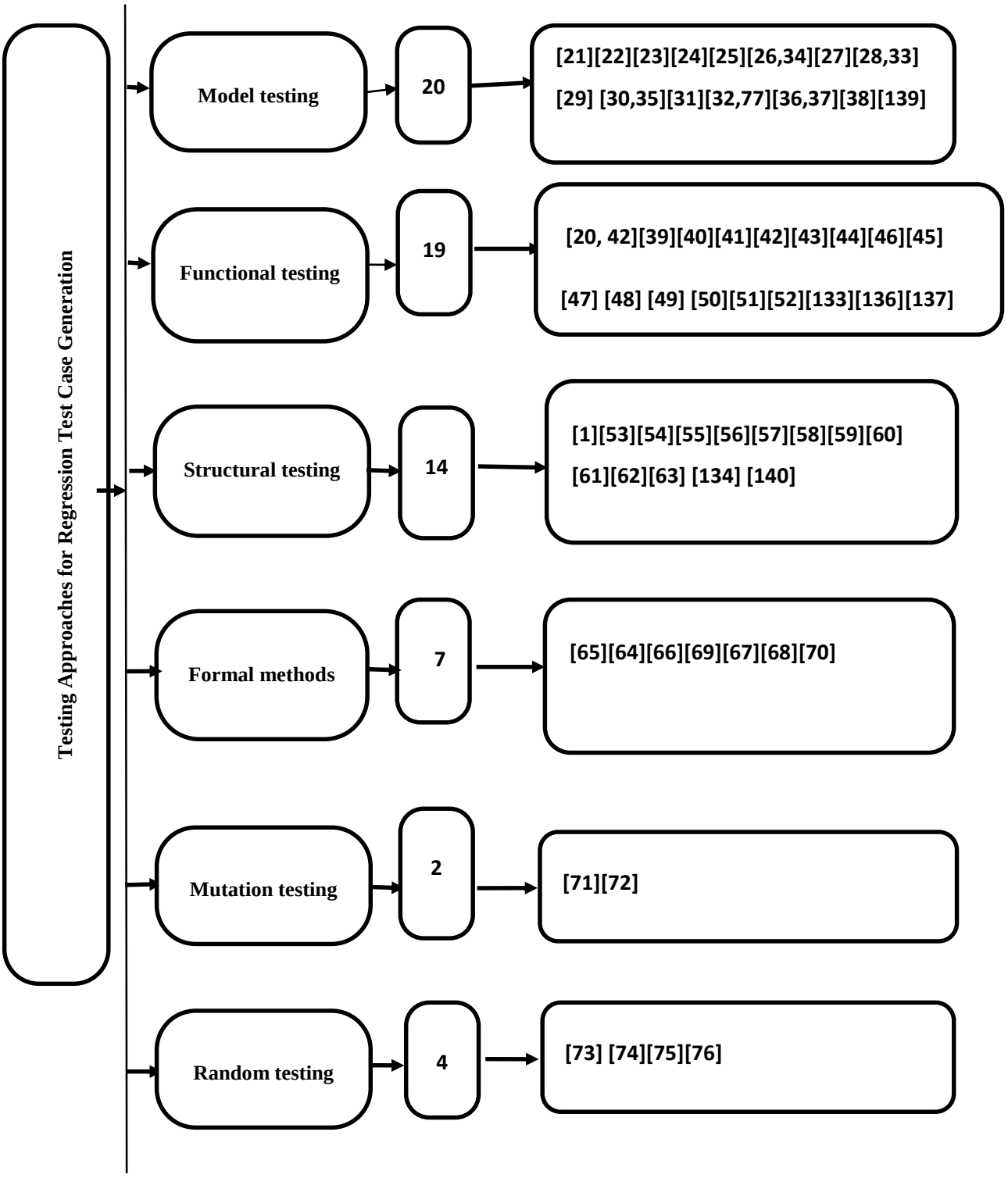


Fig. 2.2 Studies related to test case generation in regression testing

program for testing through activity diagram for generating rich test cases randomly to automate the testing process and reduce testing cost. Program execution traces are used to match the behavior of the activity diagram. Filho *et al.* [27] proposed Test Development Environment (TDE) based on UML to identify structural and semantic changes. Test cases are generated based on traceability links between models, user-defined critical paths, data coverage and path coverage. Fourneret *et al.* [28] proposed UML based test case generation using class/object diagrams and state charts augmented with OCL constraints.

In Sequence diagram based testing, Cho *et al.* [29] use a combination of flow of events, control flow of source code or flow of web pages in the sequence diagram to increase the number of test cases. Naslavsky *et al.* [30] converted UML class diagram and sequence diagram into control flow graphs based on models and a traceability model for better precision and inclusiveness. Kumar and Bhatia [31] used a model based on class, sequence and state diagram using Rational Rose tool to generate effective test cases by extracting information from petal file. Lamancha *et al.* [32] proposed model testing approach based on Query/View/Transformation from sequence diagram to generate the test behavior.

Vincent *et al.* [33] proposed a technique to reduce efforts in regression testing of object-oriented software. It is based on generation of test cases corresponding to impacted scenarios based on use cases through UML state chart and collaboration diagrams. Ye *et al.* [34] proposed used-revisions in activity diagrams to automate the regression test case generation. Activity diagrams are drawn using execution traces and behavior changes.

Zech *et al.* [35] proposed model-based regression testing approaches based on Model Versioning and Evolution (MoVE) platform. It contains support for XML-based model representations with the ability to parameterize with different change identification, impact analysis in Object Constraint Language (OCL). Fourneret *et al.* [36] generalize selective test generation method based on UML/OCL behavioral models for regression testing. It used behavior of the system from state charts and operations from class diagrams to generate test cases. Sapna and Balakrishnan [37] proposed black-box approach to check functionality by creating Steiner tree using nodes in an activity diagram. A minimum set of test cases are generated from specifications using UML activity diagram. Artzi *et al.* [38] automated the

process of test case generation by transforming classes into specific states. Al-Refai *et al.* [139] proposed fuzzy logic approach based on activity diagram and sequence diagram to classify retestable and reusable test cases. Sequence diagram is used to represent the usage scenarios of the test cases where as Activity diagram represents behavior of system methods.

2.4.2 Functional Testing

Functional testing is based on the functionality of the SUT. The examples of such techniques are equivalence partitioning, random testing, functional analysis and boundary value testing. Many researchers have used different techniques/algorithms/approaches for test case generation in regression testing through functional testing.

Ball *et al.* [20] applied testing of container classes based on state generation. Red black tree data structure is proposed through black box testing and white box testing by randomizing their selection to achieve high coverage. In Scenario based testing, Tsai *et al.* [39] have proposed the use of test case generation for end to end testing based on boundary testing, random testing and equivalent classes. It uses thin thread approach to bridge the gap between high level and low level source code implementation.

In assertion based testing, Xie [40] proposed object oriented unit test suite with regression oracle checking. Object states of class under test are collected and assertions are added into test case generation suite to improve its capability for guarding against regression faults. Muccini *et al.* [41] have proposed software architecture level reduced cost regression testing for retesting modified system and assess the regression testability of the evolved system. Qu *et al.* [42] presented code coverage based Combinatorial Interaction Testing (CIT) for test case generation and prioritization to improve the ability to detect faults.

Li *et al.* [43] have proposed Business Process Execution Language (BPEL) flow graph based on comparing different extended BPEL flow graph models. High coverage of test case selection and generation is achieved with changes occurring in processes, bindings and interfaces. Zhang and Yao [44] have proposed test case generation for operation sequence of semantic web services. Jolly *et al.* [45] proposed a tool for regression testing of supervisory control and data acquisition software. Tool generates Nunit test code by using black box specifications of the

units under test. Masood *et al.* [46] used Web Service Description Language (WSDL) for generating reusable test cases for testing web services. Original and modified WSDL specifications are parsed to generate separate operation trees to identify changes.

In Graphical User Interface (GUI) based approaches, Gladisch *et al.* [47] proposed the combination of verification and capture replay for test case generation in regression testing and unit testing using KeYGenU and GenUTest tools respectively. These tools are based on capturing interactions and creation of proof tree. Swearngin *et al.* [48] proposed human performance regression testing of the GUI. Test cases are generated by enumerating sequences of nodes in the graph. Klammer [133] proposed automatic test case generation tool for code-based API testing to generate GUI tests. Hammoudi [136] examined techniques based on record and replay for assisting the regression test case generation in web testing. Makki *et al.* [137] presented regression test case generation using capture and replay through BPMN2.0 Rodríguez *et al.* [49] proposed automation of functional tests of web applications. A tool is proposed to record interactions at communication protocol level and capture HTTP traffic. Stefan [50] generated test cases by using Document Object Model (DOM) to extract "href" links for testing web applications. Marin [51] proposed data-agnostic testing based on slicing operation for testing multi-dimensional databases. Mariani *et al.* [52] proposed test case generation and ranking based on regular sampling for components with numerical inputs and outputs in embedded software.

2.4.3 Structural Testing

In structural testing also known as white-box testing, internal structure of the SUT is used to generate test cases. It is based on node coverage, edges and paths of a graph representing flow of a sequence in a program. We are presenting various techniques and approaches used by researchers for structural based regression test case generation.

In assertion based testing, Taneja [53] presented DiffGen tool for automated regression unit-test generation and checking of Java programs. The proposed method implements the code by adding new branches to expose the behavioral differences between the two class versions to achieve maximum structural coverage.

In symbolic execution testing, Lee *et al.* [54] proposed automatic test case generation based on symbolic execution by parsing source code into parse tree representations. The proposed approach provides a solution for using symbolic execution with array indices for component based system testing. Zhang *et al.* [55] used tree based approach that covered the specific paths, branches or statements of the SUT. It generated 10% lesser test suite for affected path as compared to related work using dynamic symbolic execution. Taneja *et al.* [56] extended their work through dynamic symbolic execution based on path exploration for generation of test cases by exploring all feasible paths of program under test. Path pruning is proposed to optimize test case generation in goal based and path based techniques. Jamrozik *et al.* [57] proposed generation of test sets through augmented dynamic symbolic execution based on path coverage to cover all the simple paths in a program.

Bohme *et al.* [58] proposed partition based regression verification using gradual exploration of differential input partitions. Test cases are generated to expose different behaviors across two program versions. Tiwari *et al.* [59] proposed a modified time varying acceleration coefficients based particle swarm optimization generation algorithm for generating new test cases through code coverage analysis. Kim *et al.* [1] presented test case generation using hybrid test suite augmentation based on concolic augmentation and genetic augmentation. A dynamically interleaving approach is used to achieve higher branch coverage. Eda [60] presented technique to identify reusable constraint values to reduce human effort for regression test cases. A new set of regression test cases are generated using a tool based on analysis of code elements affected by code changes. Braione *et al.* [61] proposed ARC-B and DeltaTest techniques based on program slicing that generates effective test cases using symbolic execution for uncovered branches. The control flow graph is used to detect paths that lead to uncovered elements.

In search based testing, Blanco *et al.* [62] proposed test case generation based on scatter search approach to cover greater number of branches and difficult nodes of the control flow graph. Shamshiri *et al.* [63] presented an extension of unit test generation tool to generate tests that propagate regression faults using a search-based approach. Search-based technique is proposed to detect regression faults by maximum code coverage in both versions of the class. Hui [134] proposed integrated approach based on genetic algorithm and artificial immune algorithm for regression test case generation fault localization method. Sahoo and Ray [140] proposed

regression testing of web services by comparing operation trees of WSDL specifications. Forward dynamic slicing technique is used to capture the dynamic behavior of the web service and to detect the modified as well as the affected part of the web service.

2.4.4 Formal Method Based Testing

Formal method uses mathematical expression to represent vocabulary, syntax and semantics of the system. We observed various techniques and approaches including behavioral, model checking, OCL and Z Specifications etc. for test case generation in regression testing using formal specifications.

In behavioral based testing, Stotts and Lindsey [64] proposed unit testing framework for Java programming language (Junit) axioms based on algebraic specification of abstract data types for test case generation. Tsai *et al.* [65] extended their work by using OCL constraints to generate test cases for testing embedded system. Cavarra *et al.* [66] translated object oriented specifications based on UML models into formal, behavioral semantics for generating test cases.

In model checking based testing, Xu *et al.* [67] proposed model checking for specification based regression testing. Regression test cases are generated at the state transition level. Paths/branches of the models cover state transitions as well as variables having initial/final states changed. Fraser *et al.* [68] proposed a technique to identify obsolete test cases and test case generation approach based on model checking. Askarunisa *et al.* [69] compared orthogonal array testing, pair-wise testing and test case reduction techniques for semantic web services testing. The structure of web service is represented using UML and conditions for service rules are described using OCL. Xu [70] proposed integration and system test automation tool for automated test case generation based on coverage criteria. Test code is generated according to chosen coverage criteria and organized as transition tree. It enables the user to debug Model implementation description specification and manage the tests before executable test code is generated.

2.4.5 Mutation Testing

In mutation testing, the program under test is mutated to various versions also known as mutants [71]. Each mutant contains one automatically seeded fault based on set of mutant operators to mutate the statement of the original program into a faulty statement [71]. Zhang *et al.* [71] proposed PexMutator mutation testing tool based on dynamic symbolic execution. PexMutator uses dynamic symbolic execution to generate test inputs for the meta-program. Fraser and Arcuri [72] proposed EVOSUITE to automatically generate test cases for classes written in Java. It performs mutation based testing using oracles by inserting a set of assertions representing behavior.

2.4.6 Random Testing

Random testing is widely used for automatic generation of test data without human favor [73]. For regression test case generation through random testing, Parizi *et al.* [73] presented random testing based automated test case generation for testing AspectJ programs. Li and Yang [74] proposed optimization of test suite by ant colony arithmetic to generate pair-wise covering test data. Robinson *et al.* [75] used enhanced feedback-directed random testing approach for generating regression unit tests by using Randoop test generation tool. Alipur *et al.* [76] proposed directed swarm testing for generating random test cases focused on selected code targets to improve their coverage. We explored various techniques/approaches used for regression test case generation and presented them in Table 2.2 including the count of referenced articles and their citation numbers. We noted that research publications in the field

Table 2.2: Technique/algorithm used for test case generation in regression testing

Testing Type	Technique/Algorithm	Count	Reference
Model Based	Path Expression based on node reduction algorithm	1	[21]
	Scenario based	1	[22]
	Message communication	1	[29]
	Extended finite state machine model	1	[23]

	Eiffel contracts	1	[24]
	Behavioral slicing	1	[25]
	Behavioral changes	2	[38] [139]
	Execution trace	2	[26,34]
	Path coverage	1	[27]
	Matching program traces with activity diagram behavior	1	[31]
	Class, associations and state change identification	2	[30,35]
	Use case, state charts	2	[28,33]
	Based on activity diagram	3	[36,37][139]
	UML Class diagram	2	[32,77]
Functional Testing	Combinatorial algorithm based on state generation	2	[20,42]
	Boundary value testing	1	[39]
	Assertions based	2	[40][42]
	Software architecture conformance testing	1	[41]
	WSDL specification changes	3	[43,44,46]
	equivalence classes based	1	[45]
	GUI based testing	6	[47-49] [133][136][137]
	Slicing based	1	[51]
	Hyperlink	1	[50]
	Behavior of function	1	[52]
Structural Testing	Symbolic execution	5	[54-58]
	Assertion based	1	[53]
	Search based	2	[62,63]

	Particle Swarm Optimization (PAO) Algorithm	1	[59]
	Hybrid augmentation approach based on concolic and genetic augmentation	1	[1]
	Program slicing	1	[60]
	RC-B and Delta test approach based on concolic execution and program slicing respectively	1	[61]
	Genetic Algorithm	1	[134]
	WSDL comparison based on trees	1	[140]
Formal Specification	Algebraic semantics of ADT	1	[64]
	Associated formal and behavioral semantics	1	[66]
	OCL	2	[65,69]
	Model checking	2	[67,68]
	Petri-nets based on coverage criteria and transition tree	1	[70]
Mutation testing	Symbolic execution	1	[71]
	Assertion generation	1	[72]
Random testing	Ant Colony Optimization	1	[74]
	Assertion generation	1	[75]
	Aspect-related testing	1	[73]
	Direct Swarm Testing	1	[76]

of test case generation in regression testing are more inclined towards model based testing as compared to structural and functional testing. We observed that test case generation in structural testing is preferred using symbolic execution and OCL is preferred in formal specification testing. We also noted that some researchers have used multiple approaches for test case generation in regression testing. Even though many researchers have used diverse

techniques and approaches for test case generation, these techniques and approaches can benefit from each other and further be used to improve test case generation in regression testing.

2.5 Subject Systems used in Regression Test Case Generation Studies

In the absence of standard parameters, comparison of regression test case generation techniques and approaches is a challenging task. We noted different subject systems used by researchers for evaluating their techniques and presented them in Table 2.3. We categorized these subject systems into small (<10 K LOC), medium (10K to 50K LOC) and large (>50K LOC) groups.

Table 2.3: Subject systems used in test case generation in regression testing

Subject System	Language used	Size	Count	Study
Banking applications	Java/ C++	Medium	8	[20,22-24,39,44,47,63]
Programs from software repository	C	Medium	4	[1,35,58,59]
Shopping cart component	Java	Medium	4	[25,37,40,64]
Control systems	C, Labview	Small	4	[52,61,67,68]
Web based projects	Java/PHP	Medium	3	[21,29,46]
Online stock exchange system/open source	Java	Medium	3	[33,34,42]
Java based projects from software repositories	Java	Small	3	[53,56,72]
Ecinema, GlobalPlatform (GP) smart card	Java/PHP	Medium	2	[28,36]
Simple programs based on Factorial, Power, MaxValue, GCD, Sorting	.NET	Small	2	[57,71]
Store management & shopping cart	PHP	Large	2	[60,70]
Elevator and cargo system	C2 Framework	Medium	1	[78]

Flex	Lexical Analyser	Medium	1	[42]
Online hotel reservation system	Java	Small	1	[31]
Rocket SCADA control system	C#	Medium	1	[45]
Online shopping systems MAGENTA, ZEN CART	PHP	Large	1	[70]
Library system	Java	Medium	1	[32]
Human Interface Machine	Java	Small	1	[133]
Siemens Suite and Space	C	Medium	1	[134]
Patient Monitoring System	Junit	Medium	1	[137]
Airline seat reservation System	Not Specified	Medium	1	[139]
Calculator System	Not Specified	Medium	1	[140]

We observed that diverse subject systems have been used for evaluating these techniques and approaches but still banking applications using open source Java/C++ remain popular among researchers.

2.6 Time Based Map of Test Case Generation Studies in Regression Testing

The annual trends of test case generation studies in regression testing over the last seventeen years are presented in Fig. 2.3. The graph summarizes report on 66 studies with a maximum of 7 studies in year 2013 having 4 major studies in the area of structural testing. 6 studies reported in the each 2010, 2011 and 2014 with maximum studies 3 in functional testing. 2 studies each in model based testing and functional testing were reported in 2010. 5 studies in each reported 2012 and 2015 with maximum studies on model based testing. Thus, from the graph, we observe that test case generation approaches in regression testing are continuously evolving over time.

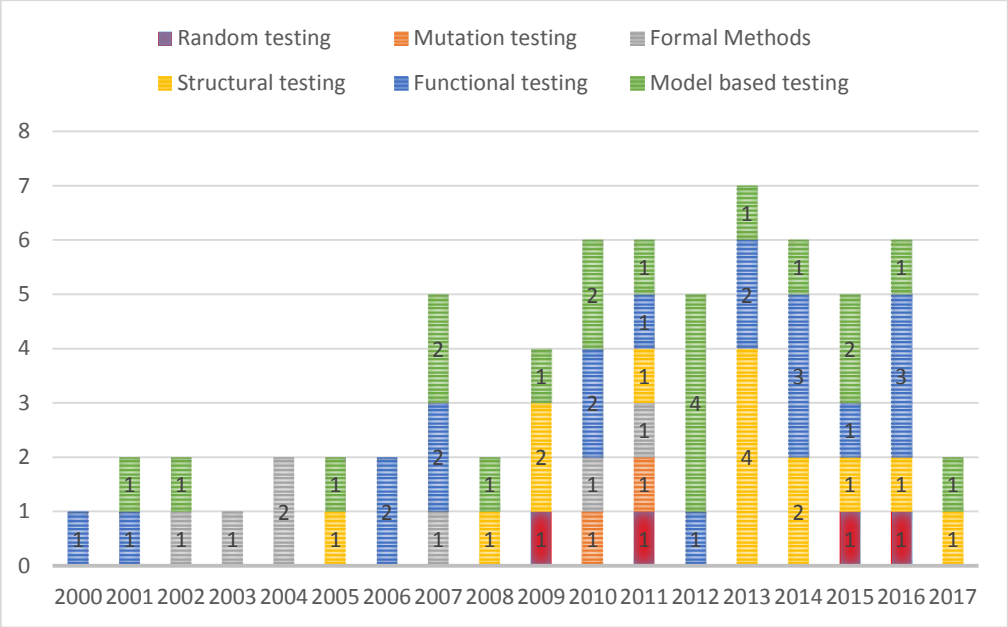


Fig. 2.3 Year wise annual trend of test case generation studies in regression testing

2.7. Studies Related to Agent-based Software Testing

We found 57 research papers on agent-based software testing and categorized them on the basis of various software testing approaches and research areas. These 57 studies include 13 from refereed journals and 44 from premier conference proceedings and workshops. The number of research studies in each category is given in Table 2.4.

Table 2.4: Agent-based software testing studies

Sr. No.	Category	Count	Studies
1.	Web testing	16	[79-94]
2.	Object oriented testing	12	[95-106]
3.	Distributed system testing	7	[78,107-112]
4.	Regression testing	4	[113-116]
5.	Network and mobile application testing	5	[117-121]

6.	Test selection and prioritization	4	[122-125]
7.	Miscellaneous	9	[126-132] [135][138]

We have categorized the available studies related to agent-based software testing into seven broad areas. The above table clearly shows maximum 16 research studies i.e. 28% pertain to web application testing, 12 studies about 21% are in object oriented testing and 7 about 13% in distributed testing, 5 each in regression testing, network and mobile application testing. 9 studies are scattered over various domains, so we have categorized them as miscellaneous.

It shows that agent-based software testing is most popular in web testing. However, the potential benefits of mobile agent-based software testing in distributed systems need to be explored further. The studies in regression testing reflect that the effort required can be reduced to manifolds by delegating the duties to agents. So, change impact analysis and regression testing with mobile agents may be investigated further to harness the real benefits. Few studies have their domain in multiple areas. For simplicity, we have put them in one primary category.

It has been tried to identify the testing approaches used in above agent-based studies. These studies have used numerous testing approaches in particular category to achieve variety of testing objectives. These include model based testing, structural testing, functional testing, performance testing and formal testing etc. as shown in Table 2.5.

Table 2.5: Application of agents in various software testing approaches

Id	**Testing Approach**	**Count**	**Study**
T1	Structural based	12	[78-81,84,86,89,104,105,114,123,130]
T2	Model based	9	[82,94,98,99,102-104,115,131]
T3	Functional based	9	[107,111,116,117-120,127,129]
T4	Performance based	7	[88,90-93,95,127]
T5	Formal specifications based	7	[34,85,87,94,102,109,110]

T6	Mutation analysis	4	[96,97,100,101]
T7	Data flow based	4	[82,83,88,108]
T8	Fuzzy based	3	[123-125]
T9	GUI based capture and replay	2	[106,113]
T10	Miscellaneous	5	[126,128,132][135][138]

In agent-based web testing studies many researchers have used different techniques and approaches. Qingning and Hong [80] proposed multi-agents for test case generation on the basis of node coverage, link coverage and linear independent path coverage for structural testing. Different testing agents are proposed to retrieve web pages, analyze source code and generate test cases according to testing criteria. Hong [81] proposed using Lehman's theory of software evolution for web testing. Service agents and management agents are proposed for structural testing, verification and validation of web functionalities. Xu [98] proposed cloning agents to generate scenarios from UML activity diagram. Fork Join approach in UML activity diagram is used to extract scenarios and automate the test case generation. Xiaoying *et al.* [84] proposed multi-agent-based framework for collaborative testing on web services using XML based service test specification. Xiaoying *et al.* [86] extended their work by proposing various agents for dynamic generation, scheduling, execution and monitoring of test tasks. Yu *et al.* [83] proposed different agents based on BDI approach to perform data flow testing through function level, function cluster level and object level testing.

In model based web testing, Kung [82] proposed BDI model based on use case goal diagram, domain model and sequence diagram for testing structural, functional and state information of web component under test. Wang *et al.* [89] proposed mobile agent-based model for workload testing, stress testing, security testing and reliability testing of web applications. Paydar and Khani [94] compared its model based web testing framework on the basis of various parameters including test types, test strategy, information sources, manual intervention, test applicability time, target type and framework architecture. In formal based web testing, Huaikou *et al.* [85] presented formal specification agent-based framework on role methodology for testing web applications. Dong [87] proposed multi-agents to get and analyze BPEL/WSDL specifications

for testing web service composition using HPN. In performance testing, Zhang and Xu [88] proposed mobile agents to carry test cases from host machine to remote side and after successful testing they submit the result back to the local site. Zhang [93] extended their work by the integration of existing testing tool HPLoadRunner with mobile agents IBM aglet for testing web services through remote method invocation. Ma *et al.* [90] proposed BDI based agent model to exchange knowledge to accomplish testing tasks for adaptive performance testing of web services. Zhao *et al.* [91] proposed five agents assigned with different roles on BDI approach to accomplish the performance testing. Hao *et al.* [92] proposed different agents viz. test flow generator, scenario creator, test manager, load generation agents, test monitor and test analyzer to support performance testing of WSDL based web services.

In agent-based object oriented testing, Grundy *et al.* [95] proposed test case generation agents to perform validation and performance testing of deployed software components using aspect-encoded requirements and configuration scenarios. In mutation based testing, Dhavachelvan and Uma [96] proposed agent-based software testing framework consisting of two agents and fuzzy repository table to assess the complexity of the products to be tested. Dhavachelvan and Uma [97, 100, 101], in their extended work, proposed multi-agent testing environment to test SUT under different distinct environments using different testing strategies. Li and Lam [99] proposed an automated approach using ant-like agents to generate test threads from the UML artifacts. Mala and Mohan [102] proposed intelligent search agent to find optimal paths using UML state chart. Siwen and Jun [103] proposed software test case generation for testing software component. Dam and Ghose [105] proposed static and dynamic techniques for change impact analysis using source code and execution trace to study the behavior of BDI based agent models. Baiquan [106] proposed four agents viz. controller agent, RecorderAgent, ReplayerAgent and MonitorAgent for recording user actions, playback recorded script and record performance data respectively.

In test case prioritization, Malz *et al.* [123, 125] proposed Adaptive Agent-based Test Management System (ATMS) for software testing. ATMS gets the input from test object and performs the prioritization of test cases based on test importance i.e. local priority and global priority. Rongfa [124] extended their work to fuzzy based prioritization of test cases. Test unit agents are proposed to test importance for the test unit. Fuzzy based test case agent determines

local priorities and test importance by converting fulfillment degree of every rule into crisp value.

In distributed automation testing, Cherif *et al.* [107] proposed three layer multi-agent-based software architecture evaluation framework. Different agents are proposed to analyze the information and work in cooperation to achieve testing tasks. Hany *et al.* [108] proposed multi-agent framework for unit testing, integration testing, regression testing and stress testing of distributed systems using link failure, user input validation and error in retrieving user data. Gardikiotis *et al.* [78] proposed multi-agent frameworks for database application testing to support and control the overall testing process for oracle PL/SQL statements. Jing and Yuquing [109, 110] proposed multi-agents for allocating and deploying test tasks, capture test log and execute test scripts. Chu [111] proposed automated testing of client/server applications through blackboard architecture using trace file to study the interaction behavior among clients.

In network and mobile application testing, Liang *et al.* [117] proposed different agents to test components, configuration and behavior for testing network load sharing facility software, but no concrete testing technique and approach has been suggested. Chengqing *et al.* [118] proposed multi-agent-based network performance test system. Friess *et al.* [120] proposed agent-based SpamTestSim tool for testing anti-spam software. Testing agents are proposed to send, receive mail, create and maintain contact lists. Manzoor *et al.* [121] proposed multi-agent framework for testing and verification of network software.

In mobile application testing, Ilarri *et al.* [119] proposed mobile agents for testing mobile computing applications using real wireless data service.

We tried to categorize studies having diverse fields or area having little relevance under miscellaneous category. These include Tang [126] proposed multi-agents to automate the process of software test life cycle. Test cases are generated using test requirements based on functional requirements or use cases stored in test repository. Dragunov and Sahin [127] proposed agent-based framework for testing swarm robot for its evolving behavior using a discrete event simulation and XML based hardware in the loop simulation. Sivakumar *et al.* [128] proposed multi-agent testing framework based on case base reasoning to debug and test e-learning software. Merdan *et al.* [129] proposed multi-agent framework for testing multi-

agent control systems. Chen and Wang [130] proposed multi-agent fault diagnosis and spectrum based fault localization system for software intensive equipment using four different agents to generate syntax tree and perform the analysis through branch coverage. Öztürk *et al.* [131] presented multi-agent web testing framework using .NET for software test tool selection process. Three agents are proposed for detecting software error types, selecting test software according to error and execute web test software. Dejanira and Pipe [132] used BDI based agents for test generation in human-robot interaction (HRI). Use of RL algorithm, Q-learning, with a reward function on agents resulted in high percentages of automatic code coverage. Schatten [138] presented agent-based automated testing using a game oriented approach. The testing framework is divided into three layers. Agent represent the players modeled using graphical modeling language and uses knowledge base repository to the complete the assigned tasks.

The inferences drawn from Table 2.4 and Table 2.5 are represented in Fig. 2.4. We observed that agent-based web testing and object oriented testing is preferred by researchers. Structural testing seems to be popular amongst the researchers with 12.

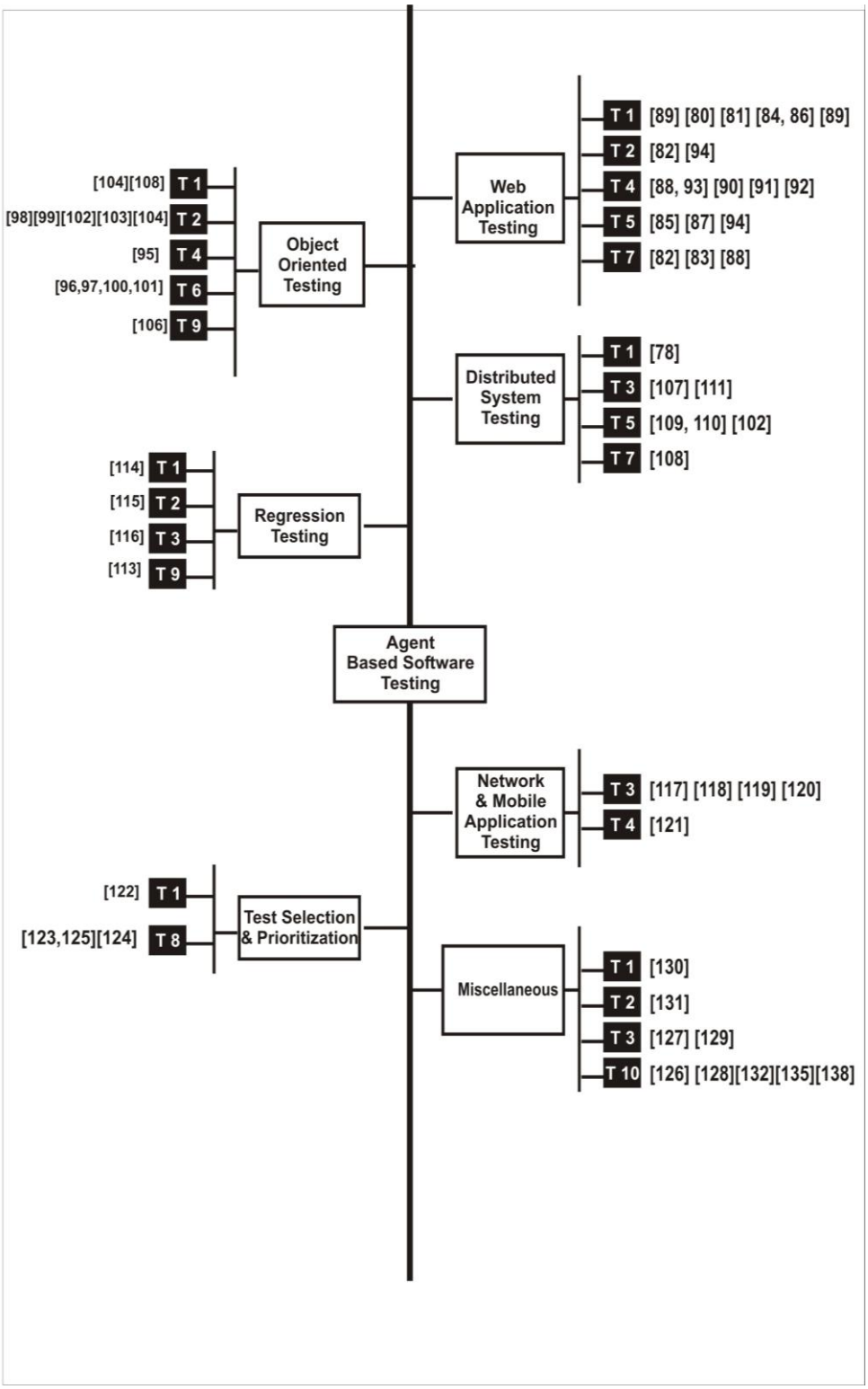


Fig. 2.4: Different areas of agent-based testing

2.7.1 Test Coverage in Agent-based Testing Studies

We explored various types of coverage considered by researchers in their studies and presented them in Table 2.6.

Table 2.6: Test coverage by different studies

Test Coverage	**Study**
Hyperlinks, node coverage and path coverage	[80-82,89,102,113]
Interclass, inter method, inter object	[83]
Branch coverage and statement coverage	[96,97,100,101,103,118,130]
Control flow and data flow	[82,108,114]
Aspect specification	[95]
Mutation testing, capability testing based on function sets	[82]
Code coverage	[78,79,98]
Role based test task e.g. validation of text boxes	[85]
WSDL specification	[84,86,87,92,94,116]
Quality, cost, time attributes associated with method	[88,93]
Constraint coverage, node coverage, variable information coverage	[103]
Fuzzy controller for checking test importance based on test coverage functions, number of defects	[123,125,128]
Knowledge base	[126]
Users actions on GUI	[93,106,117]
Web page components	[121]
Model coverage	[99,106]
Assertion coverage	[132]

We observed that a maximum of seven studies have used branch coverage and statement coverage, followed by six studies each in Hyperlinks, node coverage and path coverage and WSDL specifications. There are three studies each in control flow-data flow, code coverage, GUI and Fuzzy controller based on coverage functions, used by researchers for designing agent-based testing systems. Decision based actions are initiated in all these coverage techniques. These actions can be initiated using agents delegated with dynamic, action oriented tasks to save human effort.

2.7.2 Agent-based methodologies and platforms in testing

There exist a large number of agent-based methodologies proposed by various researchers but as of now agent-based software engineering is not standardized. Based on the available literature, we identified popular agent-based methodologies used in software testing as BDI, Gaia and MaSE etc. as shown in Table 2.7.

Table 2.7: Methodologies used in agent-based software testing systems

Methodology	**Count**	**Study**
BDI	7	[82][83][90][91][105][132] [138]
GAIA	2	[85][135]
TTCN-3 , SODA	1	[92]
MaSE	1	[128]
Rule Based MAGSY	1	[112]

Out of reported 57 agent-based studies, 12 i.e. 21 % have used existing agent-based methodologies. Out of these 12 studies, 7 studies have reported the use of BDI making it more favorite among researchers for developing agent-based software testing system. In the remaining studies, the use of agent methodology is not explicitly stated or they have used their tailor made methodology. Kung [82] used BDI approach for modeling agent-based web testing. Yu *et al.* [83] proposed the use of agents based on BDI approach for data flow testing of web applications. Ma *et al.* [90] used agents based BDI approach for performance testing of web

services. Zhao *et al.* [91] used BDI approach for designing multi-agent-based automated testing framework. Agents are assigned roles from knowledge base to achieve different objectives. Huaikou *et al.* [85] used Gaia methodology for assigning different roles to the agents for testing web applications. Padmanban and Thirumaran [135] presented agent-based methodologies and comparative analysis of object oriented software testing with agent oriented software testing. Unit testing, system testing and integration testing based on agents is proposed for testing agent-based software.

There are large numbers of platforms available for developing agent-based systems. These include Java Agent Development Environment (JADE), Aglet, Grasshopper, Voyager, Springs, Concordia and Ajanta. We examined all identified studies for various platforms used by researchers for developing agent-based software testing systems. Out of the 57 studies, only 17 i.e. 30 % studies have specifically mentioned the use of agent platforms as given in Table 2.8.

Table 2.8: Agent-based platforms used in software testing studies

Platform	Count	Study
JADE	13	[89-91,94,103,106,108,109,114,115,121,122,128]
Aglet	2	[88,93]
Springs	1	[119]
SPADE	1	[138]

Out of these 17 studies, 13 have used JADE platform to develop agent-based software testing systems, two studies Zhang and Xu [88], Zhang [93] used aglet platform for integration with HPLoadrunner tool and Springs by Ilarri *et al.* [119]. Other researchers have used various platforms viz, C++ by Chengqing *et al.* [118] and Visual C++ by Sharma and Capretz [79], C# by Öztürk *et al.* [131], Prolog by Charaf *et al.*[112]. Six have used different versions of Java by Kung [82], Grundy *et al.* [95], Yu *et al.* [83], Mala and Mohan *et al.* [102], Hao *et al.* [92] and Chu [111]. JADE seems to be the most popular platform among various researchers because of its open standard and compliance with IEEE Foundation of Intelligent Physical Agents (FIPA).

2.7.3 Subject Systems in Agent-based Testing Studies

Table 2.9 lists all the subject systems used in existing agent-based testing systems including size of the system, language used for developing the subject system and number of times the subject system was used in different studies. We categorized these subject systems into small (<10 K LOC), medium (10K to 50K LOC) and large (>50K LOC) group. We noted that diverse subject systems have been used in designing and testing agent-based software testing systems.

Table 2.9: Subject systems used in agent-based software testing studies

Subject System	Language used	Size	Count	Study
ATM System	Java	Medium	4	[87,102,104,111]
Game, Text editor, Experiment pack, Medical image analysis tool	C++	Small	4	[96,97,100,101]
Online book store	JADE	Medium	3	[84,86,115]
Software from industrial automation system	Not Specified	Not Specified	3	[123-125]
National digital forecast Database	Web Applications	Large	2	[88,93]
Home appliance control systems (HACS)	Java	Medium	1	[83]
Web based Projects	Web based	Medium	1	[108]
Online Stock Exchange System/open source	Java	Medium	1	[86]
Foundation software platform	Operating System, DBMS	Large	1	[109]
Spam filters: BogoFilter,DSPAM, CRM114 Spam Assassin simulators	Not specified	Not Specified	1	[120]
Shopping cart component	Java	Medium	1	[87]
E- learning architecture	JADE	Medium	1	[128]
Control Systems	C	Small	1	[104]

Robot: ABB IRB 140	C++	Medium	1	[129]
Selenium, Fitnesse, Vega Subgraph , SqlQueryStres, Visual Studio Test Analyzer	Not Specified	Medium	1	[131]
ROS-Gazebo simulator	Python	Small	1	[132]
Mana World Game	MMORPG	Large	1	[138]

From the above table, it is obvious that ATM developed in Java is the most popular subject system among researchers. This may be because it is an open source. Another popular subject system among researchers is online book store based on Java from open source repositories. Dhavachelvan and Uma [96, 97,100,101] used game, text editor, experiment pack and medical image analysis tool based on C++ for evaluating agent-based testing systems. Malz and Göhner [123], Malz *et al.* [125] and Rongfa [124] used software from industrial automation system for testing fuzzy base test case prioritization. Java based subject system seems to be the most popular among the researchers.

2.7.4 Time Based Map of Agent-based Software Testing Studies

We searched all the agent-based testing studies from 2000 to 2017. In Fig. 2.5, annual trends of agent-based software testing studies over the last sixteen years is presented. The graph reported a maximum of 9 studies in year 2012, 7 studies each in 2005 and 2009, and 6 studies each in the years 2007 and 2010. Where as in year 2001, 2008 and 2013, comparatively lesser numbers of studies were reported. No study has been identified in the years 2000 and 2002. We have noted that agent-based testing has started growing in diverse fields from 2012. Thus, apart from web testing and object oriented testing, new avenues have opened up in the area of agent-based software testing.

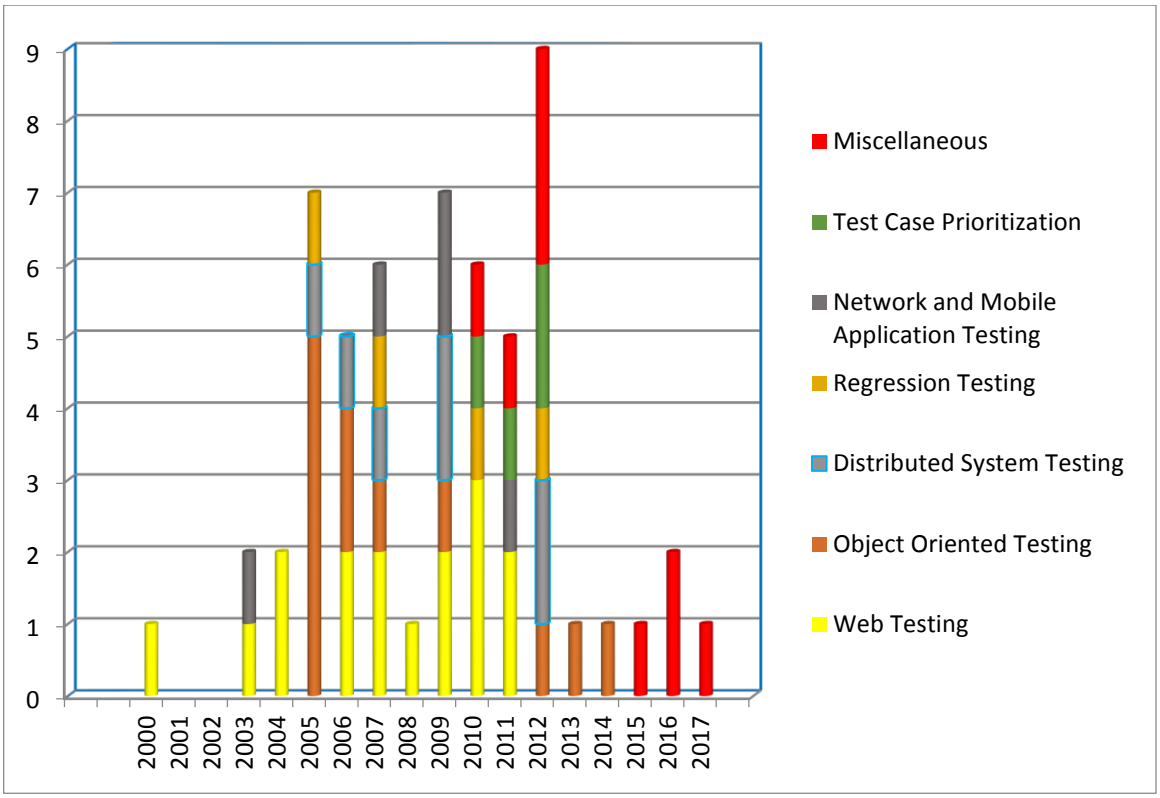


Fig. 2.5: Time based count of key sub areas

2.8 Agent-based Regression Testing

We compared existing studies in agent-based regression testing and presented them in Table 2.10. These studies are compared on the basis of various parameters, activity performed, test coverage, input and output.

Table 2.10: Comparative analysis of agent-based regression testing approaches

Activity Performed	Test Coverage	Input	Output	Study
Test Case generation	Code coverage	Source Code	Report file	[79]
Test case generation, execution and repair	Capturing users actions	User actions	Report file	[113]

Test Case Generation, Test Case Execution	Data flow coverage, control flow coverage	User Input	Report file	[108]
Test case monitor, generation, prioritization, optimization	Code coverage	Two versions of source code	Set of test cases	[114]
Test case monitor, generation and execution	Code coverage and behavior based	Files containing agents under test	Report file	[115]
Test planning, test control, Test optimization	Fuzzy controller based on coverage functions	Test Object consisting of test units, test cases, test resources	Set of test cases	[123-125]
Test case generation, management and scheduling	Capturing user actions	User actions	Report file	[126]
Test case prioritization	Loop/branch/statement/ condition based	Set of test cases	Test cases	[122]
Test case monitor, generation and execution	Code based	XML file	Test cases	[116]

Sharma and Capretz [79] proposed agent-based framework for performing software maintenance. Structural code based testing is performed based on changed function names. The proposed framework contains four agents viz. monitor agent, impact agent, search agent and email agent. These agents interact with each other through agent communication language for various testing activities. They stressed the use of intelligent multi agents to perform the regression testing in open-source Internet software applications. Salima *et al.* [114] emphasized the use of multi agents for regression test case generation, prioritization through monitoring the code for changes. Test Prioritize agent uses code coverage for prioritization of test cases. Srivastava and Kim [115] emphasized the use of agent for behavioral based approach for regression test case generation and execution. They proposed monitor agent and test generator

agent for agent-based regression testing. An algorithm is presented to study the changes in code and behavior. Different agents viz. monitor agent, test case generator agent are presented to perform code comparison and test case generation respectively. Malz and Göhne [123] proposed adaptive agent-based test management system for prioritization of test cases based on test importance i.e. local priority and global priority. Malz *et al.* [125] extended his work to fuzzy based test cases prioritization.

2.8.1 Advantages of Agent-based Regression Testing

Various researchers have recommended the use of agents for increasing the efficiency and preferred these systems for distributed testing. In the age of Internet, truly distributed systems are in place. In distributed environments, mobile agent-based testing can save computing resources to a large extent.

Mobile agents based on BDI approach can be used to enhance performance testing of web services. Mobile agents based on fuzzy based techniques can be used to enhance test case prioritization. Zunliang *et al.* [113] proposed a multi-agent-based actionable knowledge model to support regression testing of real world applications. Different agents are assigned the task to operate actionable knowledge model for test case generation and execution to support regression testing. Srivastava and Kim [115] proposed an agent-based regression testing framework to simplify time and work complexities related with regression testing. Askarunisa [122] proposed agent-based regression test case prioritization to reduce the time and cost associated with testing. Testing tasks are distributed among various agents to compute coverage, to identify, compare and implement suitable type of prioritization techniques for fault detection. Bassil [116] proposed multi-agent platforms for testing multiple web services using different distributed environments. Testing tasks are distributed among parallel agents to identify changes in service oriented architecture. These agents act as load balancers across various servers to get optimal resource utilization and maximize throughput.

Software failures have always been a challenge for software tester making it laborious, time consuming process. Regression testing has always been a bottleneck for software industry. To cut short on schedule and cost, mobile based software testing can be a promising alternative. The shortcomings in regression test case generation techniques using localized environment

can be addressed by software agents. Testing systems developed with agent oriented approach will help in decision making of processes, better distribution of information, tasks, activities and resources [84]. Agents possess properties like mobility, autonomy and decision making. These characteristics of agents enable them to play the role of a tester by performing test cases planning, generation and execution. Multiple parallel agents can be initiated simultaneously in distributed environment to collect changes in various versions of software artifacts. Agents can be assigned the task of structural analysis, functional testing, model comparison and formal specifications comparison to identify changes in comparatively less time. This will result in developing techniques having more coverage and with reduced human effort.

2.9 Discussions

Due to its dynamic nature, testing always offers challenges to the developer as well as the tester. Test case generation, being effort intensive and difficult, has become seeds for further research in the field of testing. Over the past few years, agent-based testing has emerged as a potential alternative. In our SLR, we analyzed 123 articles out of 9781.

We made quantitative review of existing regression test case generation and agent-based testing studies. Based on our three research questions, we systematically extracted the data from primary studies and then synthesized it to identify the various frameworks, techniques, methods and platforms for regression test case generation and agent-based software testing. Table 2.11 contains various pros and cons of various testing techniques and approaches used for regression test case generation.

Table 2.11: Pros and Cons of test case generation approaches in regression testing

Model based testing
Pros Scenario based modeling enables coverage analysis for requirements, highlighting risk and usage analysis [22]. Integration of control flows, events in sequence diagram may increase number of test case generation [29]. Dependence analysis and EFSM model based test case generation may require less manual work [23] and helps in reducing the overhead for test case generation. Agents can be applied to explore modeling techniques based on UML activity, sequence diagrams, UML artifacts to generate test scenarios and enhancement of test case generation. Model based regression approach based on UML/OCL behavioral models may reduce number of tests cases [18].
Cons Path expression based techniques may require manual activities for state unrolling and merging [21]. It requires additional effort for tester in terms of UML modeling, state machine etc.
Structural testing
Pros Component based systems with complexity of individual methods is generally limited by design may offer an ideal target for symbolic execution for regression test case generation using symbolic execution [54]. Path pruning based on path oriented and goal oriented approach can be used to enhance regression test case [53]. Test case scatter search using branch coverage may be applied for better test case generation [64]. Particle swarm optimization (PSO) being simple and easier to implement is an effective technique by adjusting the few parameters that makes it ideal for the generation new test cases in regression testing [59]. Agent-based approach may be applied to path based testing. Agent-based on BDI approach can be applied for better web testing. Symbolic execution can be generalized to apply on Java source code. Assertions based on searching can be used to generate test case and solve propagation aspect of regression testing [63]**.**
Cons Most of the VBT and CaR tools are applied to smaller programs, so it affects scalability of the testing approach [47]. Static and dynamic approach use behaviors for searching suitable test cases and suffer from state space explosion problem [61]

Functional testing
Pros Specification testing of object oriented programs based on container classes assists practitioners to collect the unusual output of the theory of algorithms community [20]. Functional testing can be improved by using scenario-based technique by enforcing the hierarchical definition and scenarios modeling at various abstraction levels. Approaches using Extensible BPEL Flow Graph (XBFG) elements along with nodes and edges can help finding out paths to be tested by path comparison. The analysis of path conditions cuts down the quantity of test cases generated.
Cons Generation algorithms are complex and tester needs to understand the algorithm output. [20]. End to end testing [39] is complex and requires a concrete approach for better test case generation.
Formal specifications based testing
Pros Method based on Guttag's algebraic specification may be used for regression test case generation for Junit using lesser formalism [64]. Java axioms exploit method interactions and may find more errors in the algorithm or GUI class [64]. State/event tree approach model the behavior of system using scenarios and states may be used for formal analysis including completeness and consistency checking and coverage analysis of state-based embedded systems [65]. Model checking based on comparison of state transitions may reduce testing costs and produces test cases with greater confidence [67]. Agents can be applied in web testing using HPN. The BPEL-based web service composition modeled by HPN can be easily referenced by test case generation.
Cons It demands regular effort in terms of formal specification language e.g. Object Z, OCL etc. Junit Axiom (JAX) framework provides a manual pattern and user needs to write test code. The main problem of model-checker based approaches is the performance issue. If the model is too complex, then test-case generation may take very long time [68].
Mutation based testing
Pros Augmented dynamic symbolic execution can be applied for boundary value analysis, mutation analysis, logical coverage criteria and error conditions [71]. Assertions based EvoSuite [72] uses branches may achieve high structural code coverage for Java programs. Use of dynamic symbolic execution may result in efficient and effective for mutant killing.
Cons

Pex Mutator introduces many constraints as mutants of the program under test into the corresponding meta-program, making it expensive for the DSE engine to generate test inputs for a large number of branches [71]. EvoSuite is limited to single threading applications [72]. It has scalability issues and can be costly in terms of time and effort and require manual intervene.
Random testing
Pros Ant colony based random testing may lead to better optimization of test cases [74]. Random testing based on feedback-directed approach using assertions may be applied to large, real-world software systems. Guided random testing based test case generation may be used to verify the precision and quality of AspectJ programs [73].
Cons Ant colony based random testing takes longer run and is less efficient as compare to other methods [74].

2.10 Summary

For the validation of our study, we synthesized the data to present the year wise systematic maps showing the broader areas and trends of agent-based software testing. Furthermore, we investigated existing techniques and approaches of agent-based regression testing. Various international conferences have been conducted in the field of agent-based software engineering indicating the importance as well as need for implementation of agent-based software testing. Most of the studies have recommended the use of agents to reduce the testing time and effort.

We identified 57 papers and categorized them in 7 studies. We found that, in most of the studies only prototype is suggested with formulated design. But none of the study in agent-based testing has evaluated proposed model on industrial applications. So there is a need to develop more sophisticated techniques with orientation towards industry requirement. The reliability and scalability of most of the agent-based software testing techniques is still an issue. We also observed that there is a call to address the dearth of empirical studies in the area of agent-based software testing systems.

Chapter – 3

Mobile Agent-based Regression Test Case Generation Using Model and Formal Specifications

In this chapter, Section 3.1 contains background of the study. Section 3.2 presents methodology. Experimental setup is presented in section 3.3. Section 3.4 contains Notations used and section 3.5 have algorithms related to our technique. Case study of Library System and ATM System is presented in section 3.6. Section 3.7 and 3.8 covers Test Sequence. Section 3.9 contains Results and Discussions and Section 3.10 holds Threats to Validity.

Several approaches and techniques including structural testing, functional testing, model based testing and formal specification based testing have been suggested by researchers for regression test case generation. In model based testing, the UML class diagram, activity diagram and sequence diagram alone do not provide all the relevant aspects of specifications. In literature, constraints are not considered in UML diagrams. These constraints about the objects of the model can be defined and incorporated using formal languages such as Object-Z and OCL. Thus combining model based approach with formal specifications will lead to a better understanding of syntactic and semantic changes in modified code. It will facilitate in the identification of reusable, obsolete and newly generated test cases for regression testing. Additionally, mobile agent-based technology will assist in collecting changes from various stakeholders in distributed environment. It assists in reducing effort and time required for regression test case generation.

3.1 Background

We identified studies in existing literature on regression test case generation using agents, model based and formal specifications techniques. Araújo & Moreira [141] proposed a set of rules to for the transformation of collaborations and collaboration diagram into an object-oriented formal notation. Ricca & Tonella [21] proposed regression test case generation based

on path expression of UML model using Reweb and test web tools. Miao *et al.* [142] proposed a method of formalizing static and dynamic UML models using Object–Z to automate the transforming process. Roussev [143] presented Newtonian based approach to generate formal specifications in OCL and class diagrams from use case model. Use case specifications are obtained with a state machine using sequence of model transformations. Tsai *et al.* [65] proposed regression test case generation through specifications using OCL constraints. Test scenarios are generated through the paths of a tree. Cavarra *et al.* [66] proposed test case generation based on formal and behavioral semantics by using UML diagrams. Cho *et al.* [29] used self-call messages from UML sequence diagram for the generation of test cases for testing web applications. Dhavachelvan & Uma [97] presented an agent-based software testing framework for mutation testing and capability testing. Different agents are proposed to conduct testing at Autonomous Unit Level (AUL) and Inter-Procedural Level (IPL). Yu *et al.* [83] used agents to explore "def use" pairs for testing web applications using methods, objects and object cluster level testing. Data flow and Control flow statements are represented through control flow graphs. Minsong *et al.* [144] used Extended Finite State Machine (EFSM) to represent specifications of the software for regression test case generation. Salima *et al.* [114] proposed multi agents to monitor changes in code for generating regression test cases. Gorthi *et al.* [25] presented behavior slicing through use case activity diagrams to generate regression test cases. Chen *et al.* [26] proposed test case generation based on execution trace using activity diagrams. Siwen and Jun [103] proposed agent-based approach to generate test cases from UML class diagram using Graph Miner tool. Askarunisa *et al.* [69] proposed test case generation for web services using reduction techniques namely Pair-Wise Testing (PWT) and Orthogonal Array Testing (OAT). Web service structure and conditions for service rules are represented using UML and OCL respectively. Filhos *et al.* [27] used tool for test case generation and prioritization based on UML. Naslavsky *et al.* [30] generated test cases using traceability links and representing sequence diagrams as control flow graphs. Ye *et al.* [34] used behavior changes through activity diagrams based on execution traces for regression test case generation. Malz *et al.* [125] suggested test case prioritization using fuzzy complexity and agents based on Java Agent Development Environment (JADE) [145]. Vincent *et al.* [33] used use cases, state charts and collaboration diagrams to generate regression test cases for object oriented software. Zech *et al.* [35] applied model based testing approaches on UML Testing Profile (UTP) and

Telling TestStories (TTS) for regression test case generation. Ali & Society [146] proposed model based test case generation using heuristics search targeted to OCL constraints to guide test data generation using genetic algorithm, evolutionary algorithm and alternative variable method. Fourneret *et al.* [36] used UML and OCL model for regression test case generation. Table 3.1 contains regression test case generation techniques using UML diagrams and formal specifications.

Table 3.1: Regression test case generation techniques using UML and formal specifications

S. No.	Reference	Technique Used	Research Gap
1.	Ricca & Tonella [21]	Used UML model for page, hyperlinks, all use, def use, all path for web testing	Needs manual inspection, unable to detect changes in semantics
2.	Cho *et al.* [29]	Compared sequence diagrams using self-call messages of web pages under test	Lacks in-depth code coverage and formal specifications
3.	Briand *et al.* [147]	Categorized regression test cases into Reusable, Retestable, Obsolete on the basis of changes in UML class diagram of old and new version	Lacks formal specifications and unable to identify behavioral changes
4.	Gorthi *et al.* [25]	Comparison of XMLs for nodes generated from use case activity diagrams	Prototype model, needs more case studies to validate this approach
5	Askarunisa *et al.* [69]	Presented reduction techniques based on Pair-Wise Testing (PWT) and Orthogonal	Compared two techniques for testing,

		Array Testing (OAT) to test web services. UML and OCL are used to represent structure of web service and conditions for service rules respectively.	but testing efficiency remains an issue
6.	Filho *et al.* [27]	Comparison of UML model based on traceability links, user-defined critical paths, data coverage and path coverage	Unable to identify all the changes including behavior and specifications
7.	Naslavsky *et al.* [30]	Used sequence diagram to represent model-based control flow graphs, traceability model and compared them	Relies on certain abstraction levels and lacks in-depth code coverage
9.	Cavarra *et al.* [148]	Based on UML models with input labelled transition system representing associated formal and behavioral semantics	Difficult to detect behavior in the absence of other behavioral based UML diagrams
10.	Ye *et al.* [34]	Compared activity diagrams based on execution traces for behavior changes	Unable to identify static changes at class level
11.	Vincent *et al.* [33]	Compared use cases, state chart and collaboration diagrams	Lacks formal specifications and needs validation
12.	Zech *et al.* [35]	Model-based approach using UML Testing Profile (UTP) and Telling TestStories (TTS) based on specifications and model respectively	Lacks OCL queries based on the standardized XML model

13.	Lamancha *et al.* [32]	Proposed model based UML sequence diagram to generate the test behavior	Lacks formal specifications
14.	Fourneret *et al.* [36]	Compared UML/OCL behavioral models of the system expressed in guards/transitions of state charts	Unable to identify static changes
15.	Sapna & Balakrishnan [37]	Regression test case generation based on activity diagrams based on fork join approach and Steiner tree using CFG	Unable to identify static changes at class level
16.	Dahiya *et al.* [77]	Regression test case generation based on class, sequence and activity diagrams	Proposed approach lacks formal specifications
17.	Farooq *et al.* [182]	Used class diagram to identify state based changes in old code and new code	Unable to identify changes at behavior level and constraint level
18.	Mansour *et al.* [149]	Compared interaction view diagrams to detect changes at action level for generating test cases	Lacks in-depth code coverage and formal specifications

From Table 3.1, we have observe that most of the techniques are limited to either UML models or formal specifications. Only a few researchers have integrated model and formal specifications based approach for in-depth code coverage. In existing work, the combined use of model based approach, Object-Z, OCL and mobile agent technology could not be established for regression test case generation.

Formal speciation language Object-Z is an object-oriented extension of the formal specification language Z. It contains all the syntactic structures of Z along with class notation and object-

oriented features. An object-Z class is represented syntactically as a named box with generic parameters [150]. Object-Z specification consists of several class definitions preceded by global definitions, local types, constant definitions, state schema and operations [150].

OCL is a formal specification language used to enhance description in the form of invariables and constraints not represented through UML diagrams [151]. Thus, use of formal speciation languages OCL and Object-Z may increase clarity of specifications for regression test case generation through enhanced structuring. Further, formal properties of the codes can be explored using formal specification language for semantic and syntactic comparison.

3.2. Methodology

In this section, we have presented agent-based technique for regression test case generation. This multi-agent model consists of five agents viz. model comparison agent, behavior comparison agent, specification comparison agent, impact analysis agent and regression test case generation agent. Agents are initiated from the host machine to different testing stakeholder machines in the distributed environment. These agents perform comparison tasks assigned to them including UML class comparison, behavior comparison and specifications comparison at different machines. Model comparison agent compares UML class, behavior agent compares sequence file and specification agent compares Object-Z and OCL of new and old versions. All these tasks are simultaneously performed by the agents at different machines, resulting in comparatively less time for test case generation. These agents submit the results back to impact analysis agent at host machine. Impact analysis agent performs impact analysis and submits the reports to test case generation agent as shown in Fig. 3.1. For designing our agent framework, we used open source JADE framework [145].

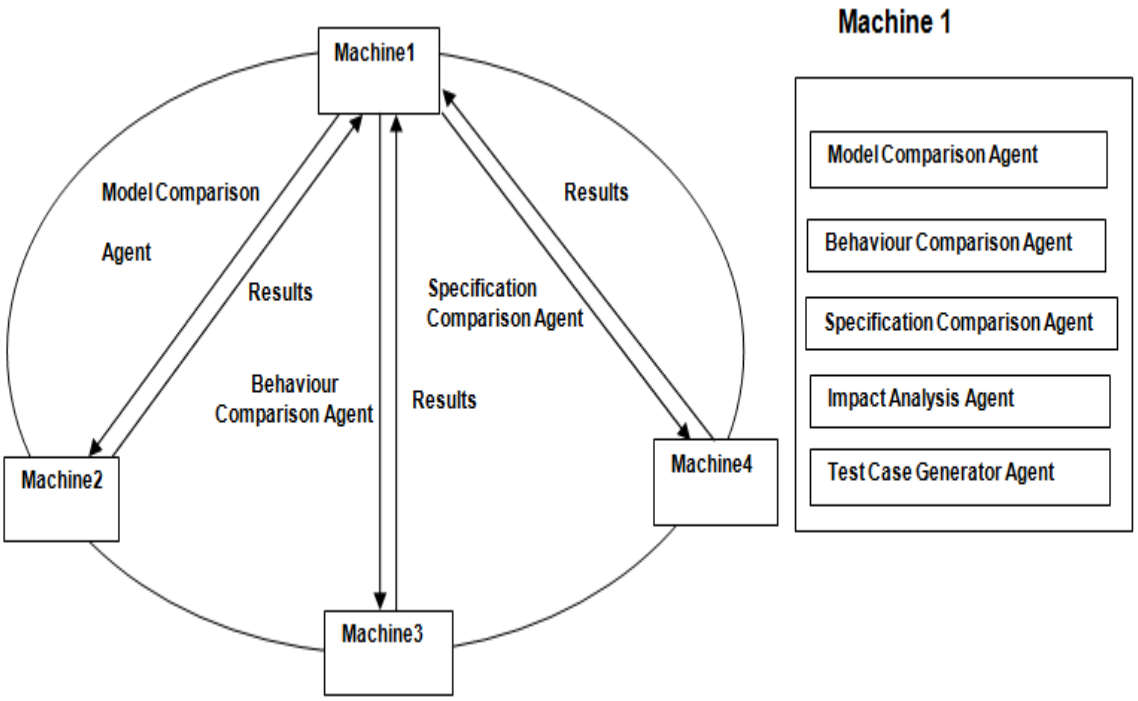


Fig 3.1: Agents based Regression Test Case Generation Framework

3.3 Experimental Setup

We analyzed Library system and ATM Java codes from github repositories. These were reverse engineered into UML class diagrams and sequence diagram using Object Aid tool [152] in Eclipse. These files were converted into XML format and their associated Object-Z and OCL codes were developed using Eclipse CZT tool [153]. Agents based on JADE environment are designed to perform comparison tasks in the distributed environment as shown in Fig. 3.2.

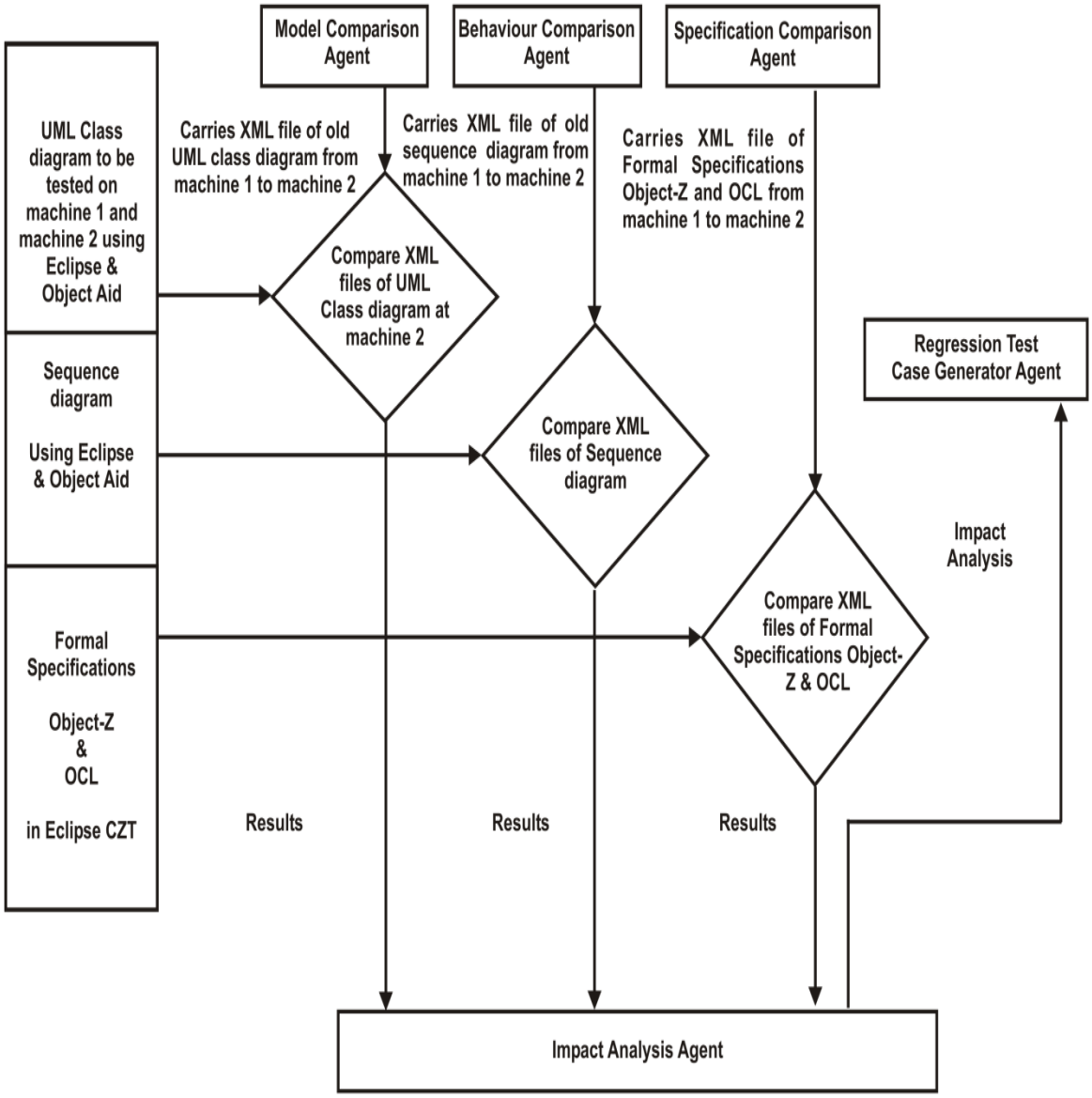


Fig 3.2: Comparison Tasks Performed by Different Agents

3.4 Notations Used

Our technique is based on the assumption that identification of changes in the classes, their associations, dependencies, attributes, operations, state invariants and state schema will lead to regression test case generation. We used the following notations as summarized in the Table 3.2 for our research.

Table 3.2: Notations used

Notation	Description
CS & CS'	Classes from original and modified UML class diagrams respectively
CD & CD'	Class dependencies in original and modified UML class diagrams respectively
Cas & Cas'	Class associations from original and modified class diagrams respectively
CaT & CaT'	Attributes from original and modified class diagram respectively
CoP & CoP'	Operations from original and modified class diagram respectively
CSq & CSq'	Classes from original and modified sequence diagrams respectively
CZ & CZ'	Class name in original and modified Object-Z formal specifications respectively
Caz & Caz'	Attributes in original and modified Object-Z formal specifications respectively
Zop & Zop'	Operations in original and modified Object-Z formal specifications respectively
OCL & OCL'	OCL constraint in original and modified code respectively
O(id) & O(id)'	Objects in old and modified sequence diagram respectively
I & I'	Interactions in old and modified sequence diagram respectively

IC & IC'	Interactions call occurrence method i.e. synchronous/asynchronous method in old and modified sequence diagram
Treuse_c, Tobs_c, Tretest_c	Set of reusable, obsolete & retestable classes identified in UML class diagram
Treuse_a, Tobs_a, Tretest_a,	Set of reusable, obsolete & retestable attributes in UML class diagram
Treuse_d, Tretest_d	Set of reusable & retestable classes identified with dependencies in UML class diagram
Treuse_as, Tretest_as	Set of reusable & retestable classes identified with association in UML class diagram
Treuse_cop, Tretest_cop	Set of reusable & retestable operations in UML class diagram
Treuse_sq, Tobs_sq, Tretest_sq	Set of reusable, obsolete & retestable classes identified in sequence diagram
Treuse_o, Tretest_o	Set of reusable & retestable objects identified in sequence diagram
Treuse_sop, Tretest_sop	Set of reusable & retestable operations identified in sequence diagram
Treuse-z, Tobs_z, Tretest_z	Set of reusable, obsolete & retestable classes identified in Z specification
Treuse_az, Tobs_az, Tretest_az	Set of reusable, obsolete & retestable attributes identified in Z specification

Treuse_ocl, Tretest_ocl	Set of reusable & retestable constraints identified in OCL specification

3.5 Algorithms

Different agents are designed in JADE to identify changes impact analyses and generate regression test cases. For calculating the impact of changes for regression test case generation, we assigned 30% weight to changes identified in UML class diagram, 30% to changes identified in sequence diagram and 30% to changes in Object-Z, where as 10% weight is given to OCL constraints. In order to identify the time taken by the agents to complete their task, we recorded system current time of host machine during sending and receiving of agents. For this *public void beforeMove()* and *public void afterMove()* operations in JADE were used [14]. The algorithms for model comparison agent, behavior comparison agent and specifications comparison agent are given below.

A. Model Comparison Agent

Algorithm: Model_agent(XML1, XML2)

Input: XML files of UML class diagrams of old and new codes from machine 1 and machine 2 respectively

Output: Treuse_c, Tobs_c, Tretest_c, Treuse_a, Tobs_a, Tretest_a, Treuse_cop, Tretest_cop, Treuse_d, Tretest_d, Treuse_as, Tretest_as

Description: This algorithm generates a set of retestable, reusable, obsolete - classes, attributes, operations in class diagram and submits results to impact analysis agent at host machine 1

Procedure

Start

Initialize JADE model comparison agent

// Carries XML file of old model from machine1 to machine 2

Call Extract_static_details(XML1)

// Parsing XML file of UML class diagram-old model

Call Extract_static_details(XML2)

// Parsing XML file of UML class diagram-newly modified code

∀ (C ∈ CS) and ∀ (C’ ∈ CS’)

Begin

Treuse_c←(CS ∩ CS’) // Set of reusable classes

Tobs_c←(CS U CS’)-CS’ // Set of obsolete classes

Tretest_c←(CS U CS’)-CS // Set of retestable classes in CS’

End

∀C∈CS ∃(CaT ∈ C) and ∀C’∈CS’ ∃(CaT’∈ C’)

Begin

Treuse_a←(CaT ∩ CaT’) // Set of reusable attributes

Tobs_a←(CaT U CaT’)-CaT’ // Set of obsolete attributes

Tretest_a←(CaT U CaT’)-CaT // Set of retestable attributes

End

∀C∈CS∃(CD∈C) and ∀C’∈CS’∃(CD’∈C’)

Begin

If (CD(S,T) && CD’(S,T))= = 0

// Comparing source and target dependency of each class

Treuse_d←∃cs ∈ CS’

// If dependency of corresponding classes are same, store in reusable set

else

Tretest_d← ∃cs'∈ CS'

// If dependency of old class and new class is not same then store corresponding class from CS' in retestable set

End

∀C ∈ CS ∃ (Cas ∈ C) and ∀C'∈ CS' ∃ (Cas'∈ C')

Begin

If (Cas(S,T) && Cas'(S,T))= =0 // Comparing for association of corresponding classes

Tresuse_as← ∃c'∈ CS'

// If association of corresponding classes are same then store corresponding class in CS' set of reusable set

else

Tretest_as←∃c'∈ CS'

// If association of corresponding classes is not same then store corresponding class in CS' in retestable set

∀C ∈ CS ∃(CoP ∈ C) and ∀C'∈ CS' ∃(CoP' ∈ C')

End

Begin

If (CoP(parameters, return type) && CoP'(parameters, return type))!=0

// Comparing corresponding operations with arguments and return type

Tresuse_cop← ∃CoP ∈ C'

// If corresponding operations in both classes are same store into reusable set

else

Tretest_cop← ∃CoP'∈ C'

// If corresponding operations in both classes are not same then store into retestable set

Submit results to impact analysis agent at machine1

End

End

Extract_static_details(XML file)

// Input XML code of UML class diagram to be parsed

{

Extract classes, assign unique id to each class

Extract association and dependencies of each class

Extract attributes of each class

Extract operations in each class along with their parameters and return type

}

B. Behavior Comparison Agent

Algorithm: Behavior_agent(XML1, XML2)

Input: XML files of sequence diagrams of old and new codes from machine 1 and machine 3 respectively

Output: Treuse_sq, Tobs_sq, Tretest_sq, Treuse_o, Tretest_o Treuse_sop, Tretest_sop

Description: This algorithm generates a set of retestable, reusable, obsolete-classes, objects, interactions in sequence diagram and submits results to impact analysis agent at host machine 1

Procedure

Start

Initialize JADE behavior agent from machine 1

// Carries XML of old code to machine 3

Call Extract_dynamic_details(XML1) // Parsing XML file of sequence diagram-old code

Call Extract_dynamic_details(XML2) // Parsing XML file of sequence diagram-newly modified code

Begin

Treuse_sq←(CSq ∩ CSq') // Set of reusable classes

Tobs_sq← (CSq U CSq')-CSq' // Set of obsolete classes

Tretest_sq← (CSq U CSq')-CSq // Set of re-testable classes

End

∀O∈C∈ CSq and ∀O'∈C'∈ CSq'

Begin

If ((O(ID) && O'(ID))= = 0

// Comparing corresponding objects in both sequence diagram

Tresuse_o← O' // Storing objects in reusable set of objects

else

Tretest_o←∀ O' // Storing objects in retestable set of objects

∀I∈O and ∀I'∈O' // Comparing corresponding interactions among objects

End

If I(C) && I'(C)

// Comparing call method of corresponding interactions i.e. Asynchronous /Synchronous

Begin

If (I (S,T) && (I'(S,T))!=0

// Comparing corresponding interactions along with source and target id

Tresuse_sop← ∃i'∈ I''

// If corresponding interactions in both classes are same store into reusable test cases

Else

Tretest_sop← ∃i'∈I'

// If corresponding interactions in both classes are not same store into retestable test cases

End

End

Submit results to impact analysis agent at machine1

End

Extract_dynamic_details (XML file)

// Input XML code of specification diagram to be parsed

{

Extract all objects of each class

Extract source of interactions in each class

Extract receiver of interactions in each class

Extract call method of interactions i.e. asynchronous/synchronous

}

C. Specification Comparison Agent

Algorithm: Specification _agent (XML1, XML2)

Input: XML file containing Object-Z codes and OCL codes of old code and newly modified code

Output: Treuse_z, Tobs_z, Tretest_z, Treuse_az, Tobs_az, Tretest_az, Treuse_ocl, Tretest_ocl

Description: This algorithm generates a set of retestable, reusable, obsolete classes, attributes in operations, constraints in formal specifications Object-Z & OCL and submits results to impact analysis agent at host machine

Procedure

Start

Initialize JADE specification agent from machine 1 carries XML file to machine 2

Call Extract_objectZ (XML1) // Parsing Object-Z for extraction-old code

Call Extract_objectZ (XML2) // Parsing Object-Z for extraction-New modified code

$\forall (C \in CZ)$ and $\forall (C' \in CZ')$

Begin

Treuse_z$\leftarrow(CZ \cap CZ')$ // Set of reusable classes

Tobs_z← (CZ U CZ')-CZ' // Set of obsolete classes

Tretest_z← (CZ U CZ')-CZ // Set of retestable classes in CZ'

End

∀C ∈ CZ ∃(ZoP ∈ C) and ∀C'∈ CZ' ∃(ZoP' ∈ C')

Begin

∀C∈CZ ∃(Caz ∈ C) and ∀C'∈CZ' ∃(Caz'∈ C')

Begin

Treuse_az←(Caz ∩ Caz') // Set of reusable attributes in each operations

Tobs_az← (Caz U Caz')-Caz' // Set of obsolete attributes in each operation

Tretest_az← (Caz U Caz')-Caz // Set of retestable attributes in each operation

End

End

Call Extract_OCL_details (XML1)

Call Extract_OCL_details (XML2)

∀C ∃(ocl ∈ OCL) and ∀ C' ∃(ocl ∈ OCL')

Begin

Treuse_ocl←(OCL ∩ OCL') // Set of reusable constraints in each class

Tretest_ocl← (OCL U OCL')-OCL // Set of retestable constraints in each class

End

Submit results to impact analysis agent at machine1

End

Extract_objectZ_details (XML file) // XML code of Object-Z to be parsed

{

Extract all classes

Extract all operations along with their parameters and return type

Extract attributes in side operations}

Extract_OCL_details(XML1) // Input XML code of OCL to be parsed

{Extract all constraints}

D. Test Case Generation

Algorithm: TCG(XML1, XML2)

Input: XML files of UML class diagrams, sequence diagram, Object_Z and OCL of old and new modified code

Output: Set of test cases

Start

Call Model_agent(XML1, XML2) // For model comparison - UML class diagram

Call Behaviour_agent(XML1, XML2) // For behavior comparison - sequence diagram

Call Specificaiton_agent(XML1, XML2) // For Object-Z and OCL comparison

Call RTCG_agent**(**Tretest_c, Tretest_a, Tretest_d, Tretest_as, Tretest_cop, Tretest_sq, Tretest_sop, Tretest_z, Tretest_az, Tretest_ocl, Treuse_c, Treuse_a Tresuse_d, Treuse_as, Treuse_cop, Treuse_sq, Treuse_sop, Treuse_z, Treuse_az, Treuse_ocl)

End

E. Impact Analysis Agent

Input: Changes submitted by model agent, behavior agent and specification agent

Output: Total Change Impact (TCI)

Begin

CUML= ((No. of classes with changes + No. of attributes with changes + No. of operations with changes) *30) /100

// Change identification in terms of UML class diagram

CSEQ = (No. of operations with changes * 30) /100

// Change identification in terms of sequence diagram

CZML= ((No. of classes with changes + No. of attributes with changes + No. of operations with changes) * 30) /100

// Change identification in terms of Object-Z

COCL= ((No. of OCL constraints with changes) * 10) /100

// Change identification in terms of OCL

Total change impact TCI = CUML+ CSEQ + CZML + COCL

End

F. Regression Test Case Generation Agent

Algorithm: RTCG_agent(Tretest_c, Tretest_a, Tretest_d, Tretest_as, Tretest_cop, Tretest_sq, Tretest_sop, Tretest_z, Tretest_az, Tretest_ocl, Treuse_c, Treuse_a Tresuse_d, Treuse_as, Treuse_cop, Treuse_sq, Treuse_sop, Treuse_z, Treuse_az, Treuse_ocl)

Input: Tretest_c, Tretest_a, Tretest_d, Tretest_as, Tretest_cop, Tretest_sq, Tretest_sop, Tretest_z, Tretest_az, Tretest_ocl, Treuse_c, Treuse_a Tresuse_d, Treuse_as, Treuse_cop, Treuse_sq, Treuse_sop, Treuse_z, Treuse_az, Treuse_ocl

Output: Set of test cases from Tretest_cf, Treuse_cf, Treuse_af, Tretest_af, Tretest_opf, Treuse_opf

Description: This algorithm generates and selects a set of test cases from class diagram, sequence diagram and formal specifications

Start

Tretest_cf← Tretest_c ∩ Tretest_d ∩ Tretest_as ∩ Tretest_sq ∩ Tretest_z

// Set of all retestable classes

Treuse_cf← Treuse_c ∩ Tresuse_d ∩ Treuse_as ∩ Treuse_op

// Set of all reusable classes

Treuse_af← Treuse_a ∩ Treuse_az // Set of all reusable attributes

Tretest_af← Tretest_a ∩ Tretest_az // Set of all retestable attributes

Tretest_opf← Tretest_cop ∩ Tretest_sop// Set of all retestable operations

Treuse_opf← Treuse_cop ∩ Tresuse_sop // Set of reusable operations

test_generation (Tretest_cf, Treset_o, Treset_af, Tretest_opf, Tretest_ocl, Tretest_sq, Tretest_z, Tretest_cop, Tretest_sop)

Begin

Select any class from Treset_cf

Start traversing it in breadth wise search and mark each class visited

Begin

For each class in Treset_cf

Select corresponding set of objects, attributes, operation, interactions and constraints from Treset_o, Treset_af, Tretest_opf, Tretest_cf and start traversing breadth wise search, mark each visited

End

End

test_selection(Treuse_c, Treuse_d, Treuse_as, Treuse_sq, Treuse_z, Treuse_cop, Treuse_sop)

Begin

Select any class from Treuse_cf

Start traversing it breadth wise search and mark each class visited

Begin

For each class in Treuse_cf

Select corresponding set of objects, attributes, operation, interactions and constraints from Treuse_o, Treuse_as, Treuse_opf, Treuse_ocl and start traversing breadth wise search, mark each visited

End

End

End

3.6 Case Study of Library and ATM System

We analyzed Java code of the library management system, a computerized system to maintain all work of a library. Various features include user login, search book, add new book, new member etc. User is authenticated through user_ID and password before logging into the system. It includes various modules such as add new book, add new member, issue books, return books, delete book and search book. Books can be added in the system by using book_ID (unique_ID for every book), book name, book _author and book category. Members can be added into the system using member_ID, member name, member password, entry date and category. Books can be issued, returned, searched, deleted using issue book, return book, search book and delete book receptively. For regression test case generation, we applied changes in

the class diagram. We added additional functionalities for issue book and penalty module. We made changes that reference books will be issued for a period of one week as compared to two weeks in old code. Moreover there would be a greater fine on reference books as compared to normal books and two additional modifications were added in the issue book and the fine book modules.

The UML class diagrams were drawn using Object Aid tool [152] represents old code as shown in the Fig. 3.3. These changes also affected sequence diagram as method containing attributes related to fine were changed but these were not visible in the UML class diagram. The sequence diagram of the library system is drawn using the Object Aid tool [152] as shown in Fig. 3.4. We converted Java code of the library system into Object-Z and OCL to identify the changes in the constraints and operations. These changes in constraints are not visible in UML class diagram and sequence diagram.

The second subject system analyzed for our study is ATM from the open source repositories. In the ATM system, user interacts with the graphical user interface and logins to the system using authenticated user id and passwords. The various transactions that can be performed in the ATM system are account verification, balance enquiry, withdrawals and deposits.

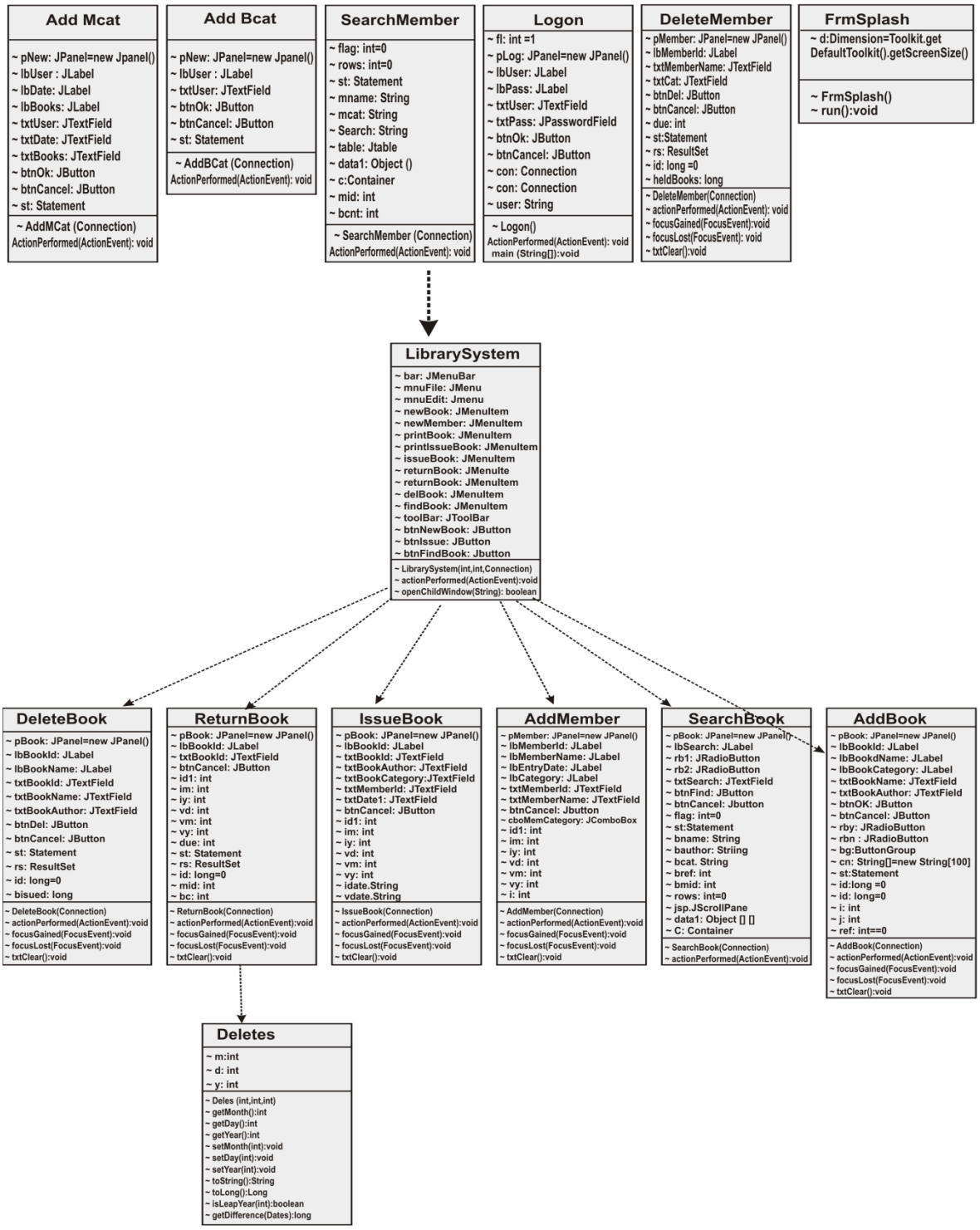


Fig 3.3: UML Class Diagram of Library System

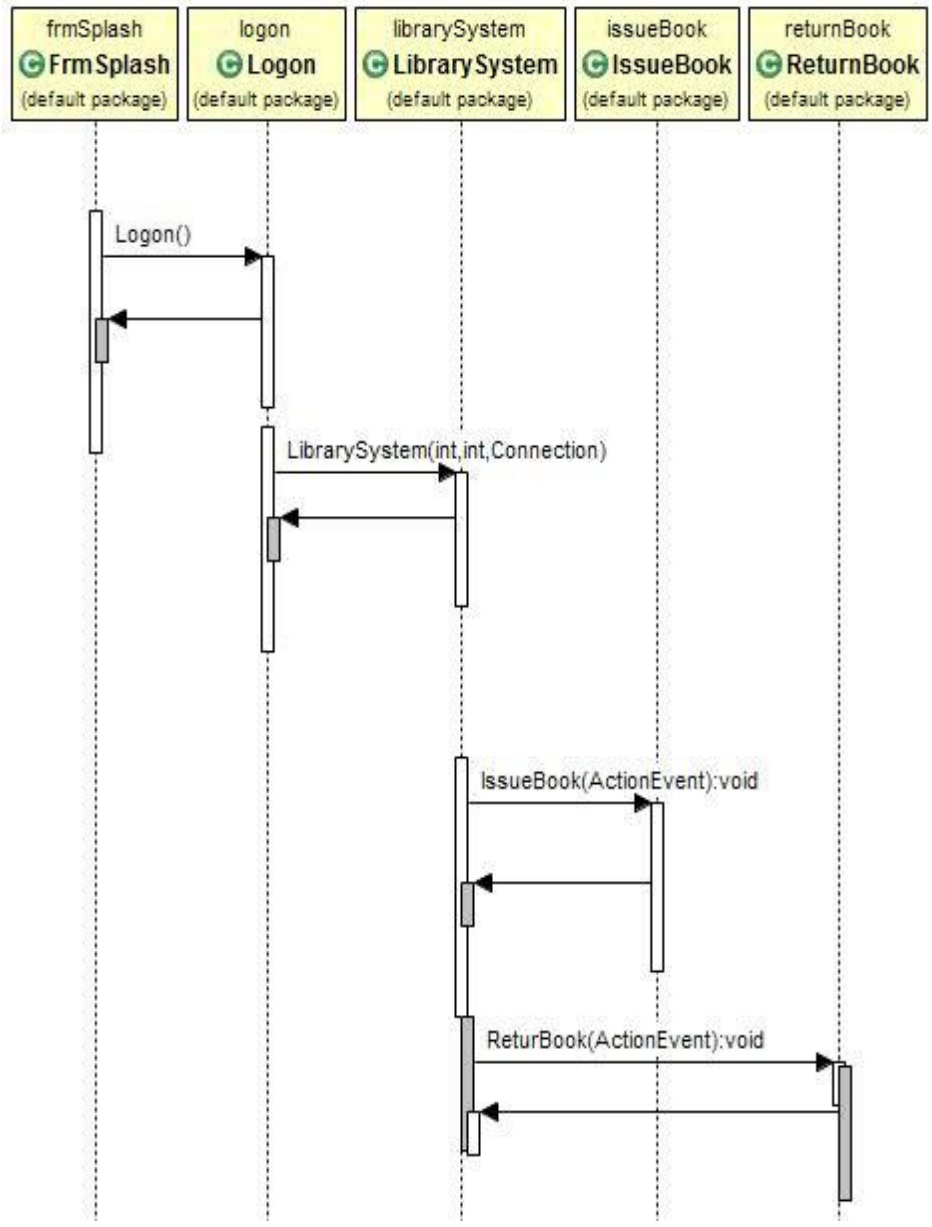


Fig 3.4: Sequence Diagram of Library System

These diagrams were compared to detect changes in the classes, attributes, operations and associations etc.

3.7 Test Sequences Corresponding to the Attributes and Operations

We designed a set of test sequences required to test library system as shown in Table 3.3. Regression test case agent uses these sequences to further generate and categorizes test cases.

Table 3.3: Test sequences for library system

Test Sequences No.	**Test Path**
TS1	Logon->LibrarySystem->AddBook->BookId->BookName->BookAuthor -> Reference-> BookCategory-> actionPerformed()-> FocusGained()-> FocusLost()
TS2	Logon->LibrarySystem->AddMember->MemberId->MemberName-> MemberPwd->EntryDate->Category->actionPerformed()-> FocusGained()->FocusLost()
TS3	Logon->LibrarySystem->IssueBook->BookId->BookName-> BookAuthor->BookCategory-> MemberId->MemberName->IssueDate -> ReturnDate->actionPerformed()->FocusGained()-> FocusLost()
TS4	Logon->LibrarySystem->ReturnBook->BookId->BookName-> BookIssuedTo-> actionPerformed()->FocusGained()-> FocusLost()
TS5	Logon->LibrarySystem->DeleteBook->BookId->BookName-> BookAuthor-> actionPerformed() ->FocusGained()-> FocusLost()

We have used CZT Eclipse tool as shown in fig. 3.5 for converting Java source code into corresponding Object-Z code

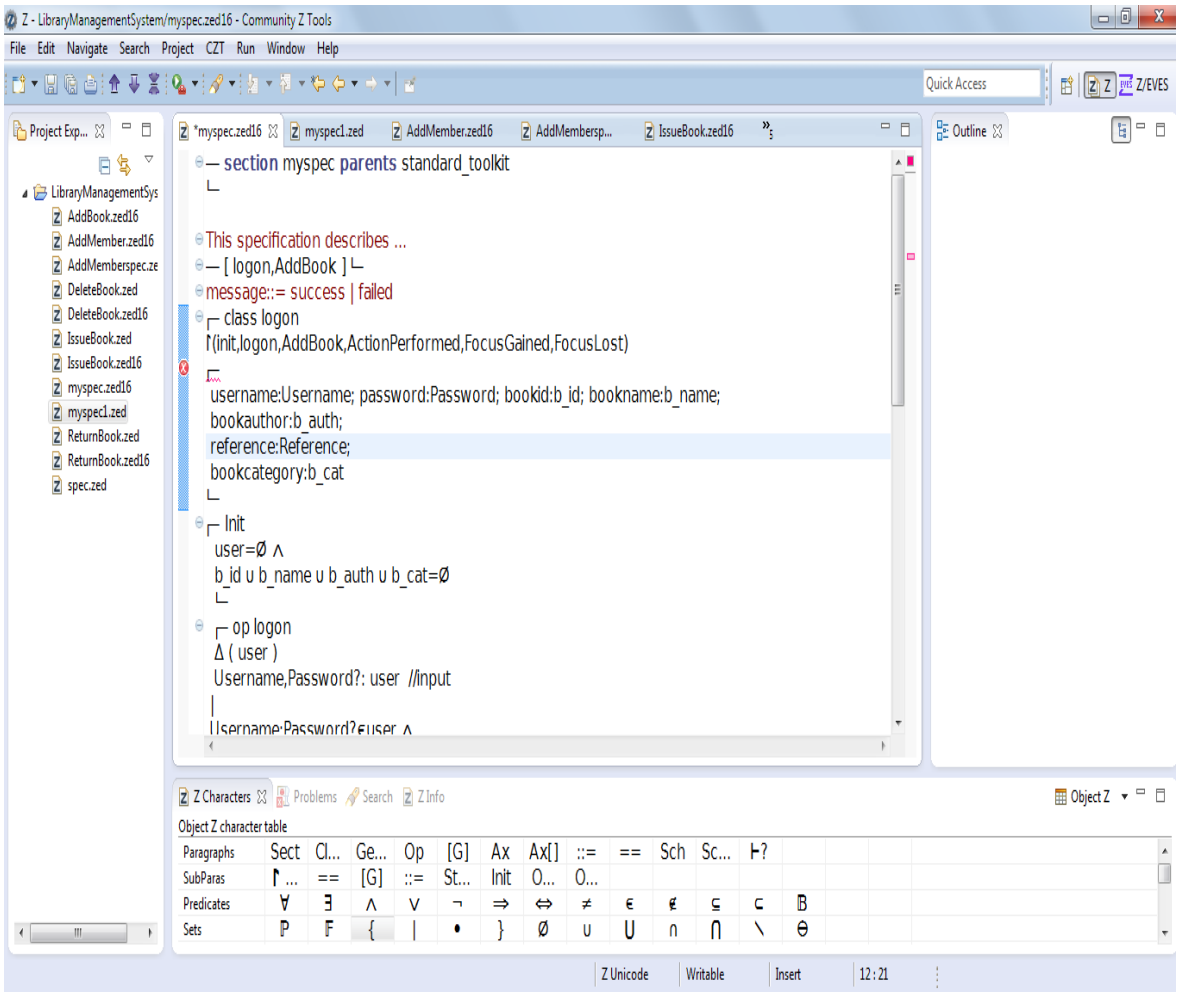


Fig. 3.5: User Interface for CZT tool with Unicode

3.8 Test Sequences in Formal Specifications Object-Z using Eclipse CZT

The formal specification Object–z code corresponding to test sequences as shown in table 3.3 is:

Test Sequence 1

TS 1: Logon ->LibrarySystem ->AddBook ->BookId ->BookName ->BookAuthor -> Reference ->BookCategory ->actionperformed()->focusGained() -> focusLost()

The Object-Z code for test sequence 1 is given below

\begin{document}

begin{zsection}

\SECTION AddBook \parents standard_toolkit, oz_toolkit

end{zsection}

begin{zed}

[logon, AddBook]

end{zed}

begin{zed}

Message::= success | failed

end{zed}

\begin{class}{logon}

\end{class}

\filter(init,logon,Addbook,ActionPerformed,FocusGained,FocusLost)

\begin{state}

username: Username; password: Password;

bookid: b_id; bookname: b_name; bookauthor: b_auth; bookcategory: b_cat;

reference: Reference

\end{state}

\begin{init}

user=\emptyset \land

b-id\cupb_name\cup b_auth=\emptyset

\end{init}

\begin{op}{logon}

\Delta (user)\\

Username, Password?:user // input

\where

Username; Password? \in user\land

Conn!=success

Username; Password? \notin user\land

Conn!=failed

\end{op}

\begin{op}{AddBook}

\Delta (library)\\

b_id, b_name, b_auth, b_cat?:A_book

\where

AddBook'=A_book

\end{op}

\begin{op}{ActionPerformed}

\Delta (action)\\

b_id, b_name, b_auth, b_cat?:A_book

\where

A_book!=ok \lor

A_book!=cancel

\end{op}

\begin{op}{FocusGained}

\end{op}

\begin{op}{FocusLost}

\Delta (book)\\

A_book? \notinAddBook \land

A_book!=record has been saved

A_book? \in AddBook \land

A_book!=problem while saving record

\end{op}

\end{document}

Test Sequence 2

TS 2: Logon ->LibrarySystem->AddMember->MemberId ->MemberName-> MemberPwd -> EntryDate ->Category ->actionPerformed()->focusGained()-> focusLost()

The Object-Z code for test sequence 2 is

\begin{document}

begin{zsection}

\SECTION AddMember \parents standard_toolkit, oz_toolkit

end{zsection}

This specification describes

begin{zed}

[logon, AddMember]

\end{zed}

Message::= success | failed

\begin{class}{logon}

\end{class}

\filter(init, logon, AddMember, ActionPerformed, FocusGained, FocusLost)

\begin{state}

username: Username; password: Password;

Memberid: M_id; MemberName: M_name; MemberPwd: M_pwd;

entrydate; category: cat

\where

cat \sdef faculty \gch student \\

\end{state}

\begin{init}

user \land Member=\emptyset

\end{init}

\begin{op}{logon}

\Delta (user)\\

Username, Password?:user // input

\where

Username; Password? \in user\land

Conn!=success

Username; Password? \notin user\land

Conn!=failed

\end{op}

\begin{op}{AddMember}

\Delta (library)\\

M_id, M_name, M_pwd?:Member \land // input

cat \sdef faculty \gch student?:Member

\where

AddMember'=Member

\end{op}

\begin{op}{ActionPerformed}

\Delta (action)\\

Member? : M

\where

M!: ok \lor

M!: cancel

\end{op}

\begin{op}{FocusGained}

\end{op}

\begin{op}{FocusLost}

\Delta (member)\\

M? \notinAddMember \land

M!= NewMember_added

M? \in AddMember \land

M!=problem while saving record

\end{op}

\end{document}

Test Sequence 3

TS 3: Logon ->LibrarySystem ->IssueBook ->BookId ->BookName ->BookAuthor -> BookCategory>MemberId->MemberName>IssueDate->ReturnDate->actionPerformed()-> focusGained() ->focusLost()

The Object-Z code for test sequence 3 is given below

\begin{document}

\begin{zsection}

\SECTION IssueBook \parents standard_toolkit, oz_toolkit

\end{zsection}

This specification describes

\begin{zed}

[logon, IssueBook]

\end{zed}

Message::= ok | Already_issued | Not_issued

\begin{class}{logon}

\end{class}

\filter(init,logon,IssueBook,ActionPerformed,FocusGained,FocusLost)

\begin{state}

username: Username; password: Password; bookid: b_id; bookname: b_name; bookauthor: b_auth; bookcategory: b_cat; MemberId: M_id; MemberName: M_name

\where

 \forall M:member @ M.Issuedate<ReturnDate

\end{state}

\begin{init}

user= \emptyset \land

b_id \cup b_name \cup b_cat=\emptyset \land

M_id \cup M_name=\emptyset

\end{init}

\begin{op}{logon}

 \Delta (user)\\

 Username, Password?:user // input

\where

Username; Password? \in user\land

  Conn!=success

Username; Password? \notin user\land

  Conn!=failed

\end{op}

\begin{op}{IssueBook}

\Delta (library)\\

b_id, b_name, b_auth?:Book \land // input

b_cat: faculty \gch student?:Book \land

M_id, M_name?:Book

\where

IssueBook'=Book

\end{op}

\begin{op}{ActionPerformed}

\Delta (action)\\

Book?: B

\where

B!=ok \lor

B!=cancel

\end{op}

\begin{op}{FocusGained}

\end{op}

\begin{op}{FocusLost}

\Delta (Book)\\

B? \notinIssueBook \land

B!=Book_issued

B? \in IssueBook \land

B!=Already_issued

\end{op}

\end{document}

Test Sequence 4

TS 4: Logon ->LibrarySystem->ReturnBook ->BookId ->BookName ->BookIssuedTo -> actionperformed() ->focusGained() ->focusLost()

The Object-Z code for test sequence 4 is given below

\begin{document}

begin{zsection}

\SECTION ReturnBook \parents standard_toolkit

end{zsection}

This specification describes

begin{zed}

[logon, ReturnBook]

end{zed}

Message::= fine| No_fine

\begin{class}{logon}

\end{class}

\filter(init,logon,ReturnBook,ActionPerformed,FocusGained,FocusLost)

\begin{state}

username: Username; password: Password; bookid: b_id; bookname: b_name; BookIssuedto: b_iss;

DueDate: D_date; ReturnDate: R_date

\where

D_date + fine \geq 0

\end{state}

\begin{init}

user=\emptyset \land

b_id \cup b_name \cup b_iss=\emptyset

\end{init}

\begin{op}{logon}

\Delta (user)\\

Username, Password?: user // input

\where

Username; Password? \in user\land

Conn!=success

Username; Password? \notin user\land

Conn!=failed

\end{op}

\begin{op}{ReturnBook}

\Delta (library)\\

b_id, b_name?:book \land

b_iss? \in book \implies BookIssued \land

D_date \land R_date?:date

R!=response

\where

D_date \leqR_date \land

R!=No_fine

D_date>R_date \land

R!=fine |

fine?\mapstoD_date-R_date

fine?:F \leq 0 \land F=0

F>0 \land

D_date=D_date + F

\end{op}

\begin{op}{ActionPerformed}

\Delta (action)\\

book!=ReturnBook \lor

book!=cancel

\end{op}

\begin{op}{FocusGained}

\end{op}

\begin{op}{FocusLost}

\Delta (book)\\

b_id? \in ReturnBook \land

b_id!=bookreturn

b_id? \notinReturnBook \land

b_id!=record not found

\end{op}

\end{document}

Test Sequence 5

TS 5: Logon ->LibrarySystem ->DeleteBook ->BookId ->BookName ->BookAuthor actionPerformed()->focusGained()->focusLost()

Object-Z corresponding to test sequence 5 is

\begin{document}

begin{zsection}

\SECTION DeleteBook \parents standard_toolkit

end{zsection}

begin{zed}

[logon, DeleteBook]

end{zed}

Message ::= ok | cancel

\begin{class}{logon}

\end{class}

\filter(init,logon,DeleteBook,ActionPerformed,FocusGained,FocusLost)

\begin{state}

username: Username; password: Password; bookid: b_id; bookname: b_name; bookauthor: b_auth

\end{state}

\begin{init}

user=\emptyset \land

b_id \cup b_name \cup b_auth=\emptyset

\end{init}

\begin{op}{logon}

\Delta (user)\\

Username, Password?:user // input

\where

Username; Password? \in user\land

Conn!=success

Username; Password? \notin user\land

Conn!=failed

\end{op}

\begin{op}{DeleteBook}

\Delta (library)\\

b_id, b_name, b_auth?:B

R!=LMSresponse

\where

B? \in library_books

library_books'=library_books \hide \{B?\}

\end{op}

\begin{op}{ActionPerformed}

\Delta (action)\\

R!=DeleteBook \lor

R!=cancel

\end{op}

\begin{op}{FocusGained}

\end{op}

\begin{op}{FocusLost}

\Delta (book)\\

B? \in library_books \land

B!=DeleteBook

B? \notinlibrary_books \land

B!=record not found

\end{op}

\end{document}

3.9 Results and Discussions

We evaluated our study and categorized results as given in Table 3.4. We observed that agent-based approach takes less time required for regression test case generation. Our technique based on integration of model based and formal specifications has also resulted in depth coverage of code syntactically and semantically.

Table 3.4: Analysis of results submitted by agents

S. No.	**Case study**	**Name of classes in Old version -> New version**	**Number of methods in Old version -> New version**	**Number of changes identified**	**Size of code**	**Time taken by agents to submit results**
1.	Library System	ADDMCat	2->2	0	3K	By model comparison agent 221 ms
		FrmSplash	2->2	0		
		DeleteMember	5->5	0		
		Logon	l->1	0		
		AddBcat	2->3	1		
		LibrarySystem	3->3	0		
		SearchMember	2->2	0		By behavior comparison agent
		SearchBook	2->2	0		
		IssueBook	5->6	1		

		SAddBook	5->5	0		223 ms
		ReturnBook	5->6	1		By comparison specification agent 227 ms
		AddMember	5->5	0		
		DeleteBook	5->5	0		
2.	ATM	ATMCase Study	2->2	0	1K	By model comparison agent 220 ms
		Transaction	5->6	1		
		Deposit	3->3	0		
		ATM	6->7	1		
		GUI	1->1	0		By behavior comparison agent 221 ms
		BalanceCategory	2->2	0		
		Account	7->8	1		
		Withdrawal	3->3	0		
		DepositSlot	4->4	0		By specification comparison agent 223 ms
		Keypad	10->10	0		
		Screen	5->5	0		
		CashDispenser	5->6	1		

Time taken by agents for regression test case generation Tt= (Tm + Tb+ Ts)/3 + Tr

Where Tm = Time taken by model comparison agent

Tb= Time taken by behavior comparison agent

Ts= Time taken by specification comparison agent

Tr= Time taken by regression test case generation agent to generate test cases

For Library system, Tm=221ms, Tb=223ms, Ts=227ms, Tr=224ms, Tt=447.6ms

For ATM system, Tm=220ms, Tb=221ms, Ts=223ms, Tr=226ms, Tt=447.3ms

In most of the existing studies in literature, execution time has not been analyzed or included in their evaluation. In proposed method, all the three comparison agents: model comparison, behavior comparison, specification comparison work concurrently. These agents pass collected changes to Impact analysis agent for further test case generation. In this way, concurrent agents save time as compared to other existing techniques. We have empirically evaluated and compared our results against Library system and ATM with Dahiya [77] and Mansour [149] in terms of the number of test cases generated. We have further enhanced the subject systems with Object-Z and OCL specifications to improve semantic analysis. Encouraging response is observed in comparative analysis shown in Table 3.5.

Table 3.5: Comparative analysis on number of test suites generated

Case Studies ➡		**Library System**		**ATM System**	
Types of Test Suite ⬇	Results from [149]	Results from [77]	Our Results	Results from [77]	Our Results
Types of Sources	UML class, Sequence and Activity diagrams	UML class, Sequence and Activity diagrams	UML class diagram, sequence diagram Object-Z and OCL	UML class, Sequence and Activity diagrams	UML class, Sequence diagrams, Object-Z and OCL
Retestable	7	6	10	3	8

Reusable	76	107	119	158	158
Obsolete	0	2	6	0	0

3.10. Threats to Validity

We could to moderate the threat to validity by improving the in-depth code coverage using formal specifications and UML models. It can detect syntactic changes along with changes in semantics of each operation. We have reduced the time required for test case generation by distributing testing tasks among agents. The external threat to validity is the unavailability of UML models and formal specifications of real world systems in public domain due to various proprietary reasons. Presently, there is no standardized repository of UML models with formal specifications.

Chapter 4

Agent-Based Regression Test Case Generation using Class Diagram, Use Cases, Activity Diagram and Formal Specifications

The chapter is organized as follows: Section 4.1 presents background, Section 4.2 describes contains methodology and section 4.3 presents experimental setup. Section 4.4 contains algorithm and section 4.5 contains case study. Results and threats to validity are presented in section 4.6 and section 4.7 respectively.

The proposed technique involves mobile agent-based technology using model based on UML class diagram, activity diagram, use cases and formal specifications based on Object-Z and OCL for envisaging agent-based regression test case generation. This approach is based on the regression test case generation using mobile agents by considering modifications in semantics of operations using UML class, activity diagrams, use cases and formal specifications based on Object-Z and OCL.

In UML class diagram, we get only static aspects of specifications, whereas use cases and acti+vity diagram provides behavioral information. Thus using UML class diagram, activity diagram, use cases with formal specifications may achieve better coverage in terms of important aspects of specifications and semantic changes. Mobile agents are used to gather information in the form of XML files and XMI files from different machines in the distributed environment.

4.1 Background

Researchers have suggested many techniques for model based test case generation using activity diagram and use cases. Hettab, *et al.* [154] converted UML activity diagram into graph grammars to capture all the relevant features for test case generation. Wang *et al.* [155] proposed use case modeling to generate executable test cases from use case specifications by applying Natural Language Processing (NLP). Shirole *et al.* [157] transformed activity diagram into Extended Control Flow Graphs (ECFG). Evolution algorithm with concurrent path

coverage is used to generate paths in the activity diagram. We identified existing studies in UML activity diagram and use cases in regression testing and pointed them in Table 4.1.

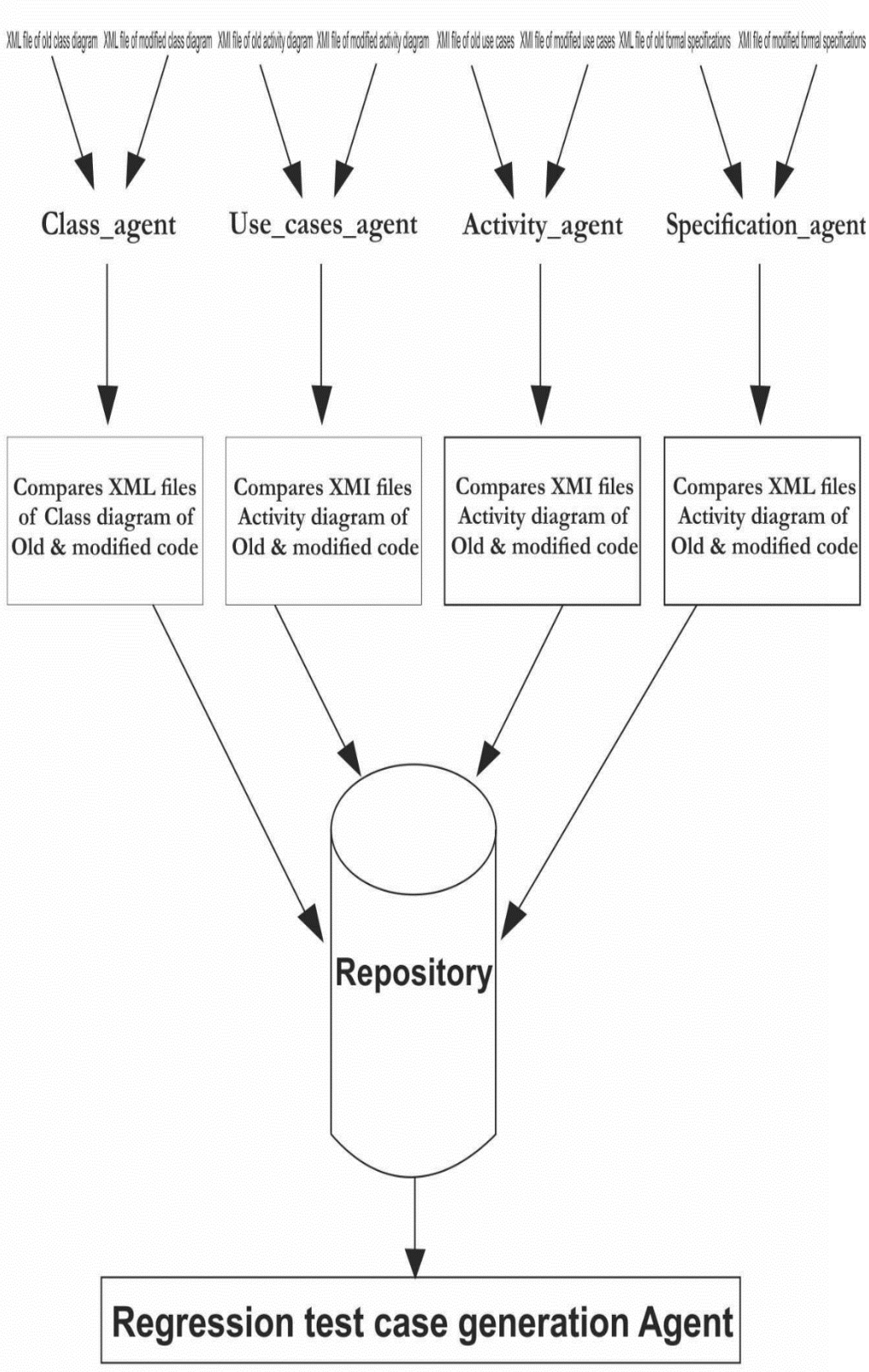
Table 4.1: Regression testing techniques using UML Activity diagram and use cases

S. No	Technique Used	Research Gap
1.	Dahiya *et al.* [77] used UML class, sequence and activity diagrams for regression test case generation	Proposed approach lacks in identifying changes at distributed environment
2.	Sunitha [156] incorporated OCL into activity diagram for test case generation	Unable to identify complete behavior and static changes
3.	Vincent *et al.* [33] represented use cases as state charts for test case generation	Lacks formal specifications and needs validation
4.	Ye *et al.* [158] used classification to select retestable and feedback-directed test cases using activity diagram	Unable to identify static changes at class level and use cases
5.	Chen [26] matched Java program traces with behaviour activity diagram to identify changes	Unable to identify static changes
6.	Sapna and Mohanty [159] converted UML activity diagrams into tree structure to prioritize the scenarios by assigning weights to nodes and edges	Lacks in in-depth code coverage and unable to identify static changes
7.	Mingsong *et al.* [144] generated test cases from design specifications using UML activity diagrams	Lacks in in-depth code coverage and case study to validate the approach

8.	Gorthi *et al.* [25] changed behavior and version number in activity diagram to generate test cases	Prototype model with retail system needs more case studies for validation
9.	Kim [160] used directed graph from UML activity diagram to extract test scenarios	Needs to generalize with more case studies and lacks in-depth code coverage
10.	Xu *et al.* [98] used UML activity diagram through scenarios based on Fork Join approach to directly generate test adaptive agents	Unable to identify static changes at class level
11.	Li and Lam [161] used agents to generate test threads from artefact using activity diagram	Unable to identify static changes

4.2. Methodology

We have used five agents as shown in Fig. 4.1 namely class comparison, activity comparison, use case comparison, specification comparison agent and regression test case generation agent for performing different comparison task and regression test case generation. Mobile agents move from host machine to different machines in the distributed environment to perform comparison tasks assigned to them. Class comparison agent carries XML file of UML class diagram of old code and compares it with XML file of modified code at machine 1. Similarly activity comparison agent compares XMI file of old and modified activity diagram at machine 2. Use case agent compares XMI file of Use cases and specifications comparison agent compares XML file of Object-Z and OCL of new and old versions. All agents perform comparison tasks simultaneously at different machines thus resulting in comparatively less time for test case generation. These agents submit the change impact analysis to repository at host machine. Regression test case generation agent uses changes from repository for regression test case generation.



4.1: Agent-based platform for regression testing

4.3 Experiment setup

We used Eclipse, Object Aid [152] plug-in to draw UML class diagram and converted it into corresponding XML code. We used ArgoUML[162] tool to draw UML activity diagrams and use cases and converted them into corresponding XMI code. Java Agent Development Environment (JADE) is used for designing five agents: class_agent, use_case_agent, activity_agent, specification_agent and RTCG_agent. These agents proceed in the distributed environment to compare old UML models with the modified UML model to identify changes in the software. Class_agent parses XML files of old and modified UML class diagram to get static contents of the code and compares them. Use_case_agent extracts and compares use cases from old and modified use case diagram. Activity_agent parses XMI files of old and modified activity diagram and compares them. Specification agent parses XML file containing Object_Z and OCL. These agents submit changes to repository. RTCG_agent collects these changes from repository and categorize test cases on the basis of obsolete, retestable and reusable test cases. We used following technique to extract static information from the XML file of UML class diagram.

4.4 Algorithms

Different agents are designed in JADE to identify changes impact analyses and generate regression test cases. The algorithms for class comparison agent, activity comparison agent, use case comparison agent and specifications comparison agent are given below.

Definition: Let CL and CL‘ be the set of classes in old and new XML file of UML class diagram, then ∀ (cl ∈ CL) and ∀ (cl’ ∈ CL’), set of reusable artefacts ←(cl ∩ cl’), set of obsolete artefacts ←(cl U cl’)- cl’ and set of retestable artefacts ←(cl U cl’)-cl. These steps can be repeated to categorize retestable, reusable and obsolete test cases on the basis of set of class dependencies, attributes and operations.

1. Class Agent

Input: XMI file extracted from old and new class diagram

I. Carries XML file of old UML class diagram from host machine to machine 2.

II. Parses XML files of old and modified UML class diagrams to extract classes and their corresponding associations, dependencies, attributes and operations.

III. These UML entities are compared to find a set of retestable, reusable or obsolete test cases. The detailed steps for comparison are given below.

IV. The set of changed classes are identified based on the comparison between corresponding classes in CL and CL':

V. If there is a change in CL' corresponding to CL then the class belonging to CS' is added to the set of retestable classes i.e.Tretest_c

VI. If the corresponding class in CL and CL' are same then they are added to the set of reusable classes i.e.Treuse_c

VII. If a class belonging to CL is not present in CL' then add that class to a set of obsolete class i.e.Tobs_c

VIII. Repeat step IV for comparing dependencies, attributes, associations and operations.

2. Activity Agent

Input: XMI file extracted from old and new activity diagram

I. Carries XMI file of old activity diagram from host machine to machine

II. Parse XMI files of old code and modified code to generate trees T(N,E) and T'(N',E') where T and T' are trees representing old and modified activity diagrams respectively, N and N' represents nodes of the corresponding tree and E and E' be the edges

III. $\forall$ T(N,E) $\exists$(n $\in$ N) and $\forall$ T'(N,E) $\exists$(n' $\in$ N')

Begin

Start traversing each node for n$\in$N and n'$\in$N'

Mark each node visited

If (n==n') // Check if corresponding nodes of the tree T and T' are same

Tresua $\leftarrow$ $\forall$ (n $\in$ N)

// If corresponding nodes are same then store those nodes in set of reusable test cases Tresu_ac

else

Tretest_ac ← ∀ (n' ∈ N)

// If corresponding nodes are not same then store those nodes in set of retestable test cases Tretest_ac

End

3. Use Case Agent

Input: XMI file extracted from old and new use case diagram

I. Carries XMI file of old use case diagram from host machine to machine 3.
II. Parses XMI files of old and modified use case diagram to extract classes and their attributes to generate nodes and edges of the trees representing use case scenarios rom trees T(N,E) and T'(N',E') where T and T' are trees representing old and modified use case scenarios respectively, N and N' represents nodes of the corresponding tree and E and E' be the edges
III. These UML entities are compared to find a set of retestable, reusable or obsolete test cases.

Trest_use ← ∀ (n ∈ N)

// If corresponding nodes are same then store those nodes in set of reusable test cases Tretest_use

else

Tretest_use ← ∀ (n' ∈ N)

// If corresponding nodes are not same then store those nodes in set of retestable test cases Tretest_use

End

4. Specification Agent

Input: XMI file extracted from old and new activity diagram

I. Carries XML file of old formal specifications from host machine to machine 4
II. Parses XML files of old and modified formal specification codes to compare Object-Z and OCL codes to extract classes, operations and their attributes and constraints. These entities are compared to find a set of retestable, reusable or obsolete test cases.
III. The set of changed classes are identified based on the comparison between corresponding classes in CZ and CZ':
IV. If there is a change in CZ' corresponding to CZ then the class belonging to CZ' is added to the set of retestable classes i.e.Tretest_z
V. If the corresponding classes in CZ and CZ' are same then they are added to the set of reusable classes i.e.Treuse_z
VI. If a class belonging to CZ is not present in CZ' then add that class to a set of obsolete class i.e.Tobs_z
VII. Repeat step 3 for comparing operations, their attributes and constraints.
VIII. Submit results to impact analysis agent at machine 1.

4.5 Case study

We analyzed ATM and Library subject systems for the validation of our study. We downloaded these subject systems from Github repositories. We are presenting our technique with a case study of ATM system.

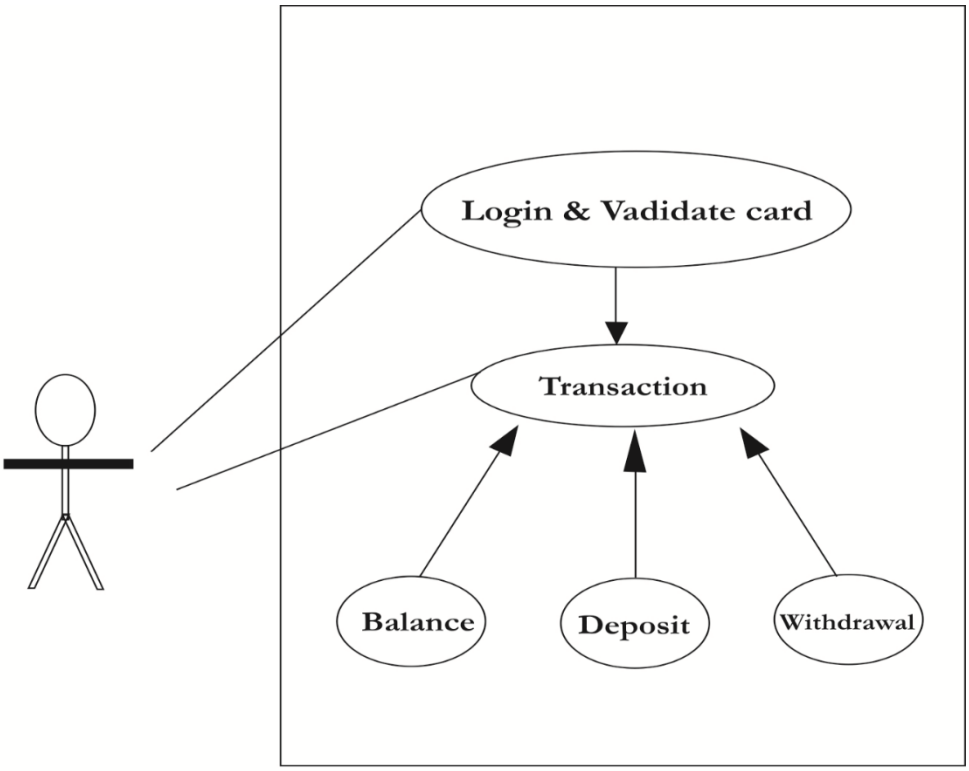


Fig. 4.2: Use cases of ATM system

In the ATM system, there is GUI based transaction mechanism as shown in use case in Fig. 4.2 and UML class diagram in Fig 4.3. User inserts his card in ATM and validates it using account number and pin. Upon successful login, user can perform desired transactions like withdrawal, balance enquiry and deposits. User can deposit cash in his account by inserting cash at cash dispenser. User gets the message upon successful completion of its transaction. The available balance can be checked by using balance enquiry option. The process of cash withdrawal is shown in activity diagram at Fig 4.4a. User is restricted to withdraw maximum amount of Rs.10,000 in one transaction. In case the amount is greater than 10,000, user is prompted to re-enter the amount less than or equal to 10000 as shown in Fig 4.4a. On later stages, the system is enhanced with new requirement which requires the user to enter the amount in the multiple of 100 or 500 for withdrawal or deposit. This leads to the changes in the class level, use cases as well as activity diagram as shown in Fig 4.4b. Apart from it there is also change in authentication level in the system. If the user enters wrong pin consecutively three times. The system confiscates the card and displays on the screen that wrong pin has been entered

incorrectly three times. Upon this system makes log entry of the event in the system that the customer has entered wrong password consecutively in three attempts. Thus the nodes with the changes are need to be retested. This lead to the changes in the user case diagram and behavior of the system.

These class, activity and use cases diagrams are compared with old and modified code to detect change in semantics in terms of paths, constraints, operations, and control flows.

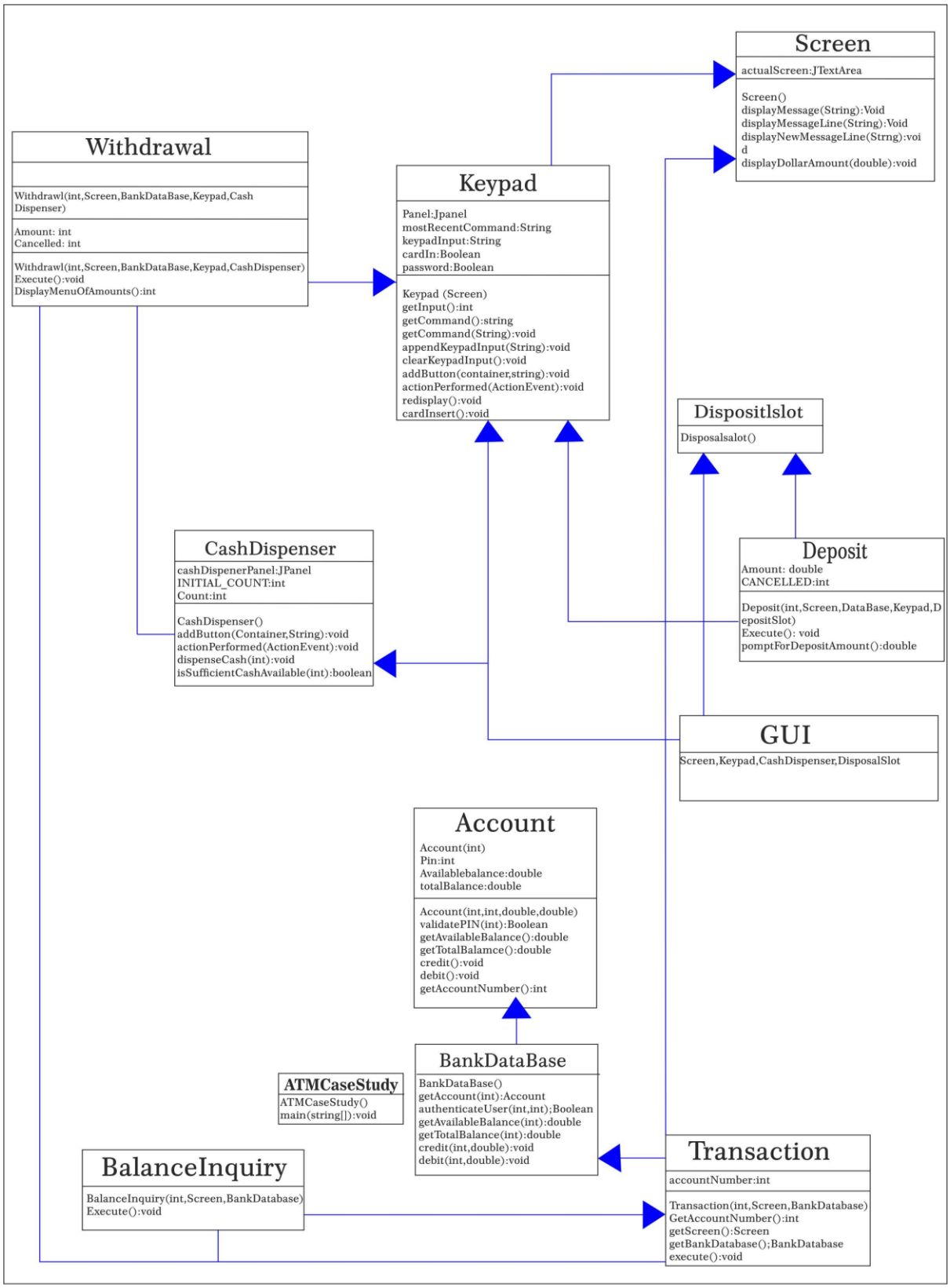


Fig 4.3: UML class diagram of ATM System

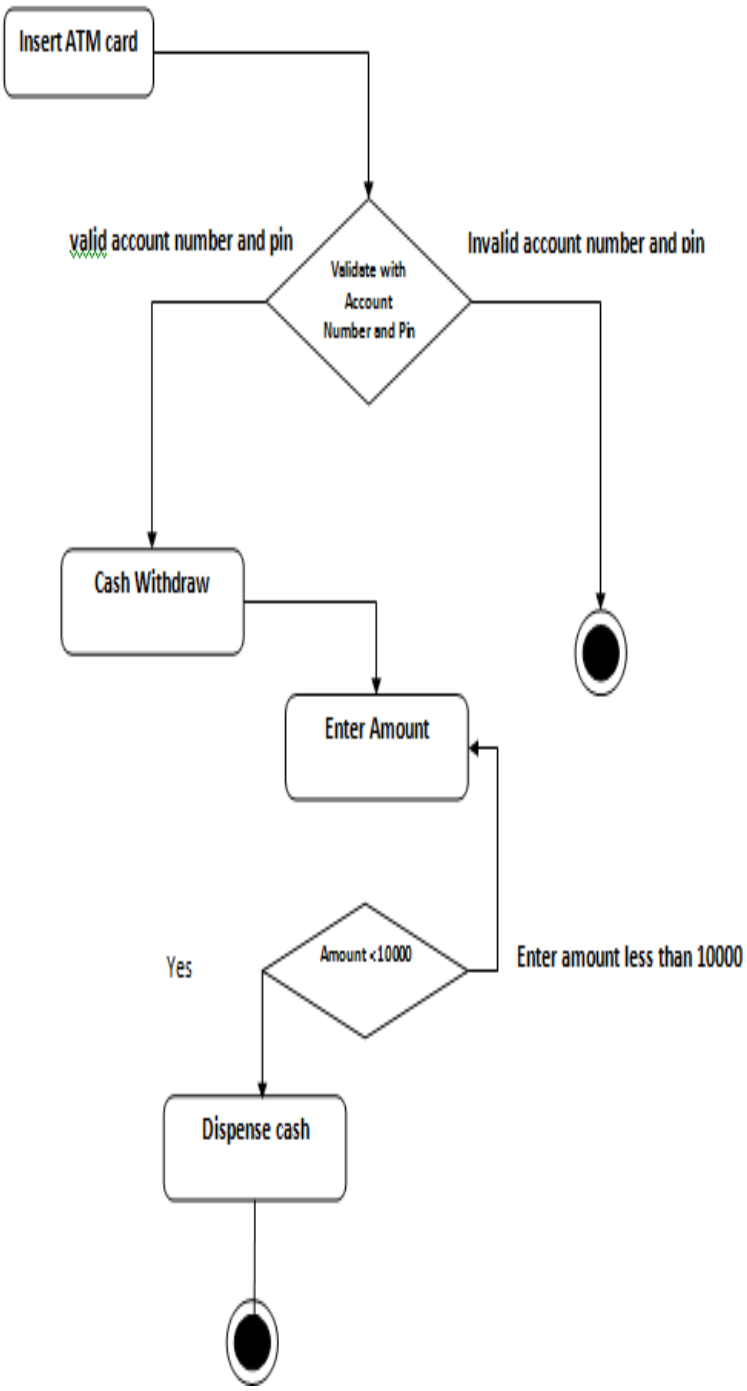


Fig 4.4a: Activity diagram of old ATM System

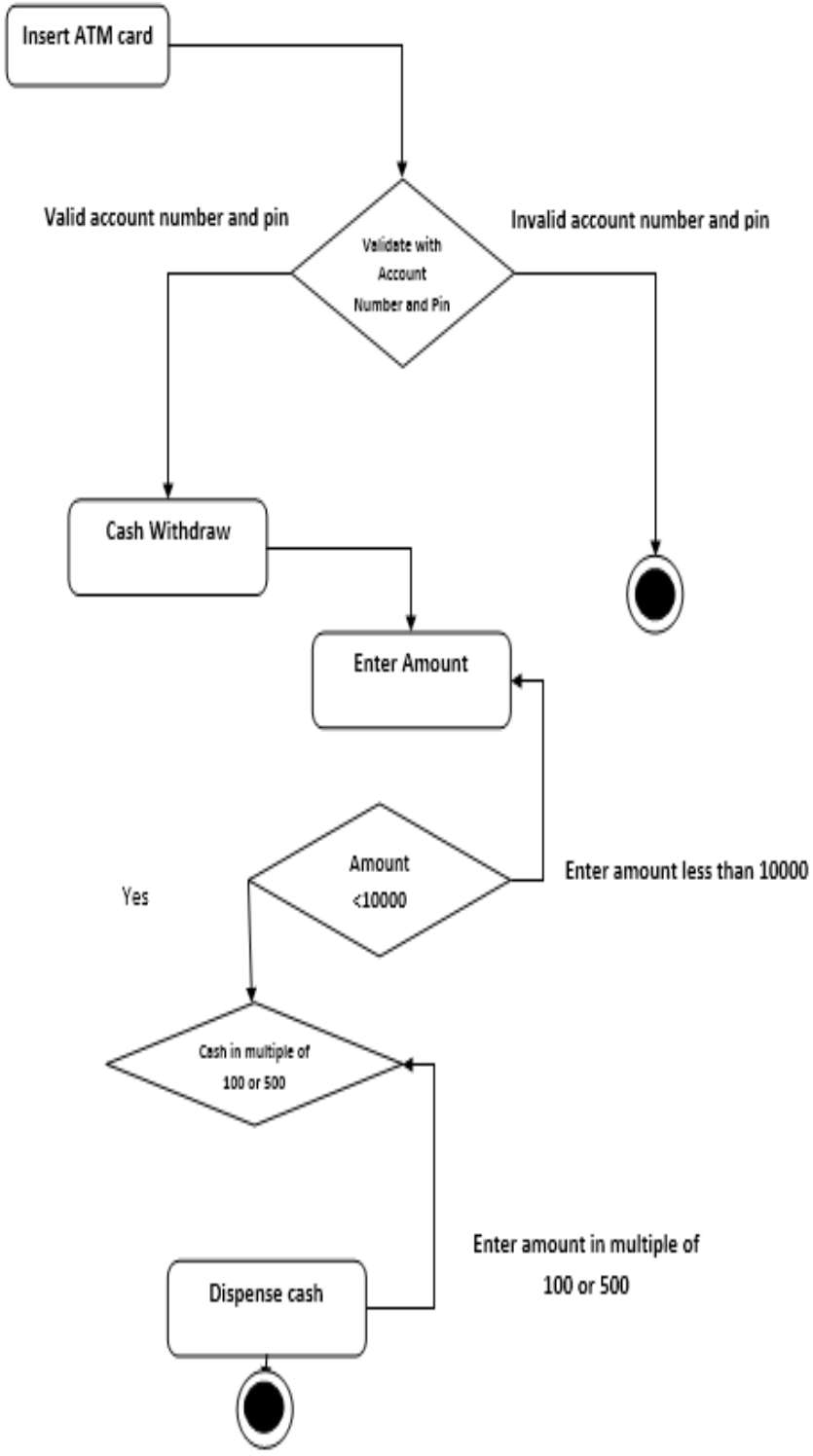


Fig 4.4b: Activity diagram of modified ATM System

We identified test sequences corresponding to ATM system and provided them in table 4.2.

Table 4.2: Test sequences of ATM System

Test Sequence No	**Test Path**
TS1	ATM->userAuthenticated->currentAccountNumber->Balance_Inquiry-> withdrawal ->deposit
TS2	ATM->userAuthenticated->currentAccountNumber->withdrawal-> amount -> withdrawal ()->execute()
TS3	ATM->userAuthenticated->currentAccountNumber->deposit->amount-> deposit( ) ->execute
TS4	ATM->userAuthenticated->currentAccountNumber->BankDatabase-> getAccount->getAvailableBalance->getTotalBalance->credit,debit -> Account-> validatepin() ->getAvailableBalance()->getTotalBalance( )-> getAccountNumber()

Implementation of ATM test sequence in OCL

Test Sequence1:

Context: ATM: userAuthenticated: Boolean

Inv: currentAccount number= AccountNumber

Else

Current Account number=0

Test Sequence2:

Context: Withdrawal: :Withdrawal (amount: int)

Pre : pinaccepted

Inv: amount<20000

Post : Totalbalance =totalbalance@pre-amount

Test Sequence3:

Context: Deposit : : deposit (amount: int)

Pre: pinaccepted

Inv: amount>500500

Post: Totalbalance= Totalbalance@pre+amount

Test Sequence 4:

Context: Account:: validatepin(int): boolean

Inv: if (userpin=pin)

Return true;

Else

Return false;

Context: account::credit (double): void

Pre: amount>0

Post:Totalbalance=totalbalance@pre+amount

Context: Account::debit (double): void

Inv: Amount>0

Availablebalance=avialablebalance-amount and totalbalance=totalbalance-amount

4.6. Results

We evaluated and compared our results against Library system and ATM with Dahiya [77] and Mansour [149] in terms of the number of test cases generated. We have further enhanced the subject systems with Object-Z and OCL specifications to improve semantic analysis. Encouraging response is observed in comparative analysis shown in Table 4.3.

Table 4.3: Comparative analysis on number of test suites generated

Case Studies ➡		**Library System**		**ATM System**	
Types of Test Suite ⬇	Results from [149]	Results from [77]	Our Results	Results from [77]	Our Results
Types of Sources ➡	UML class, Sequence and Activity diagrams	UML class, Sequence and Activity diagrams	UML class diagram, Activity diagram, Use Cases, Object-Z and OCL	UML class, Sequence and Activity diagrams	UML class, Activity diagrams, Use Cases, Object-Z and OCL
Retestable	7	6	9	3	7
Reusable	76	107	109	158	158
Obsolete	0	2	0	0	0

We observe that use of UML class diagram, use cases and activity diagram with formal specifications Object- Z and OCL has resulted in better coverage of test cases and has resulted in identifying changes at syntactic and semantic level. We have used agents to identify changes and make comparative analysis at different machines simultaneously. Also use of agents to simultaneously compare the code has resulted in less time required for regression test case generation as compared to other studies using localized environment for the same.

4.7. Threats to Validity

We used UML models and formal specifications to identify syntactic changes and behavioral changes to moderate the threats to validity. We have reduced the time required for test case generation by distributing testing tasks among agents. Currently lack of standardized tools for drawing UML diagrams embedded with formal specifications is the external threat to validity.

Chapter – 5

Mobile Agent-based Regression test Case Generation using Machine Learning

The chapter is organized as follows: Section 5.1 contains introduction, Section 5.2 describes motivation behind the research and section 5.3 presents background. Section 5.4 contains methodology and section 5.5 describes experimental setup, 5.6 contains case study. Results and threats to validity are presented in Section 5.7 and Section 5.8 respectively.

5.1 Introduction

Over the past decade, with the advent of big data and distributed systems; scalability and complexity of software has grown multiple times. Due to increased complexity, identifying changes in such software for regression test case generation with human effort has become very difficult. There is need to explore automated test case generation techniques to reduce the cost associated with regression testing and improve the performance of software testing.

Regression test case generation through change identification using models and formal specifications has certain limitations as developers do not provide UML models and formal specification codes to the testing teams. Due to these issues, testing techniques based on these approaches faces lot of challenges to identify changes in the code. Model based techniques require code to be transformed to UML models. These models are converted to XML or XMI with the help of some tools and further these XML and XMI files need to be parsed for requisite artefacts. Also, the process of transforming code to its formal specifications is very difficult and time consuming as it requires manual effort. Thus to deal with the exhaustive manual effort in identifying retestable, reusable and obsolete test cases; machine learning based decision making techniques can better assist in the process of test case generation. This will reduce the overhead associated with regression testing.

The major focus of machine learning is to automatically learn, recognize the complex and useful patterns in data. These patterns present useful information about the data and serve as basis for decisions making. Many researchers have advocated the use of machine learning techniques in the field of software testing. In present work, we recommend the use of agents and machine learning for test case generation in regression testing in order to reduce the cost and time associated with testing of complex and diverse distributed systems.

5.2. Motivation

In our systematic literature review of 123 research papers on test case generation and agent-based test case generation in regression testing. We observed that most of the existing studies in test case generation are focused on model base, functional, structural and formal specifications for desired change analysis. In our earlier approach, we used agents to extract information from models based on UML class, sequence and activity diagrams and formal specification languages based on Object-z and OCL to identify changes in the code. But models and formal specifications have threats to validity due to unavailability of UML models and tools to transform UML models into formal specifications.

This gives us opportunity to explore combined use of machine learning and agents for their viability in software testing. Thus promising developments in regression test case generation and agent-based technology has called for the need to incorporate agents with machine learning abilities for testing large complex software with reduced human effort and make it cost effective.

In this study the focus is on using agent technology in the field of machine learning with a particular interest on applying agent-based solutions to regression test case generation.

5.3. Background

Agent-based technology is emerging field to support testing of big data, convoluted and heterogeneous distributed systems. Agent technologies integrated with machine learning techniques may develop additional characteristics like learning capabilities, interoperability, scalability and parallel computation. With the help of agents and machine learning approaches, parallelization can be applied to test case generation to speed up the testing process.

Many machine learning based techniques have evolved over the time including supervised learning, unsupervised learning, semi supervised learning, analytical learning and reinforcement learning. We conducted a review of existing studies using machine learning in software testing. Brinad [164] used partially automated Category Partition approach for specification reconstruction and test suite improvement. Test cases are abstracted as category and choice combinations. Machine learning is used to learn the inputs and conditions for decision making. Gove [165] used support vector machine with Matlab to augment automatic model-based test case generation for graphical user interfaces (GUIs). Silva [166] used machine learning to measure the execution effort of machine learning. Support Vector machine and Multilayer Perception (MLP) based on Artificial Neural Networks (ANN) are compared on two databases based on Java. Malhotra [167] compared machine learning techniques and search based techniques on the basis of sensitivity, accuracy, precision f-measure and g-measure on three open source softwares for class change prediction. Pang [168] compared K-means clustering to measure the hamming distances metric to identify effective test cases and measure similarity between the test cases for enhancing the regression testing. Proposed technique is evaluated on Java based programs based on accuracy and precision. Noorian [169] proposed classification framework to systematically review research work in the machine learning and software testing. Malhotra [177] conducted empirical study of object oriented metrics. They investigated 22 metrics proposed by various researchers to determine which metrics is useful in which area. Lincke *et al.* [178] presented a study to compare software metrics tools. They presented a comparison analysis of tools to recommend suitable software metrics tools for identifying the potential attributes required for regression test cases generation. We explored existing studies on machine learning based testing and presented them in Table 5.1.

Table 5.1: Test case generation techniques using machine learning

S. No	Reference	Technique Used	Drawbacks
1.	Pang [168]	Used K means clustering for enhancing Regression testing	Used methods, blocks, and statements. Needs more attributes and experiments to validate the results.

2.	Zhang [170]	Proposed a framework based on Genetic algorithm machine learning method on value based software engineering. Divided Test data generation process of a software system into a sequence of test cycles	Lacks empirical analysis with other studies.
3.	Lachmann *et al*. [171]	Proposed ranking of test cases based on a given set of training data through supervised machine learning technique Support Vecor Machine (SVM) RANK. It utilizes high-level artefacts for black-box testing.	Needs manual inspection, unable to detect changes in semantics
4.	Gondra [172]	Performed comparative experimental study on effectiveness of Artificial Neutral Network and Support Vector Machine in predicting fault-proneness.	Needs more algorithms and data sets along with attributes to validate this approach.
5	Pierre-Christophe *et al.* [173]	Proposed use of learning model to infer Deterministic Finite Automaton (DFA) machine learning algorithm from symbolic animation of model based on generalized substitutions, such as B	The problem of scalability of the proposed algorithm is not discussed. Compared two techniques for testing, but testing efficiency remains an issue
6.	Zhang *et al.*[174]	Proposed classification and regression based machine learning fuzzy classification technique KNN (k-Nearest Neighbour) to identify truly coincidental correct test cases for single fault version programs.	Unable to identify static changes at class level
5.	Chen *et al.* [175]	Proposed semi-supervised K- means (SSKM) to improve cluster test selection.	SSKM is not suitable for the application with a small number of features. Clustering results depend on high quality constraints.
7.	Cotroneo *et al.* [176]	Presented a learning-based method for online testing techniques combination. The method aims at maximizing the number of faults	Relies on certain abstractions levels and lacks in-depth code coverage

		detected by dynamically selecting test cases.	
8.	Malhotra *et al.* [177]	Compared machine learning algorithms with logistic regression technique for prediction of change prone classes.	Lacks in test case prediction and needs more attributes to identify change proneness.

We find that literature is available on machine learning based software testing, yet the existing tools based on machine learning algorithms and techniques cannot be identified for integration of machine learning with distributed agent-based technology. Also the benefits of machine learning still needs to be explored in the field of software testing as potential attributes needed for identification of test cases.

5.4. Proposed Methodology

Our work is based on machine learning approach to generate test cases based on change impact analysis and object oriented metrics. We used agent-based machine learning technique for identifying changes in the software for regression test case generation. The proposed agents model consist of four agents viz. CSV agent, comparison agent, learning agent and regression test case generation agent. CSV agents carry CSV files of the software to be tested from client machines in the distributed environment and submit them to comparison agent. Comparison agent perform comparative analysis of the CSVs of the old and modified code and submit final CSV to repository. Learning agents carry these CSV files to the server and perform desired machine learning analysis for regression test case generation as shown in Fig 5.1. For designing our agent framework, we used open source Java Agent Development Environment (JADE) framework. In first of its kind approach, we have integrated JADE based agents with machine learning WEKA tool [181] for designing distributed, machine learning approach to identify changes in the code for test case generation. Table 5.2 contains details of all the testing tasks performed by these agents.

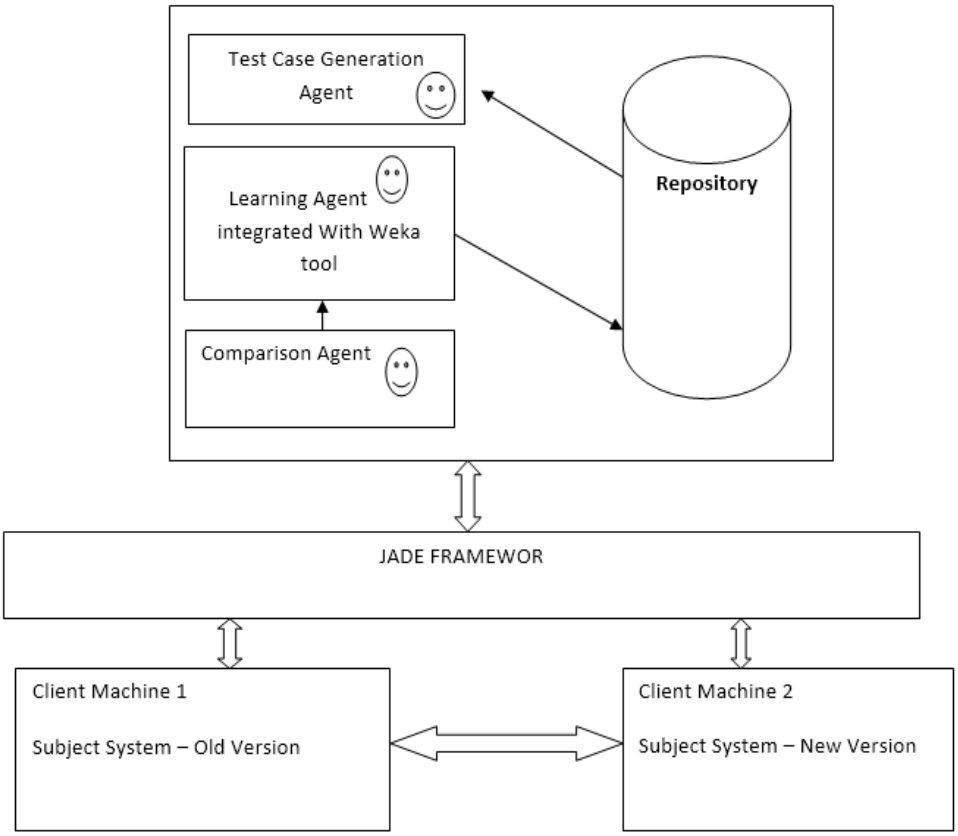


Fig 5.1: Multi agent-based machine learning framework

Table 5.2 Agents with different testing tasks for regression test case generation

Agent Name	**Description**
CSV Agent	Initialization through setup() method Carries CSV file of code from client machines to comparator agent Lifetime : Till it submits CSV file to comparison agent Incoming message to start the initialization message Outgoing messages to comparison agent

CSV Comparator agent	Initialization through setup() method Compares CSV files of old and modified code versions Lifetime : Till it submits results to learning agent Incoming message to start the initialization message Outgoing messages to learning agent
Learning agent	Initialization through setup() method integrates with WEKA tool Carries CSV file of code and submit it to WEKA tool Lifetime : Till it submits results to repository Incoming message to start the initialization message Outgoing messages to Repository
Regression test case generation agent	Generates test cases

5.5 Experimental setup

In the present work, we used classes, their associations, dependencies, methods and attributes for semantic analysis. We extracted classes, dependencies, associations, methods, their return type and access specifiers, entities and attributes from the old and modified code and stored them in CSV file. These metrics cover percentage lack of cohesion, max inheritance tree, count of base classes, count of coupled classes, count of derived classes and count of all methods etc. These metrics were extracted using Understand tool [180] to measure different attributes of object oriented software. We also used set of object oriented metrics based on CK metrics suite [179] to cover various attributes of object oriented software. We used CKJM tool [179] to extract 19 metrics to measure different attributes of object oriented software. This metrics set

measure cyclomatic complexity, inheritance, counts of classes, coupling, SLOC, cohesion etc. related to different attributes of object oriented software. Details of the metrics calculated for each class using *CKJM* tool is given below in Table 5.3.

Table 5.3: List of Object Oriented Metrics supported by CKJM tool [179]

Metric	**Details**
WMC	Weighted methods per class is equal to the number of the methods in the class. It assigns default value 1 to each method
DIT	The depth of inheritance tree (DIT) metric means inheritance levels of a class. Its minimum value is 1
NOC	Number of Children means the number of immediate descendants of the class
CBO	Coupling between object classes means number of classes coupled to a given class
RFC	Response for a Class means number of different methods that can be invoked for an object of class
LCOM	A class's lack of cohesion in methods (LCOM) counts the sets of methods in a class that are not related through the sharing of some of the class's fields.
Ca	Afferent couplings means how many other classes use the specific class.
Ce	A class's efferent couplings is a measure of how many other classes is used by the specific class. Coupling has the same definition in context of Ce as that used for calculating CBO
NPM	Number of Public Methods means number of public methods in a class
LCOM3	Lack of cohesion in methods. LCOM3 varies between 0 and 2. m - number of procedures (methods) in class

	a - number of variables (attributes in class μ(A) - number of methods that access a variable (attribute) $LCOM3 = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1 - m}$
LOC	Lines of Code is the sum of number of fields, number of methods and number of instructions in every method of given class
DAM	Data Access Metric means the ratio of the number of private (protected) attributes to the total number of attributes declared in the class. A high value for DAM is desired.
MOA	Measure of Aggregation means count of the number of data declarations (class fields) whose types are user defined classes.
MFA	Cohesion Among Methods of Class computes the relatedness among methods of a class based upon the parameter list of the methods. A metric value close to 1.0 is preferred.
CAM	Cohesion among Methods of Class computes the relatedness among methods of a class based upon the parameter list of the methods.
IC	Inheritance Coupling means the number of parent classes to which a given class is coupled in following manner □ One of its inherited methods uses a variable (or data member) that is defined in a new/redefined method. □ One of its inherited methods calls a redefined method. □ One of its inherited methods is called by a redefined method and uses a parameter that is defined in the redefined method.

CBM	Coupling Between Methods means the total number of new/redefined methods to which all the inherited methods are coupled.
AMC	Average Method Complexity means average method size for each class. Size of a method is equal to the number of java binary codes in the method.
CC	CC - The McCabe's Cyclomatic Complexity It is equal to number of different paths in a method (function) plus one. The Cyclomatic Complexity is defined as: CC = E - N + P where E - the number of edges of the graph N - the number of nodes of the graph P - the number of connected components

We merged all these object oriented metrics along with classes, their associations, methods, attributes and files containing code into single CSV file for in-depth code coverage. The next step was to decide the decision variable for change identification. For this, we compared corresponding attributes in the CSV files of old and modified code using python script. In a first of its kind study, we have integrated JADE agents with open source WEKA tool for developing machine learning based agents.

We extracted the numerical difference between each attribute of CSV files containing old and modified code. We marked YES/NO in decision variable on the basis of difference between these attributes. We marked 'NO' in the decision variable in case the difference of all the attributes corresponding to any artefact contained '0'. We marked 'YES' in the decision variable in case the difference of any attribute corresponding to an artefact has non-zero value.

Thus change identification in a code can be identified in terms of object oriented metrics, changes in class, methods, attributes, class inheritance etc.

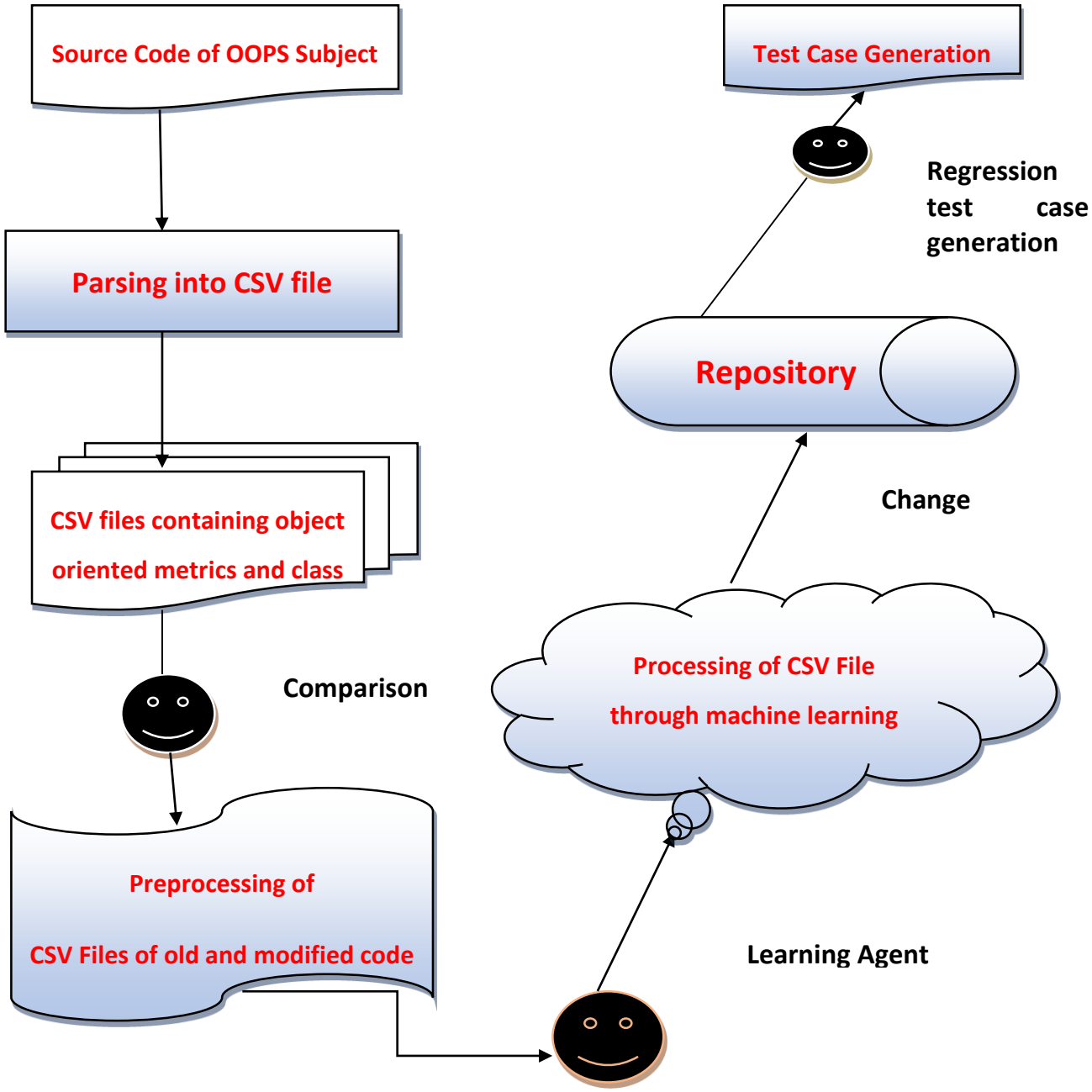


Fig. 5.2: Stepwise methodologies for regression test case generation

5.6 Case Study

We used subject systems based on JAVA from the software repositories to validate our study. The complete details of these subject systems, their versions classes, code lines, declarative statements, files and functions are given in Table 5.4. We extracted all the OO metrics of these subject system along with their classes, methods, attributes and merged them in single CSV files using Understand tool [180]. These were compared by comparator agent.

Table 5.4: Details of Subject systems

Subject System	Classes	Code Lines	Declarative statements	Executable Statements	Files	Functions
apache-tomcat						
Version 6.0.47	1812	180260	51234	79808	1204	16997
Version 6.0.48	1815	180382	51282	79833	1205	17007
apache-maven						
Version 3.2.1	1017	73510	22345	21235	886	6733
Version 3.2.2	1022	74085	22478	21429	891	6768
Struts						
Version 2.5.1	2677	151024	52293	62451	1919	15854
Version 2.5.2	2676	151009	52285	62452	1919	15854
Jmeter						
Version 3.0	1563	120762	41806	47855	1179	11247
Version 3.1	1586	122063	42230	48291	1195	11357

Jtopas						
Version 0.7	90	11077	2762	4800	75	978
Version 0.8	80	9037	2313	3681	64	787
Library System						
Version - 1.1	28	2223	416	1297	14	70
Version - 1.2	28	2234	421	1306	14	70
ATM						
Version 1.1	13	652	166	232	14	59
Version 1.2	14	662	171	237	14	59

5.7 Results

Table 5.5 contains results of 10 fold cross validations of our model developed using metrics extracted using Understand tool [180]. We have used larger subject systems and more metrics as compare to Malhotra [167] and Pang [168] to validate our study. We compared the results by using various machine learning classifiers. The various parameters used for evaluation are sensitivity, accuracy, and precision. We observe that classifiers such as Random Forest and Random Tree are giving best results on all the subject systems. We also observe that cyclomatic complexity and count of statements are good indicator for calculating impact analysis of object oriented software for test case generation by using metrics extracted through Understand tool. Thus, these metrics may highly affect the change identification in a software and can be used for as a benchmarks for change identification.

Table 5.5: Results on 10 Fold cross validation

Subject System	**TP Rate %**	**FP RATE %**	**Precision %**	**Recall F %**	**F-Measure %**	**MCC %**	**ROC Area %**	**PRC Area %**	**Correctly Classified Instances %**	**Incorrectly Classified Instances %**
Jtopas										
J48	0.984	0.016	0.985	0.984	0.984	0.969	0.980	0.973	98.4167	1.5833
RandomForest	**0.998**	**0.002**	**0.998**	**0.998**	**0.998**	**0.996**	**0.997**	**0.996**	**99.7917**	**0.2083**
RandomTree	**0.994**	**0.000**	**1.000**	**0.994**	**0.997**	**0.994**	**0.996**	**0.997**	**99.7083**	**0.2917**
REPTree	0.984	0.016	0.985	0.984	0.984	0.969	0.980	0.973	98.4167	1.5833
LMT	0.990	0.010	0.990	0.990	0.990	0.980	0.990	0.985	99	1
Decision Stump	0.730	0.274	0.824	0.730	0.708	0.545	0.715	0.707	73	27
HoeffdingTree	0.953	0.048	0.957	0.953	0.952	0.909	0.975	0.956	95.25	4.75
SMO	0.601	0.405	0.777	0.601	0.523	0.330	0.598	0.576	60.0833	39.9167
Logistic	0.749	0.255	0.832	0.749	0.731	0.574	0.583	0.643	74.875	25.125
SGD	0.716	0.288	0.818	0.716	0.690	0.523	0.714	0.675	71.5833	28.4167

SimpleLogistic	0.756	0.248	0.836	0.756	0.740	0.585	0.596	0.646	75.5833	24.4167
Jmeter										
REPTree	0.997	0.027	0.997	0.997	0.997	.983	0.996	0.998	99.7125	0.2875
RandomTree	**0.999**	**0.014**	**0.999**	**0.999**	**0.999**	**0.992**	**0.993**	**0.997**	**99.8563**	**0.1437**
RandomForest	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**	**0.993**	**1.000**	**1.000**	**99.8836**	**0.1164**
LMT	0.997	0.026	0.997	0.997	0.997	0.984	0.990	0.997	99.7194	0.2806
J48	0.993	0.068	0.993	0.993	0.993	0.958	0.961	0.987	99.2813	99.2813
DecisionStump	0.932	0.636	0.937	0.932	0.915	0.525	0.639	0.876	93.2375	0.2875
HoeffdingTree	0.993	0.636	0.062	0.993	0.993	0.962	0.992	0.997	99.3429	0.6571
SMO	0.918	0.768	0.925	0.918	0.371	0.371	0.575	0.851	91.8275	8.1725
Logistic	0.927	0.683	0.932	0.927	0.907	0.474	0.749	0.895	92.7242	7.2758
SGD	0.926	0.700	0.931	0.926	0.903	0.457	0.613	0.864	92.553	7.447
SimpleLogistic	0.921	0.740	0.927	0.921	0.895	0.407	0.776	0.896	92.115	7.885
Maven										
REPTree	0.997	0.002	0.997	0.997	0.997	0.994	1.000	0.999	99.7439 %	0.2561 %
RandomTree	**0.999**	**0.000**	**0.999**	**0.999**	**0.999**	**0.999**	**1.000**	**0.999**	**99.9405**	**0.0595**

RandomForest	**0.999**	**0.000**	**0.999**	**0.999**	**0.999**	**0.999**	**1.000**	**1.000**	**99.9379**	**0.0621**
LMT	0.991	0.005	0.991	0.991	0.991	0.981	0.998	0.996	99.1256	0.8744
J48	0.995	0.003	0.995	0.995	0.995	0.990	0.998	0.997	99.524	0.476
DecisionStump	0.853	0.243	0.865	0.853	0.845	0.680	0.801	0.801	85.252	14.748
HoeffdingTree	0.998	0.002	0.998	0.998	0.998	0.995	1.000	0.999	99.7801	0.2199
SMO	0.854	0.243	0.866	0.854	0.846	0.683	0.805	0.788	85.3632	14.6368
Logistic	0.855	0.242	0.868	0.855	0.847	0.686	0.806	0.840	85.4874	14.5126
SGD	0.854	0.243	0.867	0.854	0.846	0.684	0.806	0.788	85.4046	14.5954
SimpleLogistic	0.854	0.243	0.867	0.854	0.846	0.683	0.807	0.815	85.3787	14.6213
Struts										
REPTree	0.999	0.062	0.999	0.999	0.999	0.968	0.976	0.999	99.925	0.075
RandomTree	1.000	0.023	1.000	1.000	1.000	0.988	0.988	0.999	99.9719	0.0281
RandomForest	0.999	0.043	0.999	0.999	0.999	0.978	0.998	1.000	99.9485	0.0515
LMT	0.998	0.174	0.998	0.998	0.998	0.907	0.884	0.994	99.7892	0.2108
J48	0.998	0.194	0.998	0.998	0.998	0.896	0.889	0.995	99.7658	0.2342
DecisionStump	0.992	0.666	0.992	0.992	0.990	0.568	0.653	0.984	99.1943	0.8057

HoeffdingTree	0.999	0.112	0.999	0.999	0.999	0.941	0.956	0.998	99.8641	0.1359
SMO	0.988	0.976	0.988	0.988	0.982	0.108	0.506	0.977	98.8195	1.1805
Logistic	0.988	0.973	0.988	0.988	0.983	0.125	0.747	0.985	98.8242	1.1758
SGD	0.988	0.973	0.988	0.988	0.983	0.125	0.747	0.985	98.8242	1.1758
SimpleLogistic	0.988	0.980	0.988	0.988	0.982	0.088	0.814	0.984	98.8148	1.1852
Tomcat										
REPTree	0.998	0.050	0.998	0.998	0.998	0.973	0.989	0.998	99.798	99.798
RandomTree	**0.999**	**0.018**	**0.999**	**0.999**	**0.999**	**0.990**	**0.991**	**0.999**	**99.9278**	**0.0722**
RandomForest	**0.999**	**0.018**	**0.999**	**0.999**	**0.999**	**0.990**	**1.000**	**1.000**	**99.9278**	**0.0722**
LMT	0.999	0.024	0.999	0.999	0.999	0.987	0.984	0.998	99.9038	0.0962
J48	0.997	0.075	0.997	0.997	0.997	0.959	0.952	0.993	99.697	0.303
DecisionStump	0.961	0.961	0.924	0.961	0.942	0.000	0.929	0.970	96.1087	3.8913
HoeffdingTree	0.980	0.489	0.981	0.980	0.977	0.693	0.796	0.972	98.0183	1.9817
SMO	0.963	0.925	0.964	0.963	0.945	0.189	0.519	0.928	96.253	3.747
Logistic	0.963	0.915	0.964	0.963	0.946	0.215	0.942	0.980	96.2963	3.7037
SGD	0.964	0.899	0.965	0.964	0.948	0.249	0.532	0.930	96.3588	3.6412

SimpleLogistic	0.962	0.934	0.964	0.962	0.945	0.165	0.952	0.975	96.2193	3.7807

We also extracted CK metrics suite of two versions of ATM and Library system using CKJM tool [179]. We applied same changes in modified versions of these software which were done to validate our previous approaches on UML and formal specifications. We observe that Random Forest and Random Tree classifiers is again giving best results on these subject systems. We observe that Line of code (LOC) and Cyclomatic Complexity (CC) are found to have good characteristic to affect the change identification in response variable as compare to other metrics. We generated set of restestable, reusable and obsolete test cases on these subject systems based on changes identified on 10 fold cross validations. We evaluated and compared our results against Library and ATM system with and Dahiya [77] and Mansour [149] in terms of the number of test cases generated, presented in Table 5.6. We observe that our approach has generated more number of test cases as compare to existing studies.

Table 5.6: Comparative analysis with other studies

Case Studies ➡		**Library System**		**ATM System**	
Types of Test Suite ⬇	Results from [149]	Results from [77]	Our Results	Results from [77]	Our Results
Types of Sources ➡	UML class, Sequence and Activity diagrams	UML class, Sequence and Activity diagrams	Machine Learning	UML class, Sequence and Activity diagrams	Machine Learning
Retestable	7	6	8	3	5
Reusable	76	107	108	158	158
Obsolete	0	2	0	0	0

5.8. Threats to Validity

We used Understand tool and CKJM tool to extract object oriented metrics associated with the software. The CKJM tool works only on class files, whereas Understand tool works only on source files. These tools also have variations in object oriented metrics. We have used the outcome of these tools for developing learning agents. These number of attributes generated by these tools and the counts in representation of metrics in these tools is varied. Nonexistence of standardized representation in object-oriented metrics tools becomes an external threat to validity of machine learning based agents.

Chapter – 6

Conclusions and Future Scope

6.1 Conclusions

The focus of regression test case generation is to generate test cases for changed functionality. Most of the studies in regression test case generation work in localized environments. Regression test case generation is quite complex and testers may require to detect changes or impact of changes in the versions of software. Most of the existing tools are not single integrated platform and call for manual intervention in the form of input and intermediate interfacing. The detected changes, updates and fixes made in various versions of code are not compared in most of the tools. Tester time remains the crucial factor in existing tools and techniques. Some ways and means need to be explored to depute independent programs or tools to work concurrently on behalf of the tester. These programs may have capability of moving from one machine to another and collect information from distributed environment. This research advocates the use of mobile agent-based technology for regression test case generation using syntax and semantics analysis based on model and formal specifications.

We have carried out a systematic literature review of existing test case generation approaches for regression testing and agent based software testing systems in particular. The emphasis is articulated on agent based regression test case generation. The data extracted from our study is classified into seven broader areas of agent based software testing. Based on our systematic literature survey, we recognized available techniques, approaches, platforms as well as methodologies for regression test case generation and developing agent based software testing systems. We noticed that use of agents dominate in web testing and object oriented testing. An increase in use of agents in the test case generation using structural based and model based approaches have been observed. We observed that regression test case generation using model based testing and functional

testing have reported maximum studies during the years for which study is conducted. Symbolic execution and OCL has been preferred by existing studies for structural testing and formal methods based testing. Open source subject systems are majority preferred by these studies for evaluation purpose. It is conquered that agents based on BDI architecture and JADE platform may be assigned different roles in the distributed environment for regression testing in particular. It will reduce time and effort required for testing. Our systematic literature review is helpful in finding research gaps in agent based software testing in general and agent based regression testing in particular.

In the context of regression test case generation, most of the studies have used model based testing. Model based testing is preferred by researchers using the combination of UML class diagram, sequence diagram, activity diagram and use cases. The existing techniques based on models suffer from the fact that using alone class diagram or sequence diagram could not cover all the changed scenarios for desired impact analysis. Apart from it most of the existing studies on regression test case generation are either limited to model based or formal specifications. Thus in order to cope with the difficulties in model base testing, this work proposes three approaches for regression test case generation using mobile agents.

The first approach has carried out the issue of generating regression test cases by integrating model and formal specifications. The proposed technique involves mobile agent-based technology using UML class diagram, sequence diagram and formal specifications based on Object-Z and OCL for envisaging agent-based regression test case generation. Different agents are designed to perform model comparison, behavior comparison, specifications comparison, impact analysis and regression test case generation. Agents designed in JADE framework perform these tasks by using XML files of UML class diagram, sequence diagram and formal specifications based on Object-Z and OCL. XML files corresponding to class diagram, sequence diagram, Object-Z and OCL code have been used as an input to the agent based model. XML files have been parsed to extract classes, associations, dependencies, attributes, operations and constraints. Agent used these entities to compare old and modified versions of code to find a set of retestable,

reusable or obsolete test cases. The integration of model based approach with formal specifications resulted in identifying in-depth changes in syntax and semantics.

The second approach is based on the combined use of UML class diagram, activity diagram and use cases. Agents proceed in the distributed environment to compare XML files of old UML class diagram with modified diagram to identify changes in the software. XMI files of old and modified activity diagrams are used as input for comparison of nodes of activity diagrams. Detailed experiments on student projects related to ATM and Library system reveal that proposed approach has better coverage as compared to existing techniques on model based diagrams.

The third technique uses machine learning approach for identifying changes in the code. In this study we have integrated JADE agents with WEKA tool to design learning based agents. Machine learning based Random Forest and Random Tree classifiers have performed better in identifying change proneness in the software on the basis of Object Oriented metrics.

6.2. Scope of future work

This study will benefit the researchers to carry forward their work in the domain of regression test case generation and agent based software testing. To cut down on schedule and cost, mobile agent based software testing can be a promising alternative. It is observed that agents can be incorporated to reduce human effort and time for testing large complex problems on the distributed environment. Our technique may be applied to test large scale systems with less time and better coverage. An effort has also been made to observe the impact of identified changes on test case generation.

Mobile agent based regression testing is still infancy. In generating regression test cases we need to identify the changes made to various software artifacts. The use of mobile agents to identify software changes syntactically and semantically at code level and model level is still an open research area. This work can be further extended to incorporate agents for test cases prioritisation to reduce the time required for regression testing. Intelligent

agent based regression test cases generation technique may be further investigated using self-learning and adaptive agents.

Our technique requires the presence of UML class models, sequence diagrams and formal specification on the testing machines in the distributed environment. We have integrated JADE based learning agents with Weka tool for using machine learning classifiers. Further agent based frameworks can be explored to integrate agents with machine learning tools for regression test case generation and prioritization.

Apart from some limitations identified as threats to validity, the proposed agent based regression test case generation approach may enhance the existing technological research by integrating the benefits of model based, formal specification based technique and machine learning based agents for testing of industrial applications.

Bibliography

[1] Kim Y., Zu Z., Kim M., Cohen M. B., Rothermel G., "Hybrid directed test suite augmentation: An interleaving framework," In: Proceedings of IEEE Seventh International Conference on Software Testing, Verification and Validation, pp. 263–272, (2014)

[2] Yoo, S.; Harman, M., "Regression testing minimisation, selection and prioritization," A survey. J. Softw. Test. Verif. Reliab. 22(2), pp. 1–7 (2007)

[3] H. Leung and L. White, "Insights into regression testing", In: Proceedings of the Conference on Software Maintenance, pp. 60-69, (1989)

[4] Graves, T. L., Kim, J.M., & Porter, A., "An Empirical Study of Regression Test Selection Techniques," ACM Transactions on Software Engineering and Methodology, 10(2), pp. 184–208, (2001), https://doi.org/ 10.1145/367008 .367020

[5] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, "Prioritizing Test Cases for Regression Testing," IEEE Trans. Software Eng., 27(10), Insights into regression testing, pp. 929-948, (2001)

[6] Costin Z., Budimac B., Burkhard H., "Software Agents : Languages, Tools, Platforms," COMSIS, (2014), DOI: 10.2298/CSIS110214013B

[7] Qusay H. Mahmoud, "Software Agents: Characteristics and Classification," https://pdfs.semanticscholar.org/ead4/0c8efb0299f37f881cf8d90839bd9545f443.pdf accessed on July 2016

[8] Kitchenham, B.; Brereton, O.P., "Systematic literature reviews in software engineering—a systematic literature review," J. Inf. Softw. Technol. 51(1), pp. 7–15, (2009)

[9] Engstorm, E.; Runeson, P.; Skoglund, M., "A systematic review on regression test selection techniques,". J. Inf. Softw. Technol. 52(1), pp. 14–30, (2010)

[10] Rattan, D.; Bhatia, R.; Singh, M., ”Software clone detection: a systematic review,“ J. Inf. Softw. Technol. 55(7), pp. 1165–1199, (2013)

[11] McMinn, P., “Search-based software test data generation—a survey,” J. Softw. Test. Verif. Reliab. 14(2), pp. 105–156, (2004)

[12] Ali, S.; Briand, L.C.; Hemmati, H.; Panesar, R.K.; Walawege, “A: systematic review of the application and empirical investigation of search-based test case generation,” IEEE Trans. Softw. Eng. 36(6), pp. 742-762, (2010)

[13] Anand S. Burke, E.K.; Chen, T.S., “An orchestrated survey of 1168 methodologies for automated software test case generation,” J. Syst. Softw. 86(8), pp. 1978–2001, (2013)

[14] Garg, R. K.; Sharma, K.; Nagpal, C. K., & Garg R; Kumar Rajive., “Ranking of software engineering metrics by fuzzy-based matrix methodology,” https://doi.org/10.1002/stvr, (2011)

[15] Kumar P; Sehgal V.K.; Nitin; Chuhan D.S;, “Performance Evaluation of Evolutionary and Decision Tree Based Classifiers in Diversity of Datasets,” In: Proceedings of International Communication, Network, pp. 309-312, (2012)

[16] Parashar, P;, Kalia, A;, & Bhatia, R., “How Time-Fault Ratio helps in Test Case Prioritization for Regression Testing,” International Journal of Software Engineering, IJSE, 5(2), pp. 25-35, (2012)

[17] Singh Pratima; Tripathi Anil Kumar, “Issues in testing of software with NFR, International Journal of software engineering,” 4(3), pp. 61-76, (2012)

[18] Owens M.D and Khazanchi D., Book chapter Software Quality Assurance, Handbook of Research on Technology Project Management, Planning, and Operations, 10.4018/978-1-60566-400-2, (2009)

[19] Das S, Mihalák M; Šrámek R; Vicari E; Widmayer P, "Rendezvous of Mobile Agents When Tokens Fail Anytime," In: Proceedings of International Conference on Principles of Distributed Systems pp. 463-480, (2008)

[20] Ball, T.; Hoffman, D.; Ruskey, F.; Webber, R.; White, L., "State generation and automated class testing," J. Soft. Test. Verif. Reliab. 10(3), pp.149–170, (2000)

[21] Ricca, F.; Tonella, P., "Analysis and testing of web applications," In: Proceedings of 23rd International Conference on Software Engineering, pp. 25–34, (2001)

[22] Bai, X.; Tsai, W.T.; Paul, R.; Feng, K., Yu, L., "Scenario-based modelling and its applications," In: Proceedings of Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, pp. 253–260, (2002)

[23] Chen, Y.; Probert, R.L.; Ural, H., "Model-based regression test suite generation using dependence analysis," In: Proceedings of the 3rd International Workshop on Advances in Model-Based Testing ACM, pp. 54–62, (2007)

[24] Meyer, B.; Ciupa, I.; Leitner, A.; Liu, L., "Automatic testing of object-oriented software," In: Proceedings of International Conference on Current Trends in Theory and Practice of Computer Science, LNCS (4362), pp. 114–129, (2007)

[25] Gorthi, R.P.; Pasala, A.; Chanduka, K K.; Leong, B., "Specification-based approach to select Regression test suite to validate changed software," In: Proceedings of 15th Asia-Pacific Software Engineering Conference, pp. 153–160, (2008)

[26] Chen, M.; Qiu, X.; Xu, W.; Wang, L.; Zhao, J.; Li, X., "UML activity diagram-based automatic test case generation for Java programs," Comput. J. 52(5), pp. 545–556, (2009)

[27] Filho, R.S.S.; Budnik, C.J.; Hasling, W.M.; McKenna, M.; Subramanyan, R., "Supporting concern-based regression testing and prioritization in a model-driven environment," In: Proceedings of IEEE 34th Annual Conference workshop on Computer Software and Applications, pp. 323–328, (2010)

[28] Fourneret, E.; Bouquet, F.; Dadeau, F.; Debricon, S., “Selective test generation method for evolving critical systems,” In: Proceedings of Fourth IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 125–134, (2011)

[29] Cho, Y.; Lee, W.; Chong, K., “The technique of test case design based on the UML sequence diagram for the development of web applications,“ In: Proceedings of International Conference on Computational Science and Its Applications, LNCS (348), pp. 1–10, (2005)

[30] Naslavsky, L.; Ziv, H.; Richardson, D.J., MbSRT2: “Model-based selective regression testing with traceability,” In: Proceedings 3rd International Conference on Software Testing, Verification and Validation, pp. 89–98, (2010)

[31] Kumar, R.; Bhatia, R.K.: “Interaction diagram based test case generation,” In Proceedings of International Conference on Glob. Trends Inf. Syst. Softw. Appl. Commun. Comput. Inf. Sci. 270, pp. 202–211, (2012)

[32] Lamancha, B.P.; Polo, M.; Caivano, D.; Piattini, M.; Visaggio, G., “Automated generation of test oracles using a model-driven approach,” J. Inf. Softw. Technol. 55(2), pp. 301–319, (2013)

[33] Vincent, P.-L.; Badri, L.; Badri, M., “Regression testing of object oriented software-A technique based on use cases and associated tool,” In: Proceedings of International Conference on Computer Applications for Software Engineering, Disaster Recovery and Business Continuity, pp. 96–106, (2012)

[34] Ye, N.; Chen, X.; Ding,W.; Jiang, P.; Bu, L.; Li, X., “Regression test cases generation based on automatic model revision,” In: Proceedings of Sixth International Symposium on Theoretical Aspects of Software Engineering (TASE), pp. 127–134, (2012)

[35] Zech, P.; Felderer, M.; Kalb, P.; Breu, R., "A Generic platform for model-based regression testing," In: Proceedings of Leveraging Applications of Formal Methods, Verification and Validation, Technologies for Mastering Change, LNCS (7609), pp. 112–126, (2012)

[36] Fourneret, E.; Cantenot, J.; Bouquet, F.; Legeard, B.; Botella, J., "SeTGaM, generalized technique for regression testing based on UML/OCL models", In: Proceedings of International Conference on Software Security and Reliability, pp. 147–156, (2014)

[37] Sapna, P.G.; Balakrishnan, A., "An approach for generating minimal test cases for regression testing," J. Procedia Comput. Sci. 47, pp. 188–196, (2015)

[38] Artzi, S.; Kie, A.; Perkins, J., "Automatic generation of unit regression tests," http://mit.kku.edu.sa/NR/rdonlyres/Electrical-Engineering-and-Computer-science/6-883Fall-2005/9081C0D-1335-4EFB-A12F-EADBC149C33F/0/unit_regression.pdf (2015), Accessed 23 July 2016

[39] Tsai, W.T.j; Bai, X.; Paul, R.; Shao, W.; Agarwal, V., "End to end integration testing design," In: Proceedings of International Conference of Computer Software and Applications, COMPSAC, pp. 166–171, (2001)

[40] Xie, T.: "Augmenting automatically generated unit-test suites with regression oracle checking," In: Proceedings of 20th European Conference on Object-Oriented Programming, LNCS (4067), pp. 380–403, (2006)

[41] Muccini, H.; Dias, M.; Richardson, D.J., "Software architecture based regression testing," J. Syst. Softw. 79, pp. 1379–1396, (2006)

[42] Qu, X.; Cohen, M.B.; Woolf, K.M., "Combinatorial interaction regression testing: a study of test case generation and prioritization," In: Proceedings of IEEE International Conference on Software Maintenance, pp. 255–264, (2007)

[43] Li, B.; Qiu, D.; Ji, S.; Wang, D., “Automatic test case selection and generation for regression testing of composite service based on extensible BPEL flow graph,” In: Proceedings of IEEE International Conference on Software Maintenance, ICSM, pp. 1–10, (2010)

[44] Zhang, T.; Yao, Q., “An approach of end user regression testing for semantic web services,” In: Proceedings of International Conference on Management and Service Science (MASS), pp. 1–4, (2011)

[45] Jolly, S.A.; Garousi, V.; Eskandar, M.M., “Automated unit testing of a SCADA control software—an industrial case study based on action research,” In: Proceedings of IEEE Fifth International Conference on Software Testing, Verification and Validation, pp. 400–409, (2012)

[46] Masood, T.; Nadeem, A.; Ali, S., “An automated approach to regression testing of web services based on WSDL operation changes,” In: Proceedings of IEEE 9th International Conference on Emerging Technologies (ICET), pp. 1–5, (2013)

[47] Gladisch, C.; Tyszberowicz, S.; Beckert, B.; Yehudai, A., “Generating regression unit tests using a combination of verification and capture & replay,” In: Proceedings of International Conference on Tests and Proofs, LNCS (6143), pp. 61–76, (2010)

[48] Swearngin, A.; Cohen, M.B.; John, B.E.; Bellamy, R.K.E., “Human performance regression testing,” In: Proceedings of International Conference on Software Engineering, pp. 152–161, (2013)

[49] Rodríguez, F.T.; Reina, M.; Baptista, F.; Usaola, M. P.; Lamancha B. P., “Automated generation of performance test cases from functional tests for web applications,” In: Evaluation of Novel Approaches to Software Engineering, 417, Communications in Computer and Information Science, pp. 164–173, (2014)

[50] Stefan, I.; Ivan, I.; Miclea, L., "Assisted test case design using contextual information by DOM exploration," In: IEEE International Conference on Automation, Quality and Testing, Robotics, pp. 1–4, (2014)

[51] Marin, M.: "A data-agnostic approach to automatic testing of multi-dimensional databases," In: Proceedings of IEEE Seventh International Conference on Software Testing, Verification and Validation, pp. 133–142, (2014)

[52] Mariani, L.; Riganelli, O.; Santoro, M.; Muhammad, A., "G Rank Test: dynamic analysis and testing of upgrades in LabVIEW," In: Chockler, H., Kroening, D., Mariani, L., Sharygina, N. (eds.) Validation of Evolving Software, pp. 107–121. Springer (2015)

[53] Taneja, K.; "DiffGen: automated regression unit-test generation," In: Proceedings of 23rd IEEE/ACM International Conference on Automated Software Engineering, pp. 407–410, (2008)

[54] Lee, G.; Morris, J.; Parker, K.; Bundell, G.; Lam, P., "Using symbolic execution to guide test generation," J. Softw. Test. Verif. Reliab. 15(1), pp. 41–61, (2005)

[55] Zhang, Z.; Huang, J.; Zhang, B.; Lin, J.; Chen, X., "Regression test generation approach based on tree-structured analysis," In: Proceedings of International Conference on Computational Science and Its Applications (ICCSA), pp. 244–249, (2009)

[56] Taneja, K.; Xie, T.; Tillmann, N.; De Halleux, J., "eXpress: guided path exploration for efficient regression test generation," In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 1–11, (2011)

[57] Jamrozik, K.; Fraser, G.; Tillman, N.; De Halleux, J., "Generating test suites with augmented dynamic symbolic execution," In: Proceedings of the 7th International Conference on Tests & Proofs, LNCS (7942), pp. 152-167, (2013)

[58] Bohme, M.; Oliveira, B.C.D.S.; Choudhury, A.R., "Partition-based regression verification," In: Proceedings of the International Conference on Software Engineering, pp. 302–311, (2013)

[59] Tiwari, S.; Mishra, K.K.; Misra, A.K., "Test Case Generation for Modified Code using a Variant of Particle Swarm Optimization PSO) Algorithm," In: Proceedings of Tenth International Conference on Information Technology, New Generations, pp. 363–368, (2013)

[60] Eda, R., "Regression testing for web applications using reusable constraint values," In: Proceedings of IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 312–321, (2014)

[61] Braione, P.; Denaro, G.; Riganelli, O.; Baluda, M.; Muhammad, "A.: Static/dynamic test case generation for software upgrades via ARC-B and Deltatest." In: Chockler, H., Kroening, D., Mariani, L., Sharygina, N. (eds.), Validation of Evolving Software, pp. 147– 184. Springer, Cham (2015)

[62] Blanco, R.; Tuya, J.; Adenso-Díaz, B, "Automated test data generation using a scatter search approach", J. Inf. Softw. Technol. 51(4), pp. 708–720, (2009)

[63] Shamshiri, S.; Fraser, G.; McMinn, P.; Orso, A., "Search-based propagation of regression faults in automated regression testing", In: Proceedings of IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 396–399, (2013)

[64] Stotts, D.; Lindsey, M.; Antley, A., "An informal formal method for systematic Junit test case generation," In: Proceedings of International Conference on Extreme Programming and Agile Methods—XP/Agile Universe, LNCS (2418), pp. 131–143, (2002)

[65] Tsai, W. T.; Yu, L.; Liu, X.X.; Saimi, A.; Xiao, Y., "Scenario based test case generation for state-based embedded systems," In: Proceedings of IEEE

International Conference Performance, Computing, and Communications, pp. 335–342, (2003)

[66] Cavarra, A.; Crichton, C.; Davies, J., “A method for the automatic generation of test suites from object models,” J. Inf. Softw. Technol. 46(5), pp. 309–314, (2004)

[67] Xu, L.; Dias, M.; Richardson, D., “Generating regression tests via model checking,” In: Proceedings of International Conference on Computer, Software and Applications, pp. 336–341, (2004)

[68] Fraser, G.; Aichernig, B.K.; Wotawa, F., “Handling model changes, regression testing and test-suite update with model-checkers,” In: Proc. of Third Workshop Model Based Test. Electron. Notes Theor. Comput. Sci. 190(2), pp. 33–46, (2007)

[69] Askarunisa, A.; Abirami, A.M.; Madhan Mohan, S., “A test case reduction method for semantic based web services,” In: Proceedings of Second International Conference on Computing Communication and Networking Technologies (ICCCNT), pp. 1–7, (2010)

[70] Xu, D.: “A tool for automated test code generation from high-level petri nets,” In: Proceedings of the 32nd International Conference on Applications and Theory of Petri Nets, pp. 308–317, (2011)

[71] Zhang, L.; Xie, T.; Zhang, L.; Tillmann, N.; Halleux, J.; de Mei, H., “Test generation via dynamic symbolic execution for mutation testing,” In: Proceedings of IEEE International Conference on Software Maintenance, pp. 1–10, (2010)

[72] Fraser, G.; Arcuri, A.; “EvoSuite: automatic test suite generation for object-oriented software,” In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 11–14, (2011)

[73] Parizi, R.M.; Ghani, A.; Lee, S.P, “Automated test generation technique for aspectual features in AspectJ,” J. Inf. Softw. Technol. 57, pp. 463–493, (2015)

[74] Li, K.; Yang, Z., “Generating method of pair-wise covering test data based on ACO,” In: Proceedings of International Workshop on Geoscience and Remote Sensing, pp. 776–779, (2009)

[75] Robinson, B.; Ernst, M.D.; Perkins, J.H.; Augustine, V., Scaling up automated test generation: “Automatically generating maintainable regression unit tests for programs,” In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 23–32, (2011)

[76] Alipour, M.A.; Groce, A.; Gopinath, R.; Christi, A., “Generating Focused Random Tests using Directed Swarm Testing,” In: Proceedings of International Symposium on Software Testing and Analysis, pp. 70–81, (2016)

[77] Dahiya, S.; Bhatia, R.K.; Rattan, D., “Regression test selection using class, sequence and activity diagrams,” IET Softw. 10(3), pp. 72–80, (2016)

[78] Gardikiotis, S.K.; Lazarou, V.S.; Malevris, N., “Employing agents towards database applications testing.” In: Proceedings of IEEE International Conference on Tools with Artificial Intelligence, pp. 102–116, (2007)

[79] Sharma, A.; Capretz, M.A.M.: “Application maintenance using software agents,” In: Proceedings of IEEE International Workshop Code Analysis and Manipulation, pp. 55–64, (2000)

[80] Qingning, H.; Hong, Z.; Greenwood, S., “A multi-agent software environment for testing web based applications,” In: Proceedings of 27th Annual International Conference on Computer Software and Applications, pp. 210–215, (2003)

[81] Hong, Z., “Cooperative agent approach to quality assurance and testing web software,” In: Proceedings of IEEE Int. Conf. Comput. Softw. Appl. 2, pp. 110–113, (2004)

[82] Kung, D., “An agent based framework for testing web applications,” In: Proceedings of Int. Comput. Softw. Appl. Conf. 2, pp. 174–177, (2004)

[83] Yu, Q.; Kung, D.; Wong, E., “An agent-based data-flow testing approach for web applications,” J. Inf. Softw. Technol. 48(12), pp.1159–1171, (2006)

[84] Xiaoying, B.; Guilan, D.; Dezheng, X.; Tsai, W.T., “A multi-agent framework for collaborative testing on web services,” In: Proceedings of Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp. 205–210, (2006)

[85] Huaikou, M.; Chen, S.; Zhongsheng, Q., “A formal open framework based on agent for testing web applications,” In: Proceedings of International Conference on Comput. Intell. Secur. 2, pp. 281–285, (2007)

[86] Xiaoying, B.; Dezheng, X.; Guilan, D.; Tsai, W.-T.; Chen, Y., “Dynamic reconfigurable testing of service oriented architecture,” In: Proceedings of International Conference on Computer Software and Applications, pp. 1–5, (2007)

[87] Dong, W., “Multi-agent environment for BPEL-based web service composition,” In: Proceedings of International Conference on Cybernetics and Intelligent Systems, pp. 855–860, (2008)

[88] Zhang, J.; Xu, D., “A mobile agent supported web services testing platform,” In: Proceedings of International Conference on Embed. Ubiquitous Comput., 2, pp. 637–644, (2009)

[89] Wang, H.-R.; Li, Y.; Li, X., “A mobile agent based general model for web testing,” In: Proceedings of Second Int. Symp. Comput. Intell., 2, pp. 158–161, (2009)

[90] Ma, B.; Chen, B.; Xiaoying, B.; Huang, J., ”Design of BDI agent for adaptive performance testing of web services,” In: Proceedings of 10th International Conference on Quality Software, pp. 435–440, (2010)

[91] Zhao, C.; Guoxin, A.; Xiao, Y.; Wang, X., “Research on automated testing framework based on ontology and multi agent,” In: Proceedings of Third

Symposium on Knowledge Acquisition and Modelling (KAM), pp. 206–209, (2010)

[92] Hao, D.; Chen, Y.; Tang, F.; Feng, Q., "Distributed agent-based performance testing framework of web services," In: Proceedings of International Conference on Software Engineering and Service Sciences (ICSESS), pp. 90–94, (2010)

[93] Zhang, J., "A Mobile agent-based tool supporting web services testing," J. Wirel. Pers. Commun 56, pp. 147–172, (2011)

[94] Paydar, S.; Khani, M., "An agent-based framework for automated testing of web-based systems," J. Softw. Eng. Appl. 4(2), pp. 86–94, (2011)

[95] Grundy, J.; Ding, G.; Hosking, J., "Deployed software component testing using dynamic validation agents," J. Syst. Softw. 74, pp. 5–14, (2005)

[96] Dhavachelvan, P.; Uma, G.V., "Fuzzy complexity assessment model for resource negotiation and allocation in agent based software testing framework," Int. J. Expert Syst. Appl. 29(1), pp.105–119, (2005)

[97] Dhavachelvan, P.; Uma, G.V., "Multi-agent based integrated framework for intra-class testing of object-oriented software," J. Appl. Soft Comput. 5(2), pp. 205–222, (2005)

[98] Xu, D.; Li, H.; Lam, C.P., "Using adaptive agents to automatically generate test scenarios from the UML activity diagrams," In: Proceedings of 12th Asia-Pacific Software Engineering Conference (APSEC05), pp. 385–392, (2005)

[99] Li, H.; Lam, C.P., "Using Anti-Ant-like Agents to generate test threads from the UML diagrams," In: Proceedings of International Conference on Testing of Communicating Systems, LNCS (3502), pp. 69–80, (2005)

[100] Dhavachelvan, P.; Uma, G.V., "Evolution of agent-oriented distribution model for software testing," A layered approach. Int. Arab J. Inf. Technol. 3, pp. 111–117, (2006)

[101] Dhavachelvan, P.; Uma, G.V.; Venkatachalapathy, V.S.K., "A new approach in development of distributed framework for automated software testing using agents," J. Knowl. Based Syst. 19, pp. 235–247, (2006)

[102] Mala, D.J.; Mohan, V., "IntelligenTester–Test sequence optimization framework using graph based intelligent search agents." In: Proceedings of International Conference on Computational Intelligence and Multimedia Applications, pp. 22–27, (2007)

[103] Siwen, Y.; Jun, A., "Software test data generation based on multi agent," In: Proceedings of Adv. Softw. Eng. Appl. 59, pp. 188–195, (2009)

[104] Devasena, M.S.G.; Valarmathi, M.L., "Multi agent based framework for structural and model based test case generation," In: Proceedings of International Conference on Modelling Optimization and Computing, pp. 3840–3850, (2012)

[105] Dam, H.K.; Ghose, A., "Supporting change impact analysis for intelligent agent systems," J. Sci. Comput. Program. 78(9), pp. 1728– 1750, (2013)

[106] Baiquan, X., "Design of platform for performance testing based on JADE," In: Proceedings Sixth International Conference on Measuring Technology, Mechatronics Automation, pp. 251–254, (2014)

[107] Ramdane-Cherif, A.; Benarif, S.; Levy, N., "The platform based agents to test and evaluate software architecture," J. Object Technol. 4(1), pp. 67–82, (2005)

[108] Hany, F.; Yamany, E.; Capretz, M.A.M.; Capretz, L.F., "A multi agent architecture for testing distributed system," In: Proceedings of International Conference. Comput. Softw. Appl. 2, pp. 151–156, (2006)

[109] Jing, G.; Yuquing, L., "Agent-based distributed automated testing executing framework," In: Proceedings of International conference on Computational Intelligence and Software Engineering, pp. 1–5, (2009)

[110] Jing, G.; Yuquing, L., “Automatic test task allocation in agent based distributed automated testing,” In: Proceedings of International Conference on Computational Intelligence and Software Engineering, pp. 1–5, (2009)

[111] Chu, H.-D., “A blackboard-based decision support framework for testing client/server applications,” In: Proceedings of Third World Congress on Software Engineering, pp. 131–135, (2012)

[112] Charaf, M.E.H.; Benattou, M.; Azzouzi, S., “A rule-based multiagent system for testing distributed applications,” In: Proceedings of International Conference on Multimedia Computing and Systems (ICMCS), pp. 967–972, (2012)

[113] Zunliang, Y.; Chunyan, M.; Yuan, M.; Zhiqi, S., “Actionable knowledge model for GUI regression testing,” In: Proceedings of International Conference on Intelligent Agent Technology, pp. 165–168, (2005)

[114] Salima, T. M. S.U.; Askarunisha, A.; Ramaraj, N., “Enhancing the efficiency of regression testing through intelligent agents,” In: Proceedings of International Conference on Computational Intelligence and Multimedia Applications, pp. 103–108, (2007)

[115] Srivastava, P.R.; Kim, T.-H., “Agent based approach to regression resting,” In: Proceedings of International Conference on Advances in Computer Science and Information Technology, LNCS (6059), pp. 345–355, (2010)

[116] Bassil, Y.: Distributed, “Cross platform and regression testing architecture for service-oriented architecture,” J. Adv. Comput. Sci. Appl. 1(1), pp. 1–8, (2012)

[117] Liang, L.; Xing-She, Z.; Jian-Hua, G.; Yang, Z.-Y., “Agent based automated compatibility software test for NLSF,” In: Proceedings of International Conference on Machine Learning Cybernetics, 4, pp. 1986–1989, (2003)

[118] Chengqing, Y.; Yinglong, W.; Wang, J., "A IPv6 network performance test system using multi agent," In: Proceedings of International Conference on Electronic Measurement and Instruments, pp. 113–118, (2007)

[119] Ilarri, S.; Mena, E.; Illarramendi, A., "A system based on mobile agents to test mobile computing application," J. Netw. Comput. Appl. 32(4), pp. 846–865 (2009)

[120] Friess, N.; Crawford, H.; Aycock, J., "A multi-agent approach to testing anti-spam software," In: Proceedings of International Workshop on Database and Expert Systems Application, pp. 38–42, (2009)

[121] Manzoor, U.; Irfan, J.; Nefti, S., "Autonomous agents for testing and verification of software after deployment over network," In: Proceedings of World Congress on Internet Security, pp. 36–41, (2011)

[122] Askarunisa, A., "Agent based analysis of test case prioritization techniques for software operations," PH.D Thesis, Anna University, Chennai, (2010)

[123] Malz, C.; Göhner, H.C.P., "Agent-based test case prioritization," In: Proceedings of fourth International Conference on Software Testing. Verification and Validation Workshops, pp. 149–152, (2011)

[124] Rongfa, T., "Adaptive software test management system based on software agents," In: Proceedings of 3rd International Conference on Teaching and Computational Science, vol. 117, pp. 1–9, (2012)

[125] Malz, C.; Jazdi, N.; Gohner, P., "Prioritization of test cases using software agents and fuzzy logic, software testing," In: Proceedings of IEEE Fifth International Conference on Software Testing, Verification and Validation, pp. 483–486, (2012)

[126] Tang, J., "Towards automation in software test life cycle based on multi agent," In: Proceedings of International Conference on Computational Intelligence and Software Engineering, pp. 1–4, (2010)

[127] Dragunov, Y.P.; Sahin, F., “Agent in the loop simulation for testing swarm robots using an XML-based system of system framework in discrete-event simulation specification,” In: Proceedings of 6th International Conference on System of Systems Engineering, pp. 293–298, (2011)

[128] Sivakumar, N.; Kalimuthu, K.V.; Hemnandh, S.; Kumar, S.P.; Sasidharan, E., “Test suite amelioration using case based reasoning for an agent based system,” In: Proceedings of 4th International Conference on Global Trends in Information Systems and Software Applications, pp. 222–232, (2012)

[129] Merdan, M.; Vrba, P.; Melik-Merkumians, M., “Test-driven agent oriented software development,” In: Proceedings of IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1–8, (2012)

[130] Chen, G.; Wang, G., “Software fault diagnosing system based on multi agent,” In: Proceedings of International Conference on Third Global Congress on Intelligent Systems, pp. 327–329, (2012)

[131] Öztürk, M.M.; C˙IL, ˙I.; Zengin, A., “Development of a multi-agent framework for software quality,” ACM SIGSOFT Softw. Eng. Notes 40(1), pp. 1–10, (2015)

[132] Dejanira A-I, A.G Pipe.; Eder, K., “Intelligent Agent-Based Stimulation for Testing Robotic Software in Human-Robot Interactions,” In: Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering, pp. 9–16, (2016)

[133] Klammer, C., & Ramler, R., “Harnessing Automated Test Case Generators for GUI Testing in Industry,” In: Proceedings of International Conference on Software Engineering and Advanced Applications, pp. 227–234, (2016)

[134] Hui, Z., ”Fault Localization Method Generated by Regression Test Cases on The Basis of Genetic Immune Algorithm,” In: Proceedings of International Conference on Software and Applications, pp. 46-51, (2016)

[135] Padmanban. R. Thirumaran, M., Suganya, K., & R, V. P., "AOSE Methodologies and Comparison of Object Oriented and Agent Oriented Software Testing," In: Proceedings of International Conference on Informatics and Analytics, pp. 1-16, (2016)

[136] Hammoudi, M., "Regression Testing of Web Applications using Record/ Replay Tools," In: Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 1079–1081, (2016)

[137] Makki M., Landuyt, D. Van.Wouter Jousen, "Automated Regression Testing of BPMN 2.0 Processes," In: Proceedings of the ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, pp. 178–189, (2016)

[138] Schatten, M., & Tomi, "Towards an Agent-Based Automated Testing Environment for Massively Multi-Player Role Playing Games," In: Proceedings of International convention on Information and communication Technology, Electronics and Microelectronics, pp. 1149–1154, (2017)

[139] Al-refai, M. Cazzola W, Ghosh S, "A Fuzzy Logic Based Approach for Model-based Regression Test Selection," In: Proceedings of International Conference on Model Driven Engineering Languages and Systems, pp. 55-62, (2017)

[140] Sahoo, S., & Ray, A., "A framework for Optimization of Regression testing of Web Services Using Slicing," In: Proceedings of International Conference on Advances in Computing, Communications and Informatics, pp. 1017–1022, (2017)

[141] Araújo, J., Moreira, A., "Specifying the behaviour of UUML collaborations using object-Z," In: Proceedings of Americas Conf. on Information Systems (AMCIS), pp. 380–387, (2000)

[142] Miao, H., Liu, L., Li, L., "Formalizing UML models with object-Z," In: Proceedings of 4th International Conference on Formal Engineering Methods, ICFEM 2002 (LNCS, 2495), pp. 523–534, (2002)

[143] Roussev, B., "Generating OCL specifications and class diagrams from use cases: a Newtonian approach," In: Proceedings of 36th Annual Hawaii Int. Conf. on System Sciences, ISBN 0-7695-1874-5, (2002)

[144] Mingsong, C., Xiaokang, Q., Xuandong, L., "Automatic test case generation for UML activity diagrams," In: Proceedings of the Int. workshop on Automation of software test, pp. 2–8, (2006)

[145] Luigi, B.F., Giovanni, C., "Developing multi-agent systems with JADE," ISBN 978-0-470-05747-6, pp. 1–300, (2007)

[146] Ali, S., Society, I.C., "Generating test data from OCL constraints with search techniques," Trans. Software Eng., 39, (10), pp. 1376–1402, (2013)

[147] Briand, L.C., Labiche, Y., Osullivan, L., "Automating impact analysis of UML models," Journal of System Software, 79, (3), pp. 339–352, (2006)

[148] Cavarra, A., "A data-flow approach to test multi-agent ASMs," J. Form. Asp. Comput., 23, (1), pp. 21–41, (2011)

[149] Mansour, N., Takkoush, H., Nehme, A., "UML-based regression testing for OO software," J. Softw. Maint. Evol.: Res. Practice, 23, (1), pp. 51–68, (2011)

[150] Graeme, S., "The object-Z specification language," Adv. Form. Methods, 1, pp. 1–146, (2000)

[151] Cabot, J., Gogolla, M., "Object constraint language (OCL): a definitive guide. Formal Methods for Model-Driven Engineering," (LNCS, 7320), pp. 58–90, (2012)

[152] Object Aid Tool, available at http://objectaid.com/, accessed on July 2016

[153] Eclipse CZT Tool, available at http://czt.sourceforge.net/, accessed on July 2016

[154] Hettab, A., Kerkouche, E., & Chaoui, A., "A Graph Transformation Approach for Automatic Test Cases Generation from UML Activity Diagrams," In: Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering, pp. 88-97, (2015)

[155] Wang, C., Pastore, F., Goknil, A., Briand, L., & Iqbal, Z. (2015), "Automatic Generation of System Test Cases from Use Case Specifications," In: Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2015), 12, pp.385-396, (2015)

[156] Sunitha, E. V, & Samuel, P., (2013), "Enhancing UML Activity Diagrams using OCL," In: Proceedings of International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1-6, (2013)

[157] Shirole, M., Kommuri, M., & Kumar, R., "Transition sequence exploration of UML activity diagram using evolutionary algorithm," In: Proceedings of the 5th India Software Engineering Conference on - ISEC 12, pp. 97–100 (2012)

[158] Ye, N., Chen, X., Jiang, P., Ding, W., & Li, X., "Automatic Regression Test Selection based on Activity Diagrams," https://doi.org/10.1109/SSIRI-C.2011.31 (2011)

[159] Sapna, and Mohanty, H., "Prioritization of Scenarios Based on UML Activity Diagrams," In: Proceedings of Conference on Computational Intelligence, Communication Systems and Networks, pp. 271–276, (2009)

[160] Kim, H., Kang, S., Baik, J., & Ko, I., "Test Cases Generation from UML Activity Diagrams," In: Proceedings of International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp. 556–561, (2007)

[161] Li, H., & Lam, C. P., "Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams," In: Proceedings of Testing of Communicating Systems, LNCS (3502), pp. 69–80, (2005)

[162] Argouml tool, available at http://argouml.tigris.org/ accessed in Nov. 2017

[163] K.K.Aggarwal, Singh Y, Kaur Arvinder, Malhotra Ruchika, "Empirical Study of Object-Oriented Metrics," Journal of Object Technology 5(8), pp. 149–173, (2006)

[164] Briand, L. C., Labiche, Y., Bawar, Z., & Traldi, N. "Using machine learning to refine Category-Partition test specifications and test suites, Information and Software Technology, 51(11), 1551–1564, (2009)

[165] Gove, R., & Faytong, J., "Machine Learning and Event-Based Software Testing : Classifiers for Identifying Infeasible GUI Event Sequences," Advances in Computers, 86, pp. 109-135, (2012)

[166] Silva, D. G., Jino, M., & Abreu, B. T. De. "Machine learning methods and asymmetric cost function to estimate execution effort of software testing", In: proceedings of Third International Conference on Software Testing, Verification and Validation (ICST) pp. 275-284, (2010)

[167] Malhotra, R., "Examining the Effectiveness of Machine Learning Algorithms for Prediction of Change Prone Classes," In: Proceedings of International Conference on High Performance Computing & Simulation, pp. 635–642, (2014)

[168] Pang, Y., "Identifying Effective Test Cases Through K -means Clustering for Enhancing Regression Testing," In: Proceedings of International Conference Machine Learning and Applications (ICMLA), pp. 78-83, (2013)

[169] Noorian, M., Bagheri, E., & Du, W. (n.d.)., "Machine Learning-based Software Testing : Towards a Classification Framework," In: Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), USA, pp. 1-5, (2011)

[170] Zhang, D., "Machine Learning in Value-Based Software Test Data Generation," In: Proceedings of IEEE International Conference on tools with Artificial Intelligence, ICTAI '06, pp. 87-119, (2006)

[171] Lachmann, R., Nieke, M., Seidl, C., Schaefer, I., & Schulze, S., "System-Level Test Case Prioritization Using Machine Learning," In: Proceedings of 15th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 361-368, (2016)

[172] Gondra, I., "Applying machine learning to software fault-proneness prediction," Journal of Systems and Software, 81(2), pp. 186–195, (2008)

[173] Pierre-Christophe Bue, Dadeau F., Heam Pierre-Cyrille, "Model-Based Testing using Symbolic Animation and Machine Learning," In: Proceedings IEEE of International Conference on Software Testing, Verification, and Validation Workshops, pp. 355–360, (2010)

[174] Zheng. Li, Li M., Li Y., and Geng J., "Identify Coincidental Correct Test Cases based on Fuzzy Classification," In: Proceedings IEEE of International Conference on Software Analysis, Testing and Evolution Identify, pp. 72-77, (2016)

[175] Chen, S., Chen, Z., Zhao, Z., Xu, B., & Feng, Y., "Using Semi-Supervised Clustering to Improve Regression Test Selection Techniques," In: Proceedings of IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), pp. 1–10, (2011)

[176] Cotroneo, D., Pietrantuono, R., Russo, S., Federico, N., Savy, C. C., Universitario, C., & Sant, M., "A Learning-Based Method for Combining Testing Techniques," In: Proceedings of 35th International Conference on Software Engineering (ICSE), pp. 142–151, (2013)

[177] Malhotra, R., "Mining the Impact of Object Oriented Metrics for Change Prediction using Machine Learning and Search-based Techniques," In: Proceedings of International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 228–234, (2015)

[178] Lincke R, Lundberg J, Löwe W, "Comparing Software Metrics," In: Proceedings of the international symposium on Software testing and analysis," pp. 131-142, (2008)

[179] ckjm tool, available at https://www.spinellis.gr/sw/ckjm/ accessed on Nov 2017

[180] Understand tool, available at https://scitools.com/ accessed on Nov 2017

[181] Weka tool, available at https://www.cs.waikato.ac.nz/ml/weka/

[182] Farooq, Q., M.Z.Z Riebisch, M., “A model-based regression testing approach for evolving software systems with flexible tool support,” In: Proceedings of 7th International Conference and Workshops on Engineering of Computer Based Systems (ECBS), pp. 41–49, (2010)