Back End Development and Unit Test Case Generation for Collaborative Invention Mining

Abhishek Singh Yadav

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

Back End Development and Unit Test Case Generation for Collaborative Invention Mining

Dissertation submitted in

May 2014

to the department of

Computer Science and Engineering

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Abhishek Singh Yadav

(Roll 212CS1101)

under the supervision of

Prof. Durga Prasad Mohapatra

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

dedicated to Lord Krishna and my beloved family...

Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, India. www.nitrkl.ac.in

Certificate

This is to certify that the work in the thesis entitled *Back End Development and Unit Test Case Generation for Collaborative Invention Mining* by *Abhishek Singh Yadav*, bearing roll number 212CS1101, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: May 23, 2014

Dr. Durga Prasad Mohapatra
Associate Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgement

This dissertation, though an individual work, has benefited in various ways from several people. Whilst it would be simple to name them all, it would not be easy to thank them enough.

The enthusiastic guidance and support of *Prof. Durga Prasad Mohapatra* inspired me to stretch beyond my limits. His profound insight has guided my thinking to improve the final product. My solemnest gratefulness to him.

I am also grateful to *Prof. S.K.Rath,Head,CSE* for his excellent support throughout my research work. I am greatly indebted to his invaluable advice and support in almost every aspect of my academic life. My sincere thanks to *Prof. B.Majhi* for his continuous encouragement and invaluable support. It is indeed a privilege to be associated with people like *Prof. S.K.Jena*, *Prof.P.M.Khilar*, *Prof. A. K. Turuk*, *Prof. S.Chinara*, *Prof. Pankaj Sa*, *Prof. B. D. Sahoo*, *Prof. Ratnakar Dash* , *Dr. Suman Bhattacharya* and *Dr. Debiprasad Swain.* They have made available their support in a number of ways.

I have been fortunate to have met great friends throughout my M. Tech journey. I am forever grateful for their support and encouragement.

I am grateful to all the staffs of the computer science department for their help in generous help in various ways for the completion of this thesis. Last but not least, I forever indebted to my parents, my sisters and the rest of my family. They have been a great source of inspiration for me. This would have not been possible without their love, support and continuous encouragement.

Abhishek Singh Yadav

Abstract

Software testing plays a very important role in the development process because it is an essential mean of providing reliability and quality to any software. Designing and execution of test cases consumes lot of time since it requires planning and resources manually. To overcome this it is highly required to automate the generation of test cases.

UML, is a standard modeling language which supports object oriented technology. It is generally used to depict the analysis and design specification of software development. UML models are an essential and a rich source of information for test case design. In this thesis, we present a testing methodology which is used to generate the unit test cases from UML state chart diagram for an industrial application

Firstly, we discussed about the CIM, an industrial application which is used for transforming an Idea through a collaborative interaction into such a state so that the idea becomes patentable. This application provides a platform to the people in industry to encourage and enhance their invention skills for an enterprise.

Next we proposed a testing approach to generate unit test cases for the phases of CIM application using UML state chart diagram. UML model provides a lot of information which can be used for testing. In our approach, firstly the state chart diagram is constructed. for CIM application. Then the adjacency matrix is generated and subsequently transform the state chart diagram into a UML state chart graph. Then we traverse the graph using adjacency matrix by using DFS. Therefore test sequences are generated. Then we apply the node coverage minimization technique to generate the test cases so that maximum coverage is achieved.

***Keywords*:** Unified Modeling Language,State chart diagram,Test Sequences,Test Cases,State chart graph,CIM.

Contents

List of Figures

List of Tables

Chapter 1

Inroduction

Invention mining is the process converting a raw idea into a mature idea so that it becomes patentable in a selected set of jurisdiction by collaborative deliberation. Invention mining is the procedure of increasing the strength of an idea by means of collaborative interaction. The primary purpose of this process is to divide the idea in several dimensions such as category dimension, are coverage and business characteristics to widen, lengthens and deepens the idea by collaborative deliberation. So further, the idea gets matured and compose it as a sustainable invention that can be patented.It is the persistent methodology of working to recognize the potential innovations at an initial stage of the development cycle.

Software Testing is the way of exercising a software or application with the purpose of seeing bugs and observing failures. In other words we can say that Software Testing is the process of executing a executing a program with the intent of finding an errors [1]. Software testing is considered as an art that helps in improving the quality of the software product. It is the process of identifying an error in order to fix those errors. It helps to make the software error free.

According to the definition of IEEE (USA): Software testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item (IEEE/ANSI [Std 829-1983]) . Testing assist the softwares to measure its attribute in terms of defect found in both operative and non functional requirements. Some

of the objective of software testing are as follows:

- A successful test is one that uncovers an as yet undiscovered errors [1].
- Product Quality is ensured by software testing.
- Software testing helps to reduce risk.
- Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results [2]. In recent years, softwares are becoming complex and larger. Software mainly consists of system documentation, user documentation, configuration files and programs. Software testing is one of the most important phase of software development process.

  It holds several characteristics and some of them are as follows:

- Software testing must begin with the business requirement document.
- Software testing means testing of both stable and dynamic activities of a software.
- Software testing is done to determine the operational coverage.
- Software testing helps to ward off the inconsistency between functional and non functional requirement.
- Software testing is a method to prevent failure and determining the reason of failure if occurs.

  Testing as well serves to evaluate the software quality in the termini of its capability for achieving correctness, reliability, usability, maintainability, reusability and testability. The main intent of software testing is to know the errors of the software in order to fork up the defect free software to our clients. Benefits of software testing are as follows:

1) Reduction in Defects
2) Product Quality Improvement

3) Enhance Software Developers Productivity

4) To heighten control and accountability.

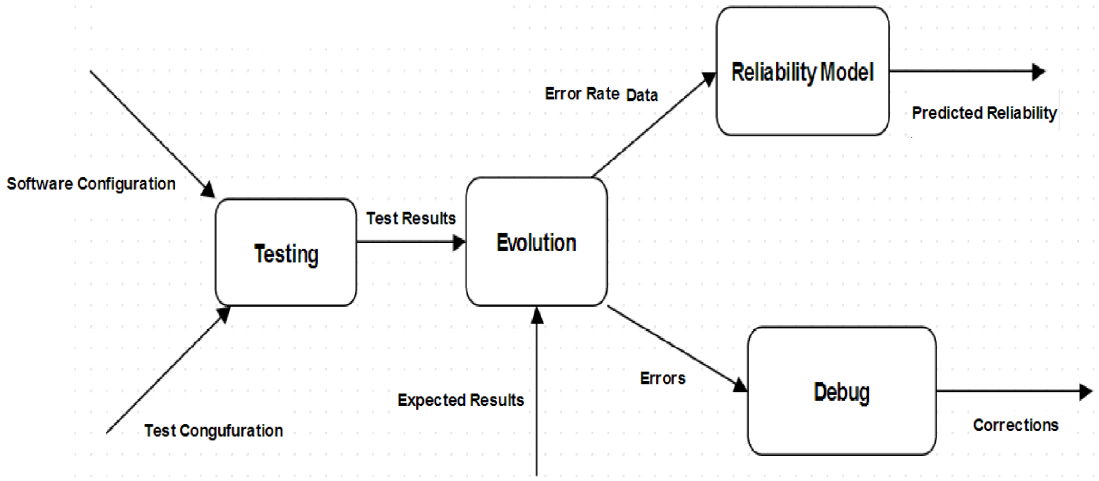

Figure 1.1: Testing Information Flow and Events [3]

Testing information flow as shown in figure 1.1 is supposed to be as a testing technique which specifies the scheme to select input test cases and analyze test results [3].

1.1 Why Software Testing is important ?

In recent time, computer applications have spread into every area of circle, for handling of various sophisticated applications. Testing of the right thing at the proper time, harmonizing to the right degree of detail in a most efficient way demonstrates that a software system functions accurately. In the software development life cycle (SDLC) the Testing assumes a substantial component, which serves to raise the quality, reliability and performance of the system with all checks what all capacity software expected to do and additionally also check that software is not practicing what it shouldn't get along.

In the today's business of competition just the quality product stays long-lasting immovably, and then to verify the produce the greatest quality

product the testing of software is a key element in SDLC. A significant number of these applications are of extremely extensive, complex and security critical. So, highly reliable software is vital. In other word, better quality software with high reliability is almost indispensable. Aside from the presence of numerous techniques for expanded reliability, software testing is an significant and regular approach took over. So, testing remains the most imperative role of quality assurance in the act of software evolution. The substantial software size is examined as a major challenge while creating a quality software. So quality assurance is a vital and significant issue for expansive scale software Development.

According to Miller [4] the goal and need of software testing is "affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances.

According to IEEE Computer Society testing is A verification method that applies a controlled set of conditions and stimuli for the purpose of finding errors [5].

Test results are the recorded verifications, which demonstrates that that prerequisites are met and could be repeated. The resulting information could be reviewed by all concerned for affirmation of capabilities [5].

1.2 UML Diagrams

The Object Management Group (OMG) specification states: " The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system." The UML offers a standard way to write a systems blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components . UML 2.0 mainly comprises of 14 diagrams. And these fourteen diagram are divided into t hree basic classification and they are as follows:

1) **Structure diagrams :** It is the most essential and generally used form of UML. It is used to show the static and stable aspect of the system.

2) **Behavior diagrams :**It is used to depicts the dynamic aspect and behavior of the system.

3) **Interaction diagrams :** It a subpart of behaviour diagram in which object interactions are emphasized.

1.3 Test Case Generation

Generally Manual software testing and test case generation, are labor intensive process. Automated testing essentially minimize the expenses of application development and maintenance. As shown by the definition, an imperative part of the test case is receiving the expected result or output.
Consequently, test case mainly consists of two entities:

1) Program Input data

2) Detail of accurate output from the set of input data.

For satisfying the arbitrary test requirements by test case generation is nontrivial issue. Several researchers have give their attention towards this process. And it concludes the various degrees of achievement as a result. Different strategies and different design documents are used to generate test cases automatically from the system which is to be tested. It might involve the design document as an input, then process it. After this test specification are generated depending upon the prespecified testing criteria termed as coverage criteria. Afterwards, the precise test data for each test is identified to organize test cases. In testing , execution of the test case is performed continuously for the peculiar set of inputs. And it helps to establish the trust of developers. Test case may be obtained from business requirement document.

It can also be derived from the source code. Test events are basically designed depends upon the origin code. And due to this, test case generation becomes complex for those who test at the cluster level. Test case generation using design documents gives additional benefit for designing the test cases in the initial stage

of the SDLC, so that test planning can be performed more effectively. Generation of test cases manually, consumes time and effort both. That is why test case design process should be either automatic or semi-automatic. Therefore, it is mostly required to design the test cases from the requirement document

1.4 Motivation

In recent years, usage of software is present almost in every field. And testing is a very essential part of any developing software system.

Nowadays, software is not allowed in the production by several companies until the testing is done properly. Unit testing is also an essential division of testing that provides quality to the software. For proper testing, we need to design the unit test cases that cover every part of the application.

Merely it is really complex to test the large software system. And if a system is large then there is need of design large number of test cases and it became very hard to achieve the proper coverage. For these kind of systems, it is unacceptable to design the test cases, manually since it consumes too much time and effort. Automated test case design can assist in cutting down the time and effort, and along with this it increases the reliability and quality of the software by increasing the test coverage. These issues motivate us to develop an approach that can generate the efficient unit test cases automatically from the UML state-chart diagram of the application. In the subsequent section, we have elaborated the major goal of our research thesis.

1.5 Problem Statement & Objective

Software testing is one of the time consuming and costly process in SDLC. Automation of this process helps to overpower this problem and it too shortens the human effort. Automation Testing would not be beneficial if we have to hold the testing until the SDLC stage is not completed. If any defect is identified at this stage, then from starting we have to verify the code and design document

to find the genuine cause of defect. So the resolution for this kind of issue is to begin the testing procedure from the initial stage, i.e. requirements specification phase to the last stage. We used the Model Bases Testing approach for test case generation to accomplish our objective.

The major objectives of our research work are as follows:

[1] To develop the back end of Collaborative Invention Mining.

[2] To generate Unit test cases for the back end Collaborative Invention Mining, using UML state chart diagrams.

[3] To minimize the developed Unit test cases.

[4] To compare the test cases generated by our approach with the test cases designed by the testing team of the TCS organization.

1.6 Thesis Layout

Rest of the thesis is organised as follows —

Chapter 2: Basic Concepts and Definitions: In this chapter, we discussed about the basic concepts and definitions related to model based testing and automatic test generation of test cases.

Chapter 3: Literature Review: In this chapter, we explore about the existing research work in the era of automatic test case generation usingUML state chart diagrams.

Chapter 4: Collaborative Invention Mining: In this chapter, we discuss about the overview of CIM application. Then we discuss about our contribution in CIM application development and working on that application.

Chapter 5: Unit Test Case Genration Using State Chart Diagram: In this chapter, we discussed about our proposed technique for automatic test case generation using a state chart diagram. We first discuss about a few basic concepts and definitions used in our methodology. Then we discussed about the working of our proposed approach using an example of CreateIST phase followed by a conclusion.

Chapter 6: Conclusion and Future Work: This chapter concludes the important contributions of our work. Finally, we discuss the possible future extension to our work.

Chapter 2

Basic Concepts and Definitions

In this chapter, we discuss about the terminology and basic concepts which is used in our research work. This chapter is organized as: In section 2.1, we have given the preliminary definition and concepts. Section 2.2 represents a brief overview of XMI. Overview of model based testing is discussed in Section 2.3 and in Section 2.4 we have discussed about the UML Overview. In Section 2.5 conclusion of this chapter is discussed.

2.1 Preliminary Definition and Concepts

2.1.1 Testing Techniques

Testing is an imperative method to verify and control software quality. It is nearly related to software evolution process. Software testing is a technique which ought to be performed within the entire process of development [6].There are following types of techniques that is performed during the development of the project:

[1] **Black Box Testing :** Black-box testing is a testing technique that analyze the utility of an application without peering into its inward structures or functioning.

This testing technique can be built singularly with respect to an analysis of the requirements (user documentation, specification), either functional or

non-functional, of a framework or segment without reference to its inward structure. It is also known as functional testing.

[2] **White Box Testing :** White box testing is a testing technique that analyze the inward structures or functioning of an application, rather than its functionality as in Black-box testing. In this form of testing internal perspective and programming abilities can be used to plan the test cases of the system. White box testing is the elaborated examination of internal logic and structure of the code. This testing technique is focused on an Investigation of the internal structure(design, code etc.) of a segment or system. It is also known as structural testing.

[3] **Gray Box Testing :** Gray Box Testing is a software testing method which is a combination of Black Box Testing method and White Box Testing method. In Gray Box Testing, partial knowledge about the inner structure is there. Internal data structure and algorithm is accessed in gray box testing for the purpose of planning the test cases, but testing is executed solely at the user, or black box stage.

This kind of testing is performed to identify the faults that occurred due to the unconventional systems design and implementation. Tester already has knowledge about the system's internal structure and testing information.

2.1.2 Test Process

There are a few abstraction levels in system development that came from requirement analysis to the usage in machine code. Testing could be addressed at all layers of abstraction. Test process basically consists of following activities as shown in Figure 2.1.

Test process dependably starts from Test Planning and finishes with the verification of test completion. Few activities can be rehashed (or in any event returned to) since various emphases is required prior criteria for completion that is characterized throughout the Test Planning activity. Some of the activities can be

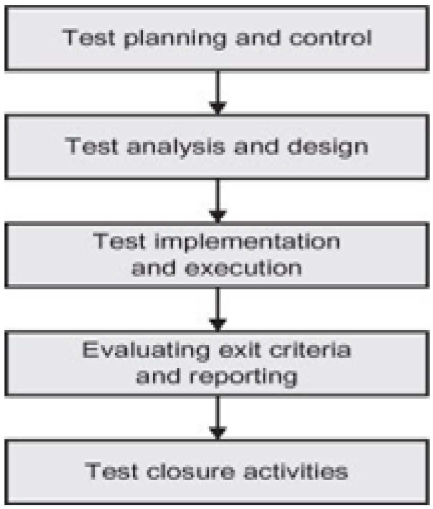

Figure 2.1: Test Process [6]

performed parallel and some sequentially. Throughout this cycle of activities, at the same time, the advancement of activities needs to be observed and controlled so we stay in accordance with the test plan.

2.1.3 Test Case

A Test Case will normally comprise of information such as requirements referenced from a design specification, a series of test step to follow, verification of test steps, prerequisites, events, outputs, expected outcomes and yield resolution.

A test case is the triplet (I,D,O) where I is the state of the system at which the test data is input, D is the test data input to the system, and O is the expected output of the system [8]. It is a set of inputs, execution conditions, and expected results that is created for a specific goal for example to act out a specific path of a program or to confirm consistence with a specific prerequisite [9]. A test case generally consists of input, an action and an expected result. A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement [9].

Test cases are the particular set of inputs that we apply and the certain steps

that we follow while testing a software. Test cases must practice each characteristic of the application to keep defects from being released. The procedure of making test cases can additionally serve to discover issues in the requirements or intent of an application. The objective of utilizing test cases is finding out the errors in a program.

2.1.4 Test Adequacy Criterion

Test adequacy criterion endows the knowledge to the tester regarding the test suite development. It is used as a method for fixing up the testing activity. It assist as a stopping condition as well as it quantifies the advancement towards the objective [10]

It is well known the way that testing produces the software error free and increments the reliability. Even so it is not known when to terminate the testing procedure or what constitutes the adequacy of a test. Test Adequacy criteria are only a crucial and essential predicate, which indicates the adequacy.

There are several approaches for test adequacy criteria such as activity coverage, branch coverage, path coverage, transition coverage. But here we are using only transition and path coverage. A test adequacy criterion assist in characterizing test goals or purpose that are to be carried out while doing a particular software testing.

2.2 XMI Overview

XMI is short form for XML Metadata Interchange. It is a standard of Object Management Group(OMG) which is used for exchanging metadata information with the help of Extensible Markup Language. It helps developers using UML with various languages and development tools so that their data model gets exchanged with each other. XMI could be utilized to interchange information about data warehouses. XMI depends upon these three industry standards or recommendations:

- **Extensible Markup Language (XML):** This is a standard by World Wide Consortium.
- **Unified Modeling Language (UML):** This is a standard by OMG.
- **Meta Object Facility (MOF):** This is also standard by OMG which is used for metamodeling & metadata repository.

2.3 Overview of Model Based Testing

Model based testing might be summed up in one sentence, as it is basically a technique for automatic creation of test cases from particular software application. The key advantage with this strategy is that the test generation can methodically infer all combinations of tests associated with the requirements demonstrated in the model to automate both the test design and test execution process [11]. In other words, MBT is a technology that automates creation of test cases in specific areas [12]. Model based testing refers to the process and techniques for the automatic derivation of abstract test cases from the abstract formal models, the generation of concrete tests from abstract test cases, and the manual or automated eexution of the resulting concrete test cases [13]. MBT generates the test process automatically by utilizing the models of system requirements and behaviors. Software model is useful for refining the ambiguous defined requirement [11]. A typical deployment of MBT in industry goes through the four stages as shown in Figure 2.1.

The model is designed generally by using information, specification or requirements. And while constructing the model, information regarding the requirements become more clear and it also leads to identify the missing information. Many researches show that the model is created by testers for testing the system. MBT is kind of testing that depends upon the external behavior of models which can code the behavior of the system. A succession of input is provided as input to the SUT, that response with a series of output or outcomes. In the last few decades, several testing approaches have been developed to automate

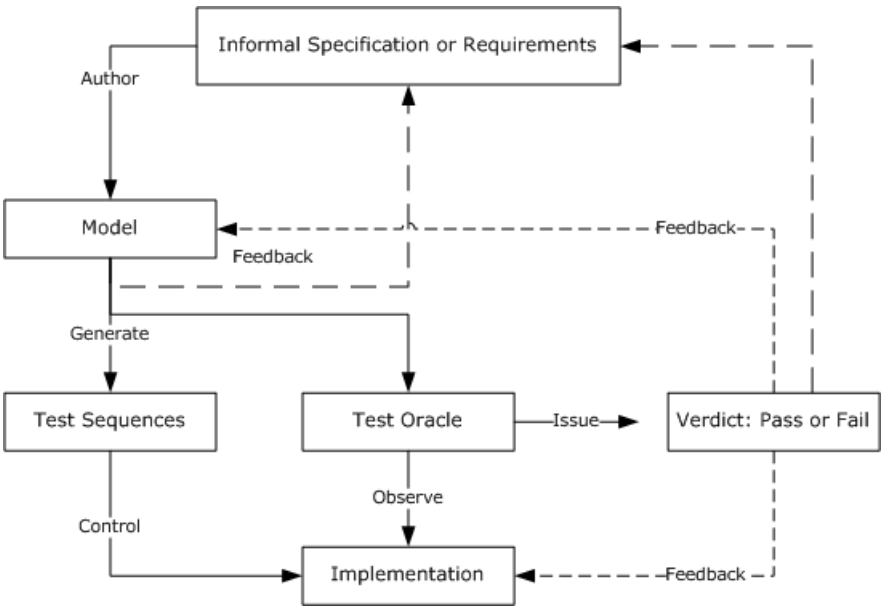

Figure 2.2: Model Based Testing Process

the testing procedure that increases the effectiveness of testing.

2.4 Overview of UML Diagrams

UML was developed by James Rumbaugh, Ivar Jacobson and Grady Booch. UML is also considered as the visual modeling language. It was adopted as a de facto standard for modeling software systems by OMG in 1997 [8]. It is widely acknowledge and utilized by the industry. UML models become popular in industry and academic level, and it is also getting the attention of researchers for the test case generation in the context of Model based testing. It is one of the general purpose modeling language in the area of an object oriented software system. UML comprises of standardized graphic notation which is utilized for making a visual model of the software system. Models are the essential part of the software project, whether the software is large or small. UML model helps to interpret the commercial enterprise model, requirement specification and design of the system. UML consists of several model elements which can be utilized to show the different components of the system.

After its debut in the late 90s, UML has been retooled several times. UML 2.0 is the latest release, that has several novel characteristics as compared to UML 1.X.In our research study we have used the UML 2.0. UML is used to create models but they are generally incomplete and ambiguous. And this happened because UML has a semi- formal nature. Then also UML model is a rich source of information to design the test cases. There are two essential cause of using UML models in the software system. Firstly, this model endows the blueprint of the application to the developer so that they come to know what they have to build and to the project manager for the cost estimation of the project. Secondly, it assists the non expert users to better interpret the software system.

2.5 Summary

In this chapter we have discussed about the basic concepts for better understanding our research work. Firstly, we have described about some basic related to testing and automatic test case generation approach. Then we have discussed about the overview of XMI. And after this we have discussed about the model based testing. We present our main attention towards the model based testing technique that assists us in modeling and automatic generation unit test cases. In the subsequent section we have discussed about the UML diagram which basic building block of our research works.

Chapter 3

Literature Review

In this section,we present about a review of the related work that has been performed in the area of UML state chart testing for an automatic test case generation. Various researchers have worked in the field of test case generation using UML Diagrams. Among UML diagrams, state chart diagram also received attention from the researchers to generate test cases.

3.1 Automatic Test Case Generation From UML State Chart Diagram

A technique is developed by Supaporn Kansomkeat and Wanchai Rivepiboon [14] for the generation of test cases using UML state chart diagram. They attempt to get the testing technique that can partly resolve the testing process. In their approach they first transformed the state chart diagram into a model of states that is called as testing flow graph. From this TFG, they define the flow of event sequences which is termed as test sequences. And finally they get the test cases from TFG by using the criteria of testing that is the state coverage and transitions.

Kim *et al.* [15] introduced a technique that generates the test cases that can do testing up to the class level by using UML state chart diagram. They convert the state chart diagram into the extended finite state machine to obtain the test

cases. After this, data flow is determined by changing the EFSM within flow graphs, on which the traditional data flow analysis techniques are enforced. As a consequence, in EFSM the structure of the states is flattened in hierarchical and concurrent manner. And in that broadcasting communication are also eliminated.

A novel approach is suggested by Kosmatov *et al.* [16] for automatic generation of the test cases from formal models. They consider a certain set of preconditions to test and the input domain. This methodology primarily performs a boundary coverage analysis on a set of variables as input values and then generate the test cases by using cost minimization and maximization functions.

Gnesi *et al.* [17] developed a mathematical model for conformance testing and that can automatically generate test cases from UML state chart diagram. They introduced a formal conformance testing relation which is used for IOLTSs. IOLTSs stands for input-enabled transition systems with transitions labeled by input/output-pairs and that endow a suitable semantic model. This model is basically for a behavioral subset of statecharts. They also suggested an algorithm that automatically generates a test suite for UMLSCs.

Systa *et al.* [18] proposed compression algorithm for statechart diagram that is used for preserving information. Their algorithm is used for converting a simple state chart diagram into more compact form. It scales down the count of states and transition and preserves semantics of state chart diagram. It also finds out the identical reaction to certain events and utilized that information to rebuild the diagram. The algorithm proposed could be enforced to manually construct as well as to automatically generate the statechart diagram.

Swain *et al.* [19] proposed a novel testing methodology that is focused around the state and activity models of the system. They form a diagrammatic representation which is termed as State-Activity diagram (SAD). And this diagram is used for test cases generation to obtain the coverage of SADs. This technique can be applied to identify seeded integration testing faults.

A method proposed by Samuel *et al.* [20] is to generate the test sequence automatically by using UML state diagrams. In their technique, they first traversed the state machine diagram and select the conditional predicates on each

transition. After the conditional predicates are selected, then selected predicates are transformed. Finally the test sequences are generated by applying the function minimization technique for both testing class and cluster level behaviors.

A method for automatic generation of test cases from UML State chart diagram is proposed by Swain *et al.* [21]. In this method, they transformed the state chart diagram to state chart graph. And so by applying DFS in the state chart graph, predicates are selected. Relation expression transforms the conditional predicates to predicate function. And these predicate functions are minimized. Then by using the minimization function technique, test cases are generated.

3.2 Summary

In this chapter, we discussed about the different strategies that are involved in automatic test case generation. We learned about various approaches related to test case generation using UML state chart diagram. This chapter gives us a brief introduction about the existing work that has been done on model based testing approach using UML state chart diagram. Various researchers have propose different approaches for test case generation using UML diagram, but many of them are semi automated and also inadequate to the complex organization. Test case generation is very essential with respect to model based testing.

Chapter 4

Back End Development for Collaborative Invention Mining

In this chapter we have discussed about the collaborative invention mining application. Firstly we have given the introduction about it. Then we have discussed about its objectives and after that we discussed about our contribution in the development of the application. We as well discuss about its working and the algorithm used for calculating CIM score.

4.1 Introduction

Invention mining is the technique of increasing the strength of an idea by collaborative deliberation. The main goal of this procedure is to evolve the capacity in the direction of widening, lengthen and deepens the idea, so that it gets mature and later considered it as a "patentable invention". This process assists in emerging of new inventors. And it also improves their skill for new innovations and setting up an inventor community for an enterprise.

Mohanty *et al.* [22] from Tata Consultancy services (TCS) had registered a patent named as "Collaborative system & method to mine inventions". Based on this, an application is being developed at TCS, which is named as "Collaborative Invention Mining (CIM)" by Dr. S. K. Mohanty. Collaborative Invention Mining is

the technique of converting a concept or an idea through a collaborative interaction into a mature state so that the idea becomes patentable in a selected set of jurisdiction. . The objective of such a process is to impart various dimensional rigor across category dimension, area coverage and business characteristic to widen, lengthen and deepens an idea, therefore make it as a feasible invention that can be patented.

An idea/concept present in an enterprise must be resilient and aligned to its business footprint to be considered as valuable asset. Hence, the objective of invention mining process should derive and strengthen an idea collaboratively to a matured state where it can be promoted as Patentable invention and protected in a select set of jurisdiction.

4.2 Objectives of CIM Application

The main objective for developing this application are as follows:

[1] To automate the process of invention mining through collaborative deliberation

[2] To set up an Inventors community in an organization

[3] To provide a platform where people can get together and share their knowledge for an enterprise.

[4] To represent it directly with the TCS Valuation Module for the predictive estimation of Non-linear in revenue.

[5] Multifaceted Usage of the Template and Process for Portfolio Analysis

[6] Multi-view Capability of the Framework: Stakeholders across all areas of IP

4.3 CIM Model Overview

The model that we have developed , mainly consists of three essential dimensions , which helps in increasing the strength of an idea through collaborative deliberation

and they are as follows:

[1] **Category :** This dimension extensively classifies the scope of an idea into technological progress, conceptual or commercial implementation classification.

[2] **Area :** This dimension classifies the idea lengthen into several competency building elements.

[3] **Characteristic :** This dimension consists time subsistence aspects to further increase the idea's competency so that it stays for a long time with its sustain value.

For each of these Category, Area and Characteristic dimensions in our model, a set of attributes has further been identified and contextually defined.

For Category dimension, we have defined Novelty, Inventive Step & Utility (NIU) as the attributes to evaluate the patentability scopes of an idea. The contextual definitions of NIU are as follows:

- **Novelty :** It is defined as the scope of an idea that is patentable, that evaluates the idea to be new and knowledge about that idea should not be present in the public domain.

- **Inventive Step :** It can be characterized as an idea that can be patented in the way of technical progress in comparison to the existing information or (accordingly) making investment importance which will make the idea not only an evident development or simple re-plan of known existing segments to an individual.

- **Utility :** It is specified as a patentable scope of an idea concept to evaluate its ability towards commercial implementation on a bulk scale.

The area dimension is again categorized into four categories named as Process, Technology, Measurement and System (PTMS) to create capacity development of an idea.

The Characteristic dimension is again categorized into four categories identified as Efficient, Adaptable, Agile and Anticipative (EA3) so that the livelihood time of an idea can be evaluated.

4.3.1 Validate and Iterate (V & R) Dimension

Each of the idea elements in Category, Area and Characteristic dimensions further get validated iteratively, with respect to Enterprise Knowledge Data and other external Patent and non-Patent literature Database. We have used image mapping of prior art contents for an idea element of a, to visualize the existing prior art documents for a respective content or cell in the IDT.

Following is the overview diagram of the Model covering the NIU, PTMS, EA3 and V &R dimensions, This Figure 4.1 demonstrate how an idea is matured through collaborative deliberation.

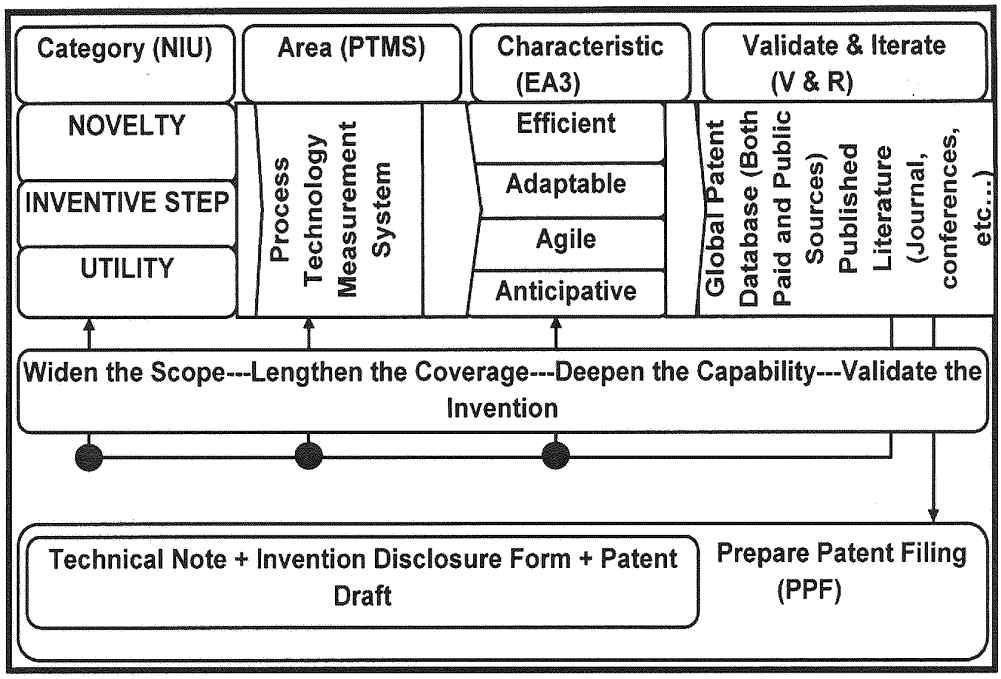

Figure 4.1: Overview of Invention Mining Architecture

4.3.2 Idea Detailing Tree Schema(IDT)

Idea detailing process is complicated due to intricate connection to attributes of manifold dimensions, the growing maturation of an idea is becoming probably hard to imagine . The schematic diagram of IDT shows the diagrammatic view about how an idea can grow and get matured.

The schema of the Idea Detailing Tree has been depicted in Figure 4.2. Tree

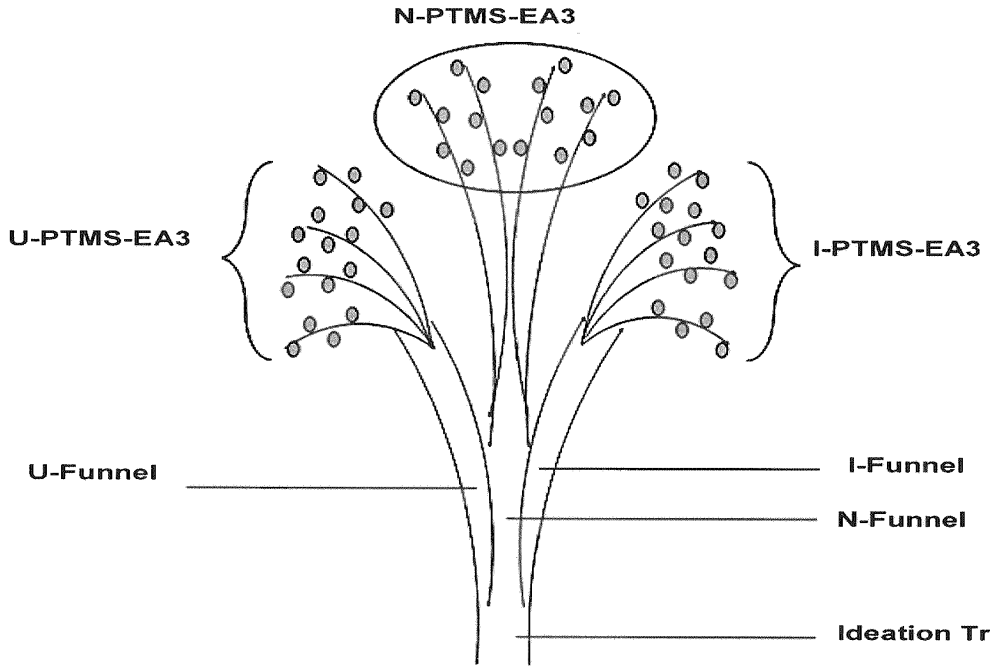

Figure 4.2: Idea Detialing Tree Schema

base is symbolized as funnel of an idea, which is the entrance CIM workflow. The idea from its development stage may first go through Novelty(N) Funnel, Inventive step(I) Funnel & Utility(U) Funnel that signify the vast scope of an idea in NIU patentable criteria.

Along with progress towards a higher degree of maturity (Form Phase), every idea element is further being classified into PTMS category, with the target of constructing competency within the PTMS area. For making an idea to be time sustainable there is need of increasing the subsistence feature of an idea to make it EA3.

These EA3 could be imagined as a fruit of every PTMS region coverage, which come out from the branch of every NIU category funnel. Thus, in all, there are 48 such fruits that an idea can produce.

In an ideal scenario, it is desirable that all 48 fruits be matured for an idea so that it can mature into a strong iconic patent. However, a realized throughput of the Collaborative Inventive Mining method is to have sufficient claimable elements that have over ridden prior art, then each idea can be termed to have shown good yield.In an actual scenario, it is not necessary that we get all 48 fruits as matured, but we are able to obtain that much which is sufficient to obtain a patent.

4.4 Contribution in Application Development

In this, we firstly shows the procedure of CIM score calculation in which our participation was there during development. This algorithm is developed at back end using Java technology which is used for handling database from the front end . It is used for calculating the CIM score to identify that whether the idea is patentable or not, Then we discuss about the back end development and working on CIM application.

4.4.1 Algorithm used in CIM Application

In this section we discussed about the algorithm that is used for calculating CIM score . And based on CIM score it is decided whether the newly created Idea is ready for filing as a patent or not. Inventor views CIM score and CIM clusters to strengthen his/ her invention. CIM score and CIM clusters help to decide if additional iterations are required to further strengthen the invention.

Steps involved in CIM Score calculation is given below:

Step 1: Calculate the total number of statements in each of N, I and U.

Step 2: Based on each N,I and U calculate the total number of statements in each of Process P,Technology T,Measurement M and System S.

Step 3: Depending upon each P,T,M and S calculate the total number of statements in Efficient E,Adaptable A,Agile A and Anticipation

Step 4: Then calculate V by using weighted values of P,T,M ,S and E,A,A,A.

Step 5: After calculating V, calculate the individual score of `N_Score`,`N_Score`,`I_Score I`and `U_Score`.

Step 6: Finally calculate CIM Score i.e.(`N_Score`+`N_Score`+`I_Score I`+`U_score`)

For calculating the CIM score we used a Idea score card template. The Figure 4.3 below shows the score card template.

Invention Mining			Idea Title: abc					Invention Scorecard
Category(NIU)		Area (PTMS) Description & Weight	Characteristic (EA3) Description & Weight					Category Score
			Desc.	Efficient	Adaptable	Agile	Anticipative	
Desc.	Wt.	Desc.	PTMS:10, EA3:10	2	2	2	4	N_Score
Novelty	4	Process	3 Max.	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		Technology	3 Value	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		Measurement	2 100	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		System	2 (10*10)	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
Desc.	Wt.	Desc.	PTMS:05, EA3:20	4	4	4	8	I_Score
Inventive Step	3	Process	1 Max.	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		Technology	1 Value	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		Measurement	1 100	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		System	2 (05*20)	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
Desc.	Wt.	Desc.	PTMS:20, EA3:05	2	1	1	1	U_Score
Utility	3	Process	4 Max.	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		Technology	4 Value	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		Measurement	8 100	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
		System	4 (20*05)	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	V+V/2+V/4	
Aggregate Invention Scorecard (out of 1000)=4*N_Score + 3*I_Score + 3*U_Score								XXX
Meta Metrics:# of cells populated=?,# of statements populated=?, Category cluster(s)=?, Area cluster(s)=?, Characteristic cluster(s)=?								

Figure 4.3: Idea Scorecard Template

Working of CIM Scoring Algorithm:

Let us calculate CIM score when there are:

3 ideation statements in Novelty+Process+Efficient cell

2 ideation statements in InventiveStep+Technology+Adaptable cell

1 ideation statement in Utility+Measurement+Agile cell

Step 1: Calculate the score of each populated cell for Area and Characteristic combination.

Since there are 3 ideation statements in Novelty+Process+Efficient cell, the score of this cell is: V+V/2+V/4 where V for this cell=3*2 = 6.

So the score of this Novelty+Process+Efficient = 6 + 6/2 + 6/4 = 11

The score of this InventiveStep+Technology+Adaptable = 4 + 4/2 = 6

There is 1 ideation statement in Utility+Measurement+Agile cell, the score of this cell is: V where V for this cell =8*1 = 8.

So the score of this Utility+Measurement+Agile = 8

Step 2: Multiply individual cell score with the weight age given for each category.

Weight age for Novelty = 4. So Total Novelty Cell Score * Weight age = 11 * 4 = 44

Similarly Total Inventive Step Cell Score * Weight age = 3 * 6 = 18

Total Utility Step Cell Score * Weight age = 8 * 3 = 24

Step 3: Add the results of Step 2 to get CIM Score:

Total CIM Score = 44 + 18+ 24 = 86

Invention is said to be patentable if CIM Score > 450 CIM score helps to strengthen the inventors invention. If the CIM score > 450 then the idea is considered for filing a patent. But if it less than 450 then further iteration is required to achieve the good CIM score so that idea can become patentable.

4.4.2 Working and Implementation of Application

CIM is an application which provides a platform to the TCS employees to give an idea by collaborating together, so that the idea becomes patentable. In CIM application, we have used Adobe Flex Builder in the front end development and we have used the Java Eclipse IDE for back end development along with Postgre SQL which is used for handling the databases. We were involved in the development of the application and also in writing Unit test cases. We were also involved in writing unit test cases.

The CIM application mainly consists of following phases for strengthening the idea :

- IST Phase
- Parking lot Phase
- Storm Pad Phase
- Form Pad Phase
- Norm Pad Phase
- Compose Pad Phase

Summarized work flow of CIM application is as shown: IST Phase–> Parking Lot Phase –> Storm Phase Pad –> Form Pad Phase –> Norm Pad Phase –> Compose Pad Phase Now we give the detailed working of each phase :

- **IST Phase:** Firstly, user login into CIM application lands on the home page of the application. Then the user can see the projects in which he/she is involved. On click of the "My Ideas", it would open the screen with the idea details. This screen is also named as Idea Sharing Template (IST) phase. In this phase user provides the details related to ideas such as Title, keyword etc. User may add more participants, but he must have to add Moderator before submitting the IST. Finally user submits the IST for reviewing. The screenshot for IST Phase is shown in Figure 4.4. After submitting or saving of IST the details provided by user is handled by Java code and is stored in the database in backend.

- **Parking lot Phase:** After the IST is approved by the moderator, with the help of base list idea element will form in the parking lot phase. User may add a new Idea element even in the parking lot phase. After adding of the idea list element , PAA may add prior art document that updates the document base list of parking lot phase. The screenshot for parking lot phase is shown in Figure 4.5. User can view the prior art document any

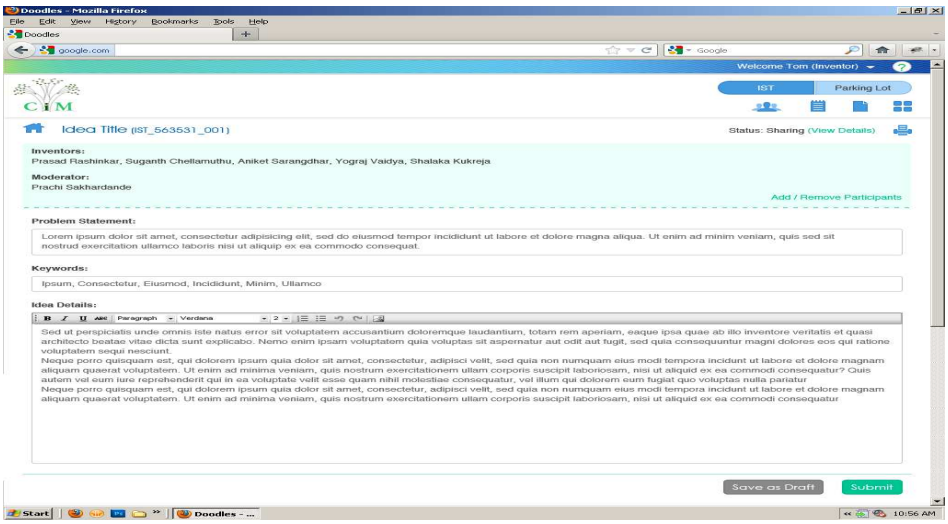

Figure 4.4: CIM IST Phase

time whenever he wants by clicking on documents that is present in the right corner. After adding prior documents or new ideas user can finally submit the IST for review.

- **Storm Pad Phase**: After the IST is approved by moderators, the idea list elements will be present in the parking lot phase. The user has to drag and drop the idea elements from Parking Lot to Norm pad phase based upon its category in N, I and U. Here also User may add a new Idea list in storm pad phase. After adding of the idea list, PAA may add prior art document that updates the document base list of parking lot. User can view the prior art document any time whenever he wants by clicking on documents that is present in the right corner. After categorizing an idea list into N,I and U user can finally submit the IST for review. The screenshot for storm pad phase is present in Figure 4.6.

- **Form Pad Phase**: The behavior of form phase is almost similar to that of storm pad phase. Here idea elements are dragged from the storm pad phase that is NIU category and drop in any of the Process P or Technology T or Measurement M or System S category. After categorizing the idea list into P,T,M and S user can finally submit the IST for review to moderator. The

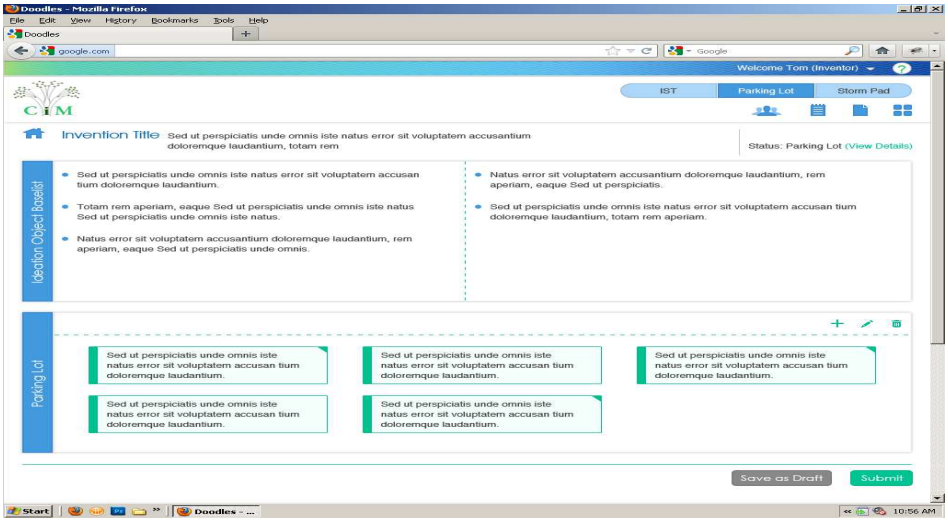

Figure 4.5: CIM Parking Lot Phase

screenshot for storm pad phase is shown in Figure 4.7.

- **Norm Pad Phase:** The behavior of Norm pad phase is almost similar to that of storm phase and Form phase. Here idea elements are dragged from Form pad phase that is PTMS category and drop in any of the Efficient E or Adaptable A or Agile A or Anticipative A category. After categorizing the idea list into EA3 user can finally submit the IST for review to moderator. In this phase user can switch to matrix view and he can also see the CIM cluster. The screenshot for norm pad matrix view is shown in Figure 4.8.

- **Compose Pad Phase:** In compose pad phase user can drag and drop the idea elements into independent, dependent or second dependent claim based on the Inventor view. User can see the claim tree is being forming and the claim document is generated after the completion of the process. The screenshot for storm pad phase is present in Figure 4.9.

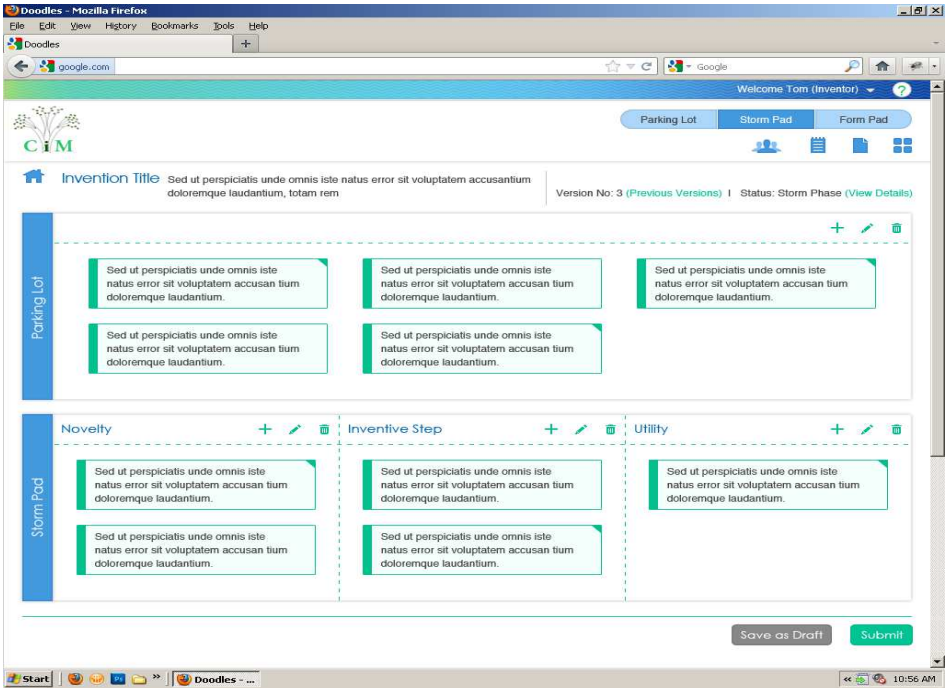

Figure 4.6: CIM Storm Pad Phase

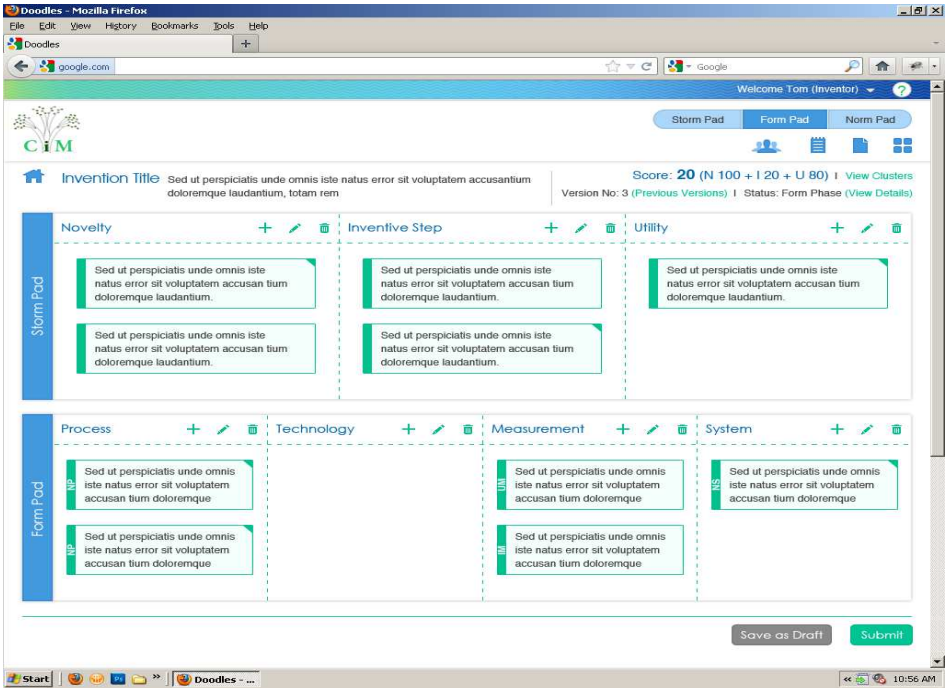

Figure 4.7: CIM Form Pad Phase

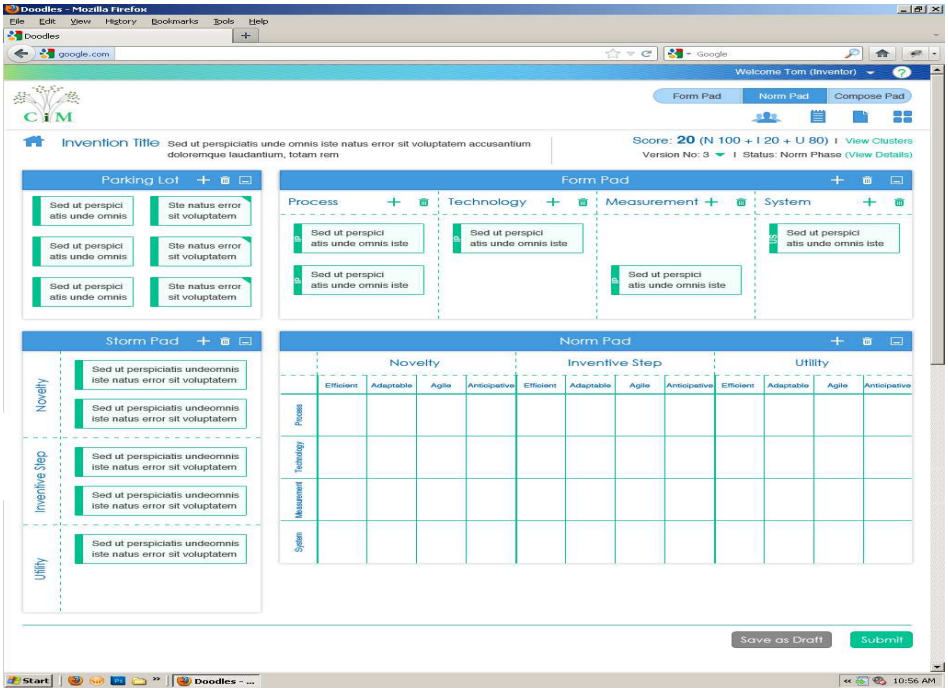

Figure 4.8: CIM Norm Pad- Matrix View

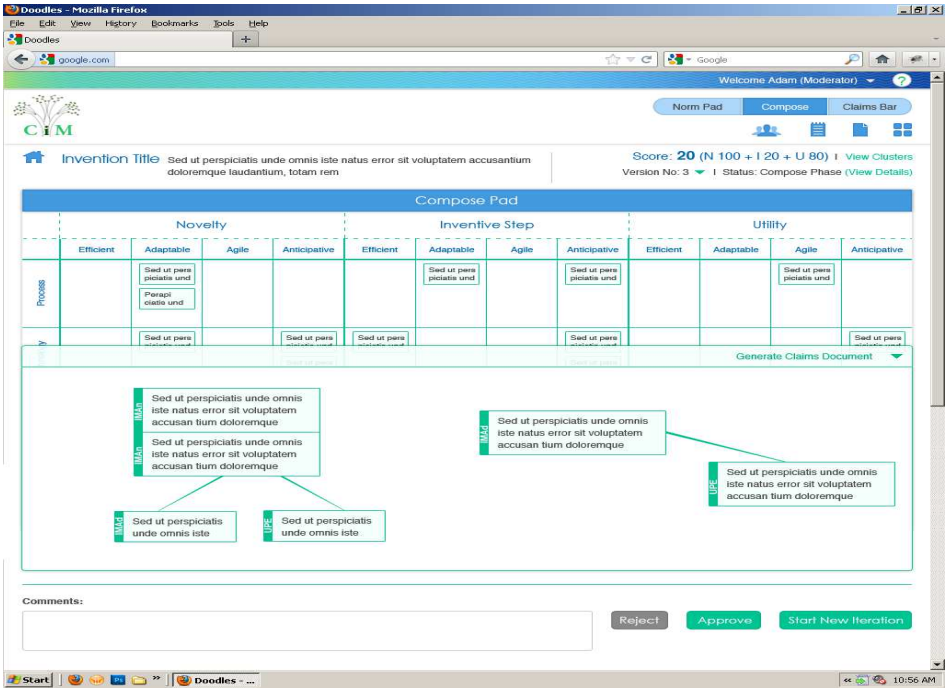

Figure 4.9: CIM Compose Pad Phase

Chapter 5

Unit Test Case Generation Using State Chart Diagram

In this chapter, we use a state chart diagram as the design specification and proposed an approach to automatically generate test cases using a state chart diagram. In this chapter firstly we discuss about the state chart diagram basics that we have used in our approach. And then we have proposed an approach and show the working of our approach by taking an example. And later we compared the test cases generated from our approach with test cases designed in industry.

5.1 Relevant UML Diagrams

UML consists of fourteen different types of diagrams, but here we just discuss about State Chart Diagram. In the subsequent subsection, we elaborate about the basic concepts and terms that are related with this chapter.

5.1.1 State Chart Diagram:

UML offers a various diagram to depict a specific view of software systems. In other words we can say that, it is utilized to indicate an extensive view of a system. And the classification of these diagrams is based upon the description view of the system whether structural or behavioural . State chart diagrams are one of the

behavioral diagrams that depict the action sequences between the objects that are regarded in the flow of control at the time of implementation. It describes the possible state that a model element may consider the transition that are allowed from every state, the events due to which the transition may occur and the actions that happened as a reaction to events. States ,Transitions and events are the most essential part of state chart diagram. State of an object is generally identified by the value that few variables(attributes) of an object may assume. State of an object is generally identified by the value that few variables (properties) of an object may take. Basically, an object has to remain in the same state until an event cause it changes to some other state.

Before continuing to our detailed approach of unit test case generation using UML state chart diagram, we should ,describe the basic terminologies related to state chart diagram.

Definition 5.1.1. State Chart Diagram: A state chart diagram is most similar to directed graph, which comprises of state as vertices and transition as directed edges. And these edges are utilized to interconnect the two states. It catches the different states that an object undergoes. It models how an object changes its states in its entire span of a life. It is useful in modelling the object's dynamic behaviour [23].

Following statement gives the summarized view of the state chart diagram Usually, an object remains in its current state. The object transit from one state to another by taking an action, when any event takes place.

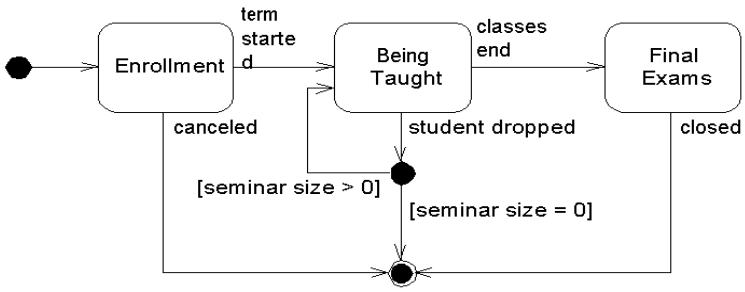

Figure 5.1: State Chart Diagram showing states & events

Regions: It is an orthogonal portion of the state machine. It comprises of states and machine. A state machine utilized dashed line so that it is split up into regions.

State: A state is presented in any system as a state variable (attribute), which is shown by discrete data. It refers to specific stage of an object in a behavioral manner. A state chart is mainly comprised of a fix number of states. A state is an abstraction of attributes and associations. For instance, the province of a mobile phone could be ringing or is used for receiving or calling. Every object can generally be in a single state at a time and moves through different states in its life. A state is represented in a rectangular form having rounded corners. It is subdivided into a number of compartments that are distinct from each other by horizontal lines. Different compartment of the state is given below:

Name Compartment: It is an optional compartment that contains the state's name in the form of a string.

Internal Activities compartment : It holds the index of internal actions or state activities that will perform until the element is present in the state.

Internal Transition Compartment: It holds the index of internal transition. An internal transition is a transition that occurs without altering its state.

An Object possibly may have the following types of states:

Initial State: : This is demonstrated as a filled black circle or small black disk, which is tagged with a name . Only one initial state is present in the state chart diagram. A transition that leads from the initial event represent the leads , that an entity can move into whenever it is created or initialized.

Final State : The final state is demonstrated as a dot or small black disk inside a large circle. It may also be tagged with a name. The final state shown that an entity is arrived, where it extinct or quits giving response to events. It is a special form of state that signifies the completion of the enclosed area. More than one final state can be there in the state chart diagram.

Activity State : It shows the duration of internal processing, that is performed by an object . In general, it is represented in the form of normal state which consist of an activity. For instance, when customer's enter an amount in ATM

and after that when the process of entering amount is completed, then the activity state becomes active, in case of ATM machine whether it is capable of dispensing cash to complete the transaction.

Event : An event takes place when an input is given to the state court. When an event occurs, it results some changes to the system. In any specific state, associated transition to a new state occur because of some events. On the other hand, some events do not cause any transition. It occurs at a particular point of time. It Both error condition and normal condition are included in the events . For Example, while modelling an ATM System, when the cash is dispensed by the machine, then picking up the cash is an event. Various kinds of events are there in the state chart diagram.The most commonly used events are Signal event,Call event,Timing event and Change event.

Action : An Action is related with a specific state and event. It includes a transition to a new state. It also describes the reaction of an object in response to the event. An action is also defined as the activity of an entity which is started by an event.

Transition and Guard Condition : The transition is an instant transformation from state to another.

In other words, it is a relationship between two states that show an eventual change of state from one to another. For example, when a telephone receiver is picked up, the telephone transition changes from idle state to dial tone state. At that time change of state is occurring, it implies a transition is fired by an event. As a response to an event, transition occurs from a one state (current state) to another state. The transition is represented by an arrow from the origin to the destination state. Generally, the name of event, due to which the transition occurs is written on the position of the pointer.

A guard transition is triggered when an event takes place. But only at that time when the guard condition becomes true. It is different from a change event. It is checked only once, when an event takes place The guard-constraint is in the form of boolean expression, which should be true, then only the transition would take place. The transition is activated only if the condition evaluates to true. The

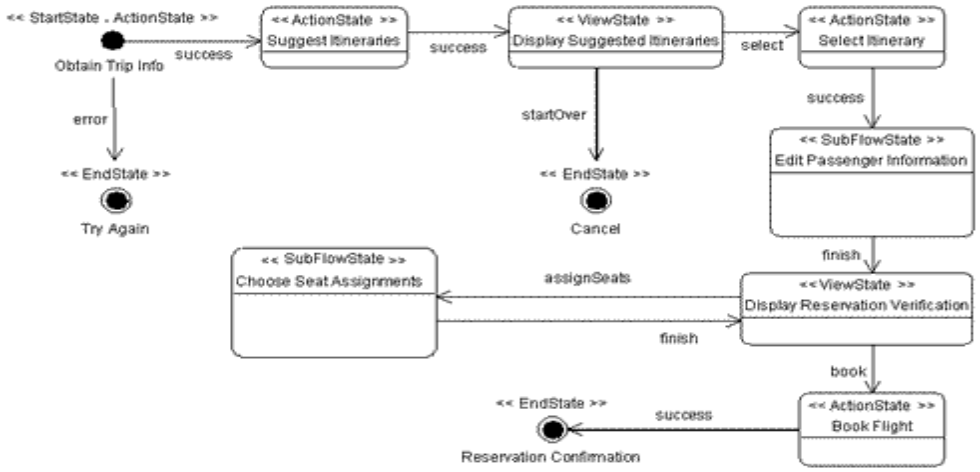

Figure 5.2: State Chart Diagarm of Flight Object showing Transitions and Guard Conditions

general syntax for the transition is given below:

trigger[guard-constraint] / activity-expression

5.2 Test Coverage Criterion for State Chart

5.2.1 State Coverage Criteria

Every state nodes in the state chart, graph should be visited at least once for the efficient generation of test cases. It is a test adequacy criterion in which testing of the program is required to be done so that output of the program is verified [27]. It is the ratio between the number of covered states to that of the total number of states present in the state chart graph [28].

5.2.2 Transition Path Coverage

Let us suppose a Graph G and a test set T . T is said to obtain the transition path coverage if every transition path in the graph G is visited at least once [24]. It covers all the different path by means of transition for comprehensive test generation .

Transition coverage = (Number of Transitions Covered)/(Total number of transitions in the state chart graph).

5.2.3 Transition Pair Coverage

There is a need of covering every pair of adjacent transition at least once in any of the test cases. So it incorporates overall transition coverage. The test cases generated by the transition-pair coverage is more as compared to transition coverage criterion [25]. For every pair of adjacent transitions Ai: Aj and Aj: Ak, TS must hold a test that navigates every pair of transition in the sequence [26].

5.2.4 Boundary-testing criterion

Testers have often inspected that mostly domain boundaries are affected by faults, and then it is required to examine the boundaries carefully. This criterion is a standard for ensuring that the boundaries are sufficiently tested. This testing is required to apply whenever the input test data field is split into two fields depending upon the decision. Rather than generating various test data value that obtains transition path coverage, we particularly tested the boundary . Boundary testing criterion assists in decrease the count of test cases. And subsequently very high test coverage is achieved.

5.3 Intermediate Graph Reprsentation

For the generation of test cases, firstly we have to transform the state chart diagram into intermediate graph. State chart diagram shows the behavioral view of the design phase. Therefore, we suggest a methodology to build an intermediate graph named as a UML state chart graph (UMLSG) and then generate the test sequences using that graph. Now in this section, we introduce a few basic concepts, notations and terminologies related to graph.

Definition 5.3.1. State Graph: A state chart graph G= (V, E) is a diagram that could be perceived as a graph, where V denotes the set of nodes and E

denotes the set of Edges. And it is named as UMLSC(UML state chart graph). In G, vertices indicate the states and edges indicates the transition. A state chart graph G= (V, E) is a diagram that could be perceived as a graph, where V denotes the set of vertices and E denotes the set of Edges. And it is named as UMLSC(UML state chart graph). In G, vertices indicate the states and edges indicates the transition. Beyond any scope of losing, we consider a unique vertex which resembles as the initial state. And initial state is shown as the origin of the tree.

Definition 5.3.2. Sub Path : A sub path from vertices vi to vk is a series of vertices vi,vi+1,....vk, in which there is an edge between every pair of vertices (vi+j,vi+j+1) in G.

Definition 5.3.3. Initial Path : Let us consider a path S of the graph. Initial state from where the sub path S begins is considered as the initial path of S.

Definition 5.3.4. Transition path : Transition path in the graph is the sequence of transition that begins from initial state and end at final state.

Definition 5.3.5. Path Domain : It is the set of overall input data values, for which the path S is visited and satisfied the path condition (i.e.it must evaluate to true).

5.4 Our Proposed approach to Generate Unit Test Cases

In this section , we describe our approach to generate the test cases automatically from UML state chart diagram. Our approach for generating the test cases from UML state chart is shown in Figure 5.3 as a schematic representation.
The steps involved in our proposed methodology for generating the cases are as follows:

Step1: Construct the state chart diagram of an application which is to be tested in IBM rational software architecture.

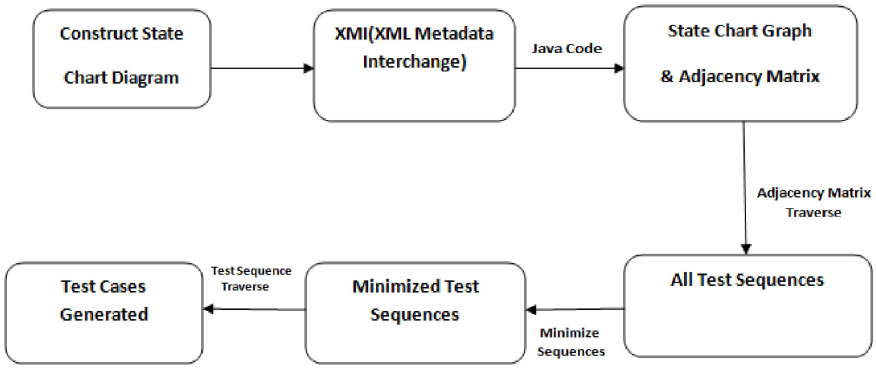

Figure 5.3: Schematic Representation of Proposed Approach

Step2: Then, save and export the XML metadata interchange (XMI).

Step3: After that, parse the XML Metadata Interchange (XMI) by using the XMI parser to generate the adjacency matrix of the graph.

Step4: Subsequently, transform the UML state chart diagram into a graph and name it as UML State chart Graph (USG).

Step5: Then, traverse the graph using adjacency matrix to obtain all the valid test paths.

Step6: Finally, test sequences are generated with the help of each traversed path.

Step7: Reduce test sequences by using the Node coverage criteria and generate the test cases.

The steps involved in our proposed approach are discussed below :

5.4.1 Construct State chart diagram for the application

Firstly the state chart diagram for the application that we have to test is created. State chart diagram is a model which is used to show the dynamic behavior of the system. Statechart diagram generally gives the summarized information of

the system. It is one of the 14 UML diagrams and is used to represent dynamic nature of the system. State chart diagram is similar to that of the graph where state indicates the node and the transition that interconnect the states indicates the edges. It is used to represent the different states of an object. It is also used to depict the objects life cycle.

5.4.2 Save model in XMI Format

XMI file parsed by using Document Object Model (DOM) API. A UML state chart diagram shown in XMI format is almost identical to XML file format. XML parser reads the XMI file and the list of events, guard conditions, states and actions are generated.

5.4.3 Construct USG and generate adjacency matrix

After parsing the XMI code of the state chart diagram, we extract all the information such as states, transition, action and events. Then we use this information, and generate the adjacency matrix from it. And subsequently transform the state chart diagram into USG (UML state chart graph).

5.4.4 Traverse the graph and generate Test cases

Then we perform the traversal of the USG using adjacency matrix to generate the valid test sequences. For traversing the graph, we use backtracking DFS technique to ensure that every transition, state and path is covered. In our approach we have used DFS for traversing, because as with DFS it becomes easy to visit each state of the path. And this is useful to achieve transition path coverage. And after traversing the graph test sequences of the graph are generated. Then we apply the node coverage process to minimize the sequences and to generate the unit test cases.

5.5 Case Study

In this section, we discuss about the working of our proposed approach by using the application that we developed as an example. Here we are taking CreateIST phase of CIM application as an example. The state chart diagram of createIST phase is shown in Figure 5.4. The object enters into `CIM_Idea_Home`, when the

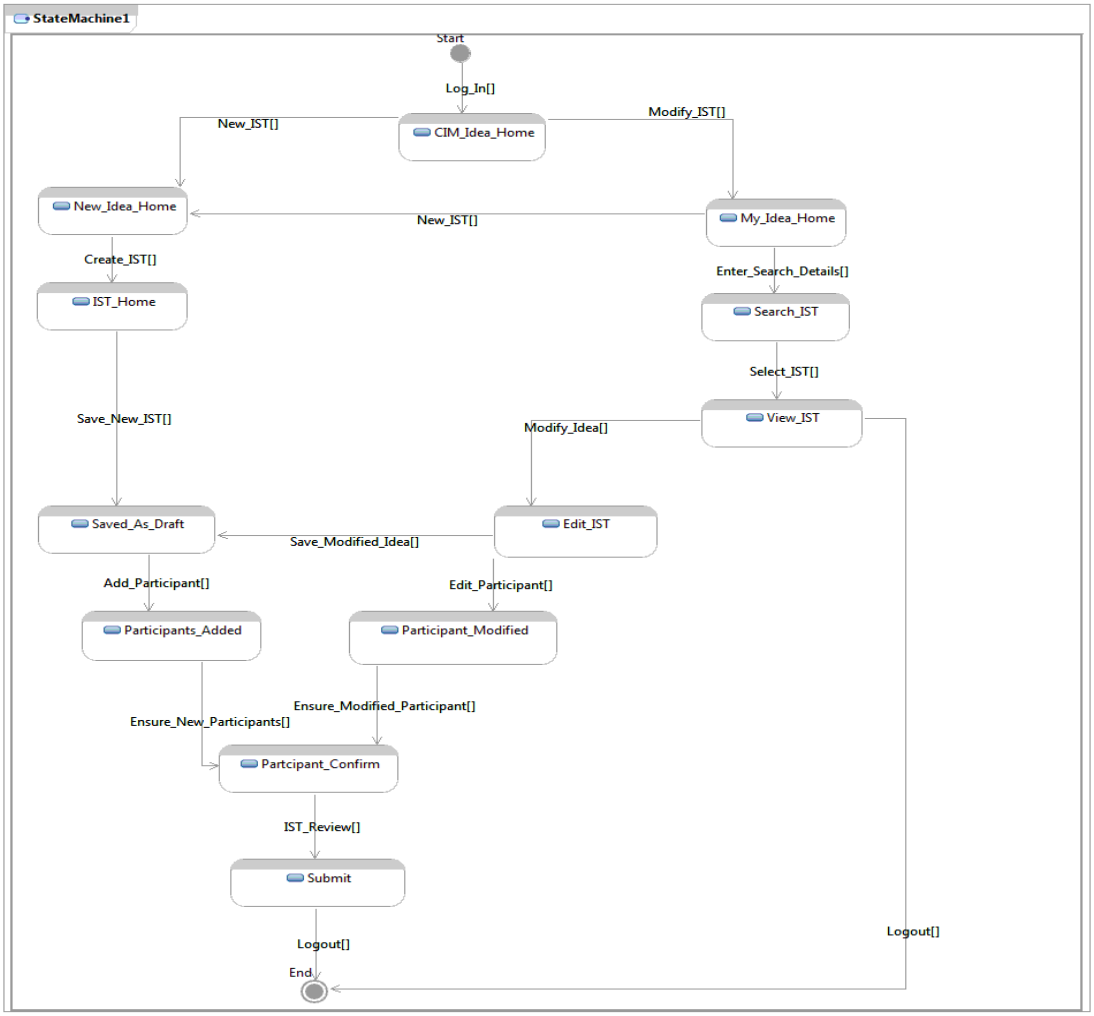

Figure 5.4: State Chart Diagram of CreateIST Phase

user provides the valid `Log_In` credentials. Now the user is able to see in which projects he is involved in `My_Idea` and if he want to create a new idea, then he can go for `New_Idea`. If the idea is novel, then the user enters into `New_Idea_Home` state. From there user has to fill all the mandatory fields in `IST_Home` by giving their

idea title, details, keywords, etc. then only the idea can be saved. Once all the mandatory fields are filled then only user can Save newly created IST by entering into `Save_As_Draft` state. Now users can add new participants by entering into a Participants Added state once the participants are added user has to ensure that participants has added by entering in `Participant_Confirm state`. After all the mandatory fields are filled and participants are added specially moderator then only user can go for `IST_Review` state by entering into Submit state. And after the IST is submitted then the user can logout.

If the idea exists then the user enters into `My_Idea_home` state. From this state user can go for `New_Idea_Home` if he has any idea. Now if user wants to modify any existing IST then firstly search the IST by entering valid search details, then only user can enter into a `Search_IST` state to search the existing IST. For viewing the existing IST user have entered the `View_IST` state by selecting IST but the IST must exist after providing the search details. Once the IST is selected one can view the IST and may logout after viewing it or user can edit the participants by entering into `Participants_Modified` state or he can go for adding any new participant by entering into Participants Added state. Once the participants added or modified user ensure depending upon addition or modification condition and go for Participant Confirm state. After all the mandatory fields are filled and participants are added or modified specially moderator then only user can go `IST_Review` state by entering into Submit state. And after the IST is submitted then the user can logout.

After the building of the state chart , diagram, we get the XMI code from the state chart diagram. And it is demonstrated in Figure 5.5. XMI code provides the information about all the states and transitions. It provides Id and the name of the object. Now consider the XMI code as an input to the parser, i.e. a Java code. Then the parser gathers all information about the state chart diagram. That is the information regarding object's state, transition & actions. Later USG is generated from the state chart diagram which represent the flow of control between two states of an object. In that USG nodes indicates the states and edges indicate the transitions. The screenshot of UML state chart graph is as shown in

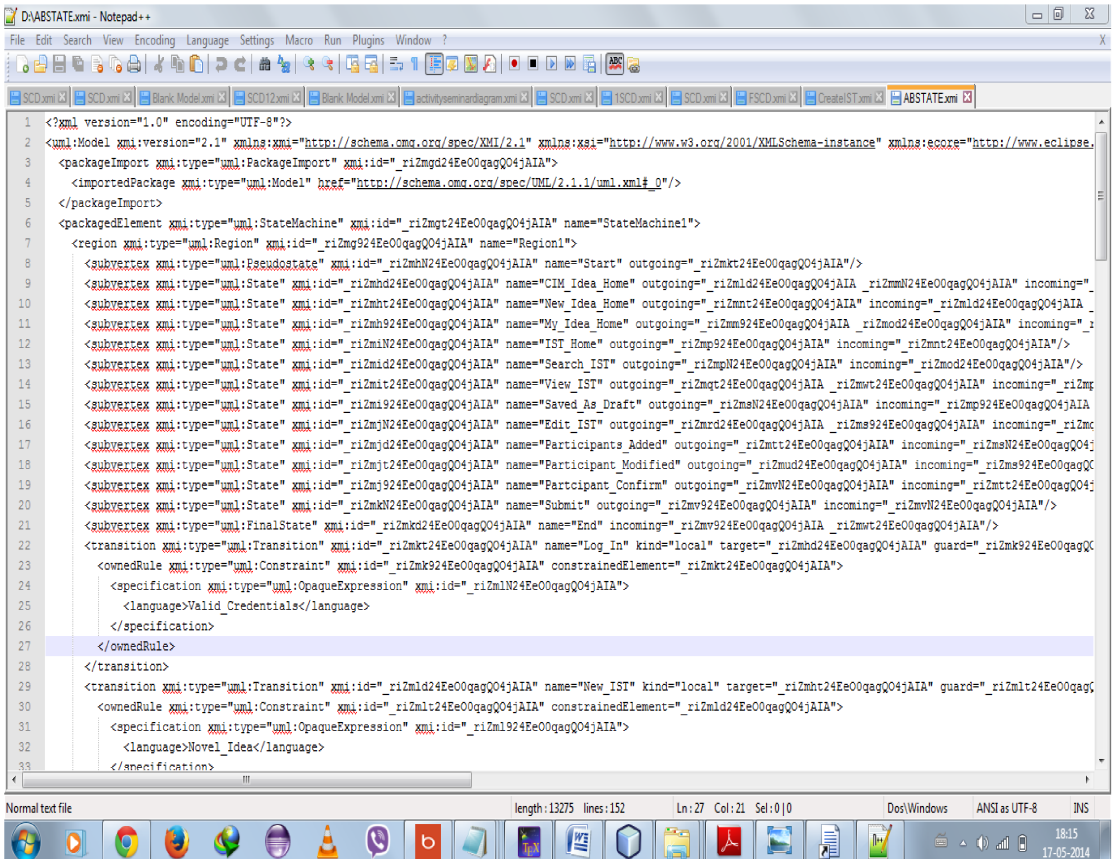

Figure 5.5: Snapshot of XMI Code of State Chart Diagram

Figure 5.6.

Pseudocode For Test Sequence Generation

Input: State Chart Diagram, Adjacency Matrix Admat of size n, n no. of states , Finalname

Ouptut: A Test Case TeS that fullfills transition Coverage.

[1] For i ϵ 1....n

[2] Begin

[3] Initialize arraylist AL;size =0;

[4] Initializie Ineteger 2D Array TeS;br=0;bc=0;

[5] Initialize Boolean 1D Array Visited;

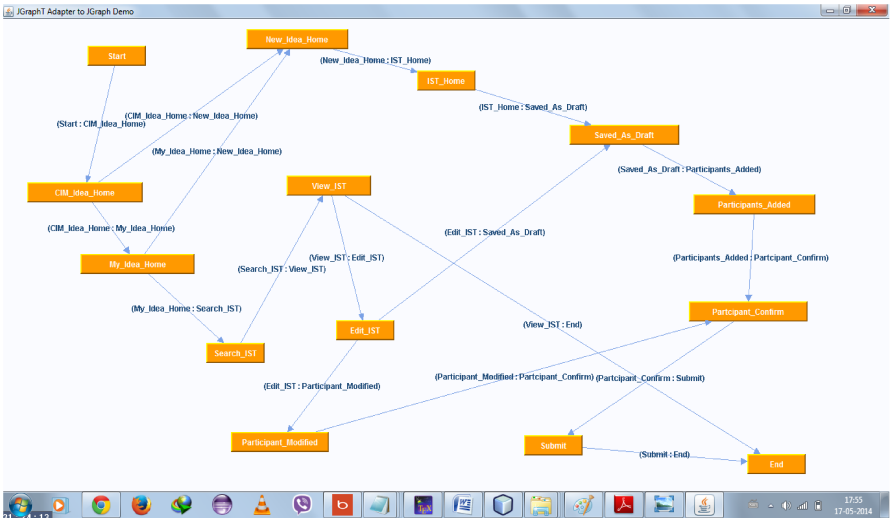

Figure 5.6: UML State Chart Graph generated by code

[6] Call Function DFS(S,i)

[7] size ← size-1;

[8] Remove sizeth element from AL

[9] End

[10] End for Loop

Function DFS(src,dest)

[1] Create an integer variable DIM and set value dest

[2] Add element src to AL

[3] size← size+ 1;

[4] Visited[src]←true;

[5] If src equal to dst

[6] for k ϵ 0 to arraylist.length

[7] Print Finalname.Elementat(k)

[8] TeS[i][j] ← k;

[9] bc←bc+1;

[10] End For Loop

[11] br←br+1;

[12] bc←0;

[13] End if

[14] For jϵ 2 to DIM

[15] if(Admat[src][j]==1)

[16] if(Visited[j]==false)

[17] Call Function(j,dest)

[18] set Visited[j]←false

[19] size←size-1

[20] Remove sizeth element from AL

[21] End if

[22] End if

[23] End For loop

[24] End Function

After that we get all the possible sequences of the state chart diagram with the help of the above algorithm. The test sequences generated by above algorithm is shown in Figure 5.7.

Minimization of the test cases

In this section we discussed about the minimization of the test cases generated to increase the test coverage. We have applied the technique name node coverage so

Test Seq ID	Test Sequences
1	Start-->Log_In-->CIM_Idea_Home
2	CIM_Idea_Home-->New_IST-->New_Idea_Home
3	CIM_Idea_Home-->Modify_IST-->My_Idea_Home
4	New_Idea_Home-->Create_IST-->IST_Home
5	My_Idea_Home-->New_IST-->New_Idea_Home
6	My_Idea_Home-->Enter_Search_Details-->Search_IST
7	IST_Home-->Save_New_IST-->Saved_As_Draft
8	Search_IST-->Select_IST-->View_IST
9	View_IST-->Modify_Idea-->Edit_IST
10	View_IST-->Logout-->End
11	Saved_As_Draft-->Add_Participant-->Participants_Added
12	Edit_IST-->Save_Modified_Idea-->Saved_As_Draft
13	Edit_IST-->Edit_Participant-->Participant_Modified
14	Participants_Added-->Ensure_New_Participants-->Partcipant_Confirm
15	Participant_Modified-->Ensure_Modified_Participant-->Partcipant_Confirm
16	Partcipant_Confirm-->IST_Review-->Submit
17	Submit-->Logout-->End
18	Start-->Log_In-->CIM_Idea_Home-->New_IST-->New_Idea_Home
19	Start-->Log_In-->CIM_Idea_Home-->Modify_IST-->My_Idea_Home
20	Start-->Log_In-->CIM_Idea_Home-->New_IST-->New_Idea_Home-->Create_IST-->IST_Home
21	Start-->Log_In-->CIM_Idea_Home-->Modify_IST-->My_Idea_Home-->New_IST-->New_Idea_Home-->Create_IST-->IST_Home

Figure 5.7: Test Sequence Generated For CreateIST Phase

that we can get the test cases effectively [28]. In our approach we choose one test sequence at a time and then finding out in the remaining test sequences, whether it is covered in any other test sequences. For example Let we have NCove (Tes1) = Tes4, Tes5, Tes6....., here we have selected the first sequence and then checking in the remaining to select those in which this sequence is covered. If for any of the test sequences NCove(Tes) =0 then it would be seen as an effective test case. After applying the above discussed approach the minimized sequences are generated. Here we can see Test39,Tes40,Tes41,Tes42 and Tes43 are empty and hence selected for test cases effectively as shown in Figure 5.8.

Minimized test sequences are as follows:

- Test Sequence for (TC_ID1)

  {Start,Log_In,CIM_Idea_Home}

  {CIM_Idea_Home,New_IST,New_Idea_Home}

  {New_Idea_Home,Create_IST,IST_Home}

  {IST_Home ,Save_New_IST,Saved_As_Draft }

  {Saved_As_Draft,Add_Participant,Participants_Added}

Node Coverage(NCove)	Resultant Test Sequence
NCove(Tes1)	{Tes18,Tes19,Tes20,Tes21,Tes22,Tes23,Tes24,Tes25,Tes26,Tes27,Tes28,Tes29,Tes30,Tes31,Tes32,Tes33,Tes34,Tes35,Tes36,Tes37,Tes38,Tes39,Tes40,Tes41,Tes42,Tes43}
NCove(Tes2)	{Tes18,Tes20,Tes24,Tes27,Tes31,Tes35,Tes39}
NCove(Tes3)	{Tes19,Tes21,Tes22,Tes23,Tes25,Tes26,Tes28,Tes29,Tes30,Tes32,Tes33,Tes34,Tes36,Tes37,Tes38,Tes40,Tes41,Tes42,Tes43}
NCove(Tes4)	{Tes20,Tes21,Tes24,Tes25,Tes27,Tes28,Tes31,Tes32,Tes35,Tes36,Tes39,Tes40}
NCove(Tes5)	{Tes21,Tes25,Tes28,Tes32,Tes36,Tes40}
NCove(Tes6)	{Tes22,Tes23,Tes26,Tes29,Tes30,Tes33,Tes34,Tes37,Tes38,Tes41,Tes42,Tes43}
NCove(Tes7)	{Tes24,Tes25,Tes27,Tes28,Tes31,Tes32,Tes35,Tes36,Tes39,Tes40}
NCove(Tes8)	{Tes23,Tes26,Tes29,Tes30,Tes33,Tes34,Tes37,Tes38,Tes41,Tes42,Tes43}
NCove(Tes9)	{Tes26,Tes29,Tes30,Tes33,Tes34,Tes37,Tes38,Tes41,Tes42}
NCove(Tes10)	{Tes43}
NCove(Tes11)	{Tes27,Tes28,Tes29,Tes31,Tes32,Tes33,Tes35,Tes36,Tes37,Tes39,Tes40,Tes41}
NCove(Tes12)	{Tes29,Tes33,Tes37,Tes41}
NCove(Tes13)	{Tes30,Tes34,Tes38,Tes42}
NCove(Tes14)	{Tes31,Tes32,Tes33,Tes35,Tes36,Tes37,Tes39,Tes40,Tes41}
NCove(Tes15)	{Tes34,Tes38,Tes42}
NCove(Tes16)	{Tes35,Tes36,Tes37,Tes38,Tes39,Tes40,Tes41,Tes42}
NCove(Tes17)	{Tes39,Tes40,Tes41,Tes42}
NCove(Tes18)	{Tes20,Tes24,Tes27,Tes31,Tes35,Tes39}
NCove(Tes19)	{Tes21,Tes22,Tes23,Tes25,Tes26,Tes28,Tes29,Tes30,Tes32,Tes33,Tes34,Tes36,Tes37,Tes38,Tes40,Tes41,Tes42,Tes43}
NCove(Tes20)	{Tes24,Tes27,Tes31,Tes35,Tes39}
NCove(Tes21)	{Tes25,Tes28,Tes32,Tes36,Tes40}
NCove(Tes22)	{Tes23,Tes26,Tes29,Tes30,Tes33,Tes34,Tes37,Tes38,Tes41,Tes42,Tes43}
NCove(Tes22)	{Tes23,Tes26,Tes29,Tes30,Tes33,Tes34,Tes37,Tes38,Tes41,Tes42,Tes43}
NCove(Tes23)	{Tes26,Tes29,Tes30,Tes33,Tes34,Tes37,Tes38,Tes41,Tes42,Tes43}
NCove(Tes24)	{Tes27,Tes31,Tes35,Tes39}
NCove(Tes25)	{Tes28,Tes32,Tes36,Tes40}
NCove(Tes26)	{Tes29,Tes30,Tes33,Tes34,Tes37,Tes38,Tes41,Tes42}
NCove(Tes27)	{Tes31,Tes35,Tes39}
NCove(Tes28)	{Tes32,Tes36,Tes40}
NCove(Tes29)	{Tes33,Tes37,Tes41}
NCove(Tes30)	{Tes34,Tes38,Tes42}
NCove(Tes31)	{Tes35,Tes39}
NCove(Tes32)	{Tes36,Tes40}
NCove(Tes33)	{Tes37,Tes41}
NCove(Tes34)	{Tes38,Tes42}
NCove(Tes35)	{Tes39}
NCove(Tes36)	{Tes40}
NCove(Tes37)	{Tes41}
NCove(Tes38)	{Tes42}
NCove(Tes39)	{ }
NCove(Tes40)	{ }
NCove(Tes41)	{ }
NCove(Tes42)	{ }
NCove(Tes43)	{ }

Figure 5.8: Calculated Node Coverage

{Participants_Added,Ensure_New_Participants, Partcipant_Confirm }

{Partcipant_Confirm,IST_Review,Submit }

{Submit ,Logout, End}

- Test Sequence for (TC_ID2)

  { Start ,Log_In ,CIM_Idea_Home }

  { CIM_Idea_Home ,Modify_IST,My_Idea_Home }

  { My_Idea_Home,New_IST ,New_Idea_Home }

  { New_Idea_Home , Create_IST,IST_Home }

  { IST_Home , Save_New_IST,Saved_As_Draft }

{ Saved_As_Draft, Add_Participant,Participants_Added }

{ Participants_Added, Ensure_New_Participants,Partcipant_Confirm }

{ Partcipant_Confirm ,IST_Review,Submit }

{ Submit, Logout , End }

- Test Sequence for (TC_ID3)

  {Start , Log_In , CIM_Idea_Home}

  {CIM_Idea_Home, Modify_IST , My_Idea_Home }

  {My_Idea_Home , Enter_Search_Details, Search_IST }

  {Search_IST, Select_IST , View_IST }

  {View_IST, Modify_Idea , Edit_IST }

  {Edit_IST, Save_Modified_Idea , Saved_As_Draft }

  {Saved_As_Draft, Add_Participant , Participants_Added }

  {Participants_Added, Ensure_New_Participants , Partcipant_Confirm }

  {Partcipant_Confirm, IST_Review , Submit }

  {Submit , Logout , End }

- Test Sequence for (TC_ID4)

  {Start, Log_In , CIM_Idea_Home }

  {CIM_Idea_Home, Modify_IST , My_Idea_Home }

  {My_Idea_Home, Enter_Search_Details , Search_IST }

  {Search_IST , Select_IST , View_IST }

  {View_IST , Modify_Idea , Edit_IST }

  {Edit_IST , Edit_Participant , Participant_Modified }

  {Participant_Modified , Ensure_Modified_Participant , Partcipant_Confirm }

  {Partcipant_Confirm , IST_Review , Submit }

  {Submit , Logout , End }

- Test Sequence for (TC_ID5)

  {Start , Log_In , CIM_Idea_Home }

  {CIM_Idea_Home , Modify_IST , My_Idea_Home }

{My_Idea_Home , Enter_Search_Details , Search_IST }

{Search_IST , Select_IST , View_IST }

{View_IST , Logout , End }

5.6 Implementation of Our Approach

In this segment we discuss about the implementation of our approach. We have used this approach in other phases of the application to analyze its effectiveness. We have applied the Rational rose software to design the CreateIST phase of Collaborative Invention Mining in UML2.0. Firstly, we design the UML state chart diagram of CreateIST phase and then saved that diagram in XMI format. And so the adjacency matrix for the state chart diagram is also generated. And subsequently the state chart diagram is transformed into USG. This implementation is done by using Java NetBeansIDE 7.2.The screenshot of Java code is shown in Figure 5.9 and for USG it is shown in Figure 5.10.

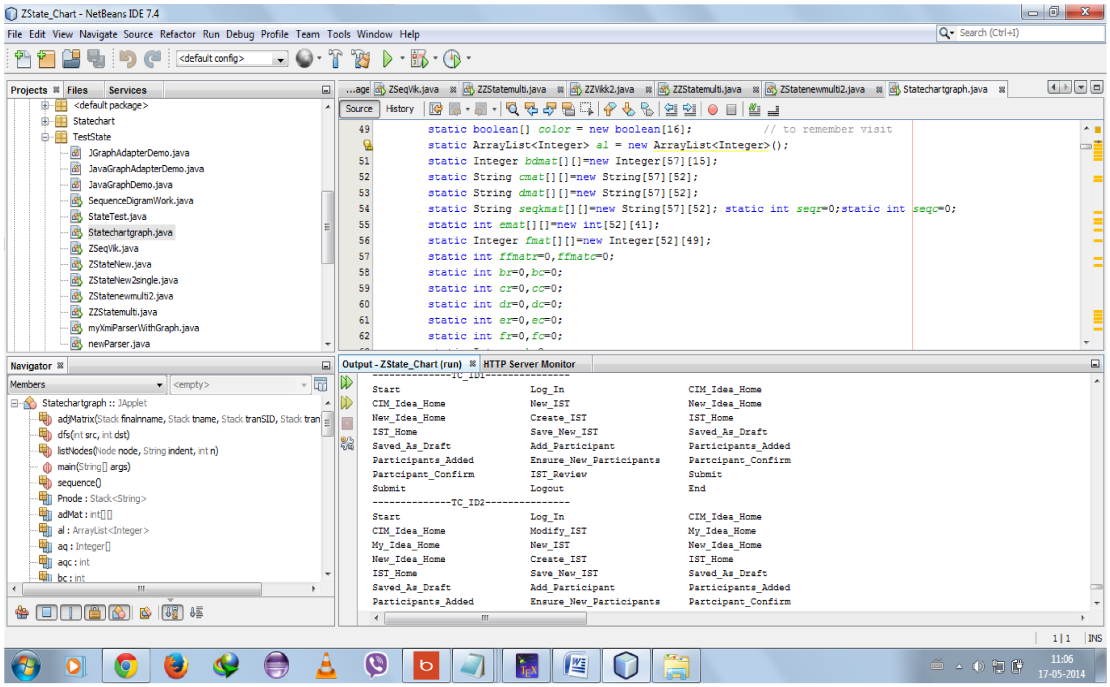

Figure 5.9: Screenshot of JAVA source code

We have implemented this approach by developing our own source code. And

after traversing the graph test sequences are generated. Then we apply our minimization algorithm to minimize the test sequences and generate the test cases. We also generate a .Dot extension file for the verification of USG by creating it, with the help of external tool. The graph generated by external is named as EFSM model. It graphically represented the graphs, where states denotes as nodes and transitions as directed edges between states. The screenshot of this graph is shown in Figure 5.10.And the test cases generated by our approach is shown in Table 5.1.

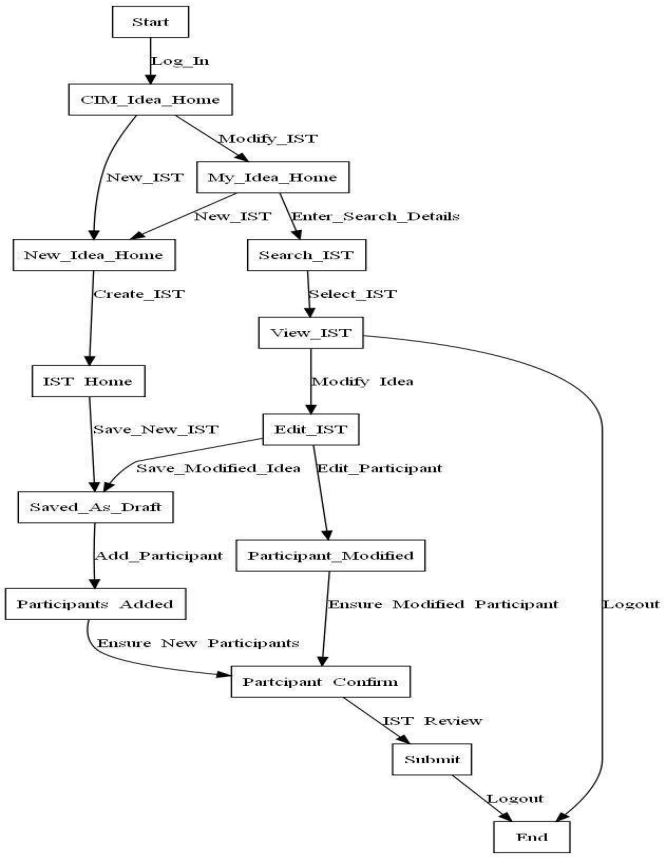

Figure 5.10: UML State Chart Graph generated by External Tool

Table 5.1: Generated Test Cases for CreateIST Phase

Test Case ID	Current State	Pre Condition	Input	Output State
1	Start	Valid_Credentials	Log_In	CIM_Idea_Home
2	CIM_Idea_Home	Novel_Idea	New_IST	New_Idea_Home
3	CIM_Idea_Home	Existing_Idea	Modify_IST	My_Idea_Home
4	My_Idea_Home	Novel_Idea	New_IST	New_Idea_Home
5	New_Idea_Home	Mandatory_Fields_Empty	Create_IST	IST_Home
6	My_Idea_Home	Valid_Search_Details	Enter_Search_Details	Search_IST
7	Search_IST	IST_Exist	Select_IST	View_IST

5.7 Comparison of our Work with Unit Test Cases Design in TCS

In this section, we compare the test cases designed by our approach with the test cases designed in TCS. One of the major advantage of our approach is that our methodology obtains the state coverage, transition coverage and action coverage. The unit test cases designed by our approach is less in count as compared to unit test cases designed in TCS. But our approach achieves the maximum functionality coverage as similar to unit test cases designed in TCS. And it is also helpful for the developers reduce their effort and time in designing test cases. The comparison between the unit test for different phases is shown in the Table 5.2.

5.8 Conclusion

We proposed a methodology to generate unit test cases from UML state chart diagrams. Firstly, we have constructed the state chart diagram for the CreateIST phase of CIM application. Then we traversed the state chart diagram construct

Table 5.2: Table Showing Comparison Between Designed Test Cases ; TCD=Test Case Design ; FC=Functionality Coverage

SI No.	TCD For Phase	Our Approach	TCD (TCS)	FC
1	Create_IST	17	30	100%
2	Add_Prior_Art_Parking_Lot	16	25	100%
3	Add_Participants_Image	19	32	100%
4	Parking_Lot_Phase	21	37	100%

the adjacency matrix and generate valid test sequences. And then apply the minimization technique to obtain the test cases with maximum coverage. Lastly we have compared the test cases generated from our approach with test cases designed in TCS.

Chapter 6

Conclusion and Future Work

In our approach, we mainly focus on test case generation of an application that was developed in TCS by using a state chart diagram.Our thesis promotes the technique of model based testing, which is used for complex software development process. In Section 6.1 we conclude about our contribution in CIM development. We conclude about our proposed approach as discussed in chapter 5 in section 6.2. And in section 6.3 scope of future work is discussed.

6.1 Contribution to CIM

In this section, we discussed about our contribution to back end development of collaborative invention mining using Eclipse IDE for Java technology. In CIM , around 25 graphical user interface is developed and 30 database table have been created for CIM aaplication. We were also part of business requirement analysis & design team. Then we have designed the unit test cases for different phases of the application. The process of testing was done by using a tool named as Application Life Cycle Management (ALM). ALM is a tool that was developed by TCS mainly for testing related activities. We have contributed from initial phase to final phase of the CIM application development.

6.2 Contribution to Proposed Approach

In chapter 5, we have developed a methodology in which the unit test cases are generated for CreateIST phase of CIM application that was developed by us in TCS. In our approach, firstly the state chart diagram is constructed for the CreateIST phase.Then generation of test cases takes place by transforming the state chart diagram into USG (UML state chart graph).Then we minimize the test case by using a node coverage technique for each of the test cases.We have considered only the path coverage as test coverage criteria for automatic unit test case generation.Then finally we compare the number of unit test cases generated by us with the test case designed in TCS.Our approach achieves the maximum coverage of functionality, which is an added advantage.

6.3 Future Work

Our work can be expanded in various directions.We have worked on automatic generation of unit test cases, but still several issues with respect to our approach both theoretical and practical. Therefore, further development is essential in our approach. In future we can apply this test case generation technique to other UML diagrams such as interaction diagrams, activity diagrams, etc. and prioritization will be done by using different approaches. Experimental studies show that this kind of approach might be useful in the future for making software more reliable by reducing the burden of testing effort.

Bibliography

[1] Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.

[2] Pan, Jiantao. Software testing.Retrieved January 5, 2006.

[3] L. Luo. Software testing techniques technology maturation and research strategy. Institute for Software Research International, Carnegie Mellon University, Pittsburgh, PA15232, USA, Tech. Rep. 17939, 2001.

[4] Miller, Edward. "Introduction to software testing technology." Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, 1981.

[5] Miller, Edward. "Introduction to software testing technology." Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, 1981.

[6] A. Bertolino, "Chapter 5: Software Testing," IEEE SWEBOK Trial Version 1.00, May 2001.

[7] Samaroo, Angelina, Geoff Thompson, and Peter Williams. Software Testing: An ISTQB-ISEB Foundation Guide. BCS, The Chartered Institute, 2010.

[8] R.Mall .Fundamentals of Software Engineering , Prentice Hall, 2nd edition , 2003.

[9] R. V. Binder. Testing Object-Oriented System Models, Patterns, and Tools. NY: Addison-Wesley, 1999.

[10] Rutherford, Matthew J., Antonio Carzaniga, and Alexander L. Wolf. "Simulation-based test adequacy criteria for distributed systems." Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2006.

[11] M.Blackburn,R.Busser & A.Nauma, Why Model-Based Test Automation is Different and What You Should Know to Get Started. International Conference on Practical Software Quality and Testing, (Survey on automation of model-based testing),Washington, USA, 2004.

[12] Gross, Hans-Gerhard. Component-based software testing with UML. Vol. 44. Heidelberg: Springer, 2005.

[13] Leiserson Cormen. Software testing technique. Vol. 15, Prentice hall, 2nd Edition, 1990.

[14] Supaporn Kansomkeat and Sanchai Rivepiboon. "Automated Generating Test Case Using UML Statechart Diagrams "SAICSIT, 2003.

[15] Y. G.Kim, H. S.Hong, D. H.Bae, and S. D.Cha.Software Testing Verification and Reliability,chapter Test cases generation from UML state diagram.ACM,pp.187 - 192, 1999.

[16] N.Kosmatov,, B.Legeard, F.Peureux, & M.Utting. "Boundary coverage criteria for test generation from formal models." Software Reliability Engineering, 15th International Symposium on. IEEE, 2004.

[17] Gnesi Stefania, Latella, Diego, and Massink Mieke. Formal test-case generation for UML statecharts, Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer Systems Navigating Complexity, 2004.

[18] T.Systa , K.Koskimiesa and E. Makine. Automated compression of state machines using UML statechart diagram. Information and Software Technology,Vol.4 pp.565-578, 2002.

[19] S. K.Swain,D.P. Mohapatra and R.Mall .Test case generation based on state and activity models. Journal of Object Technology,pp.1-27, 2010.

[20] P.Samuel, R. Mall and A. K. Bothra. Auto-matic Test Case Generation Using Unified Modeling Language (UML) State Diagrams. IET Software, Vol.2, Issue 2, pp. 79-93, 2008.

[21] R. Swain, V. Panthi, P. K. Behera, D. P. Mohapatra. Automatic Test case Generation From UML State Chart Diagram, International Journal of Computer Applications (0975-8887) Vol. 42, No.7, pp.26-36, doi: 10.5120/5706-7756, March 2012.

[22] Mohanty, S.K. and Sarkar, S. and Viswanathan, J., Collaborative system and method to mine inventions, url=http://www.google.com/patents/EP2637130A1?cl=en,Google Patents,EP Patent App. EP20,120,171,102, 2013.

[23] P. Frankl and S. N. Weiss. An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria. In Proceedings of the symposium on Testing, analysis, and verification, 1991.

[24] J.Offutt and A.Abdurazik. Generating tests from UML specifications. In Proceedings of 2nd International Conference. UML, Lecture Notes in Computer Science, pp. 416-429, 1999.

[25] R.Blanco,, J. G. Fanjul and J.Tuya.Test case generation for transition-pair coverage using Scatter Search.International Journal of Software Engineering and Its Applications Vol. 4, No. 4, October 2010.

[26] J.Offutt, S. Liu, A. Abdurazik and P. Ammann.Generating test data from state-based specifications.Software Testing,Verification and Reliability,chapter 13:25-53, 2003.

[27] K.Koster and D. C.Kao. State coverage: A structural test adequacy criterion for behavior checking. In ESEC/FSE, 2007.

[28] Ranjita Kumari Swain, Prafulla Kumar Behera, and Durga Prasad Mohapatra. "Minimal TestCase Generation for Object-Oriented Software with State Charts." International Journal of Software Engineering & Applications 3.4, 2012.