An Approach to Automatic Generation of Test Cases Based on Use Cases in the Requirements Phase

U.Senthil Kumaran[1], S. Arun Kumar[2], K.Vijaya Kumar[3]

1. Assistant Professor, SITE, VIT University, Vellore, Tamil nadu, India.
2. Assistant Professor, SCSE, VIT University, Vellore, Tamil nadu, India.
3. Assistant Professor, SCSE, VIT University, Vellore, Tamil nadu, India.

ABSTRACT

The main aim of this paper is to generate test cases from the use cases**.** In the real-time scenario we have to face several issues like inaccuracy, ambiguity, and incompleteness in requirements this is because the requirements are not properly updated after various change requests. This will reduce the quality of test cases. To overcome these problems we develop a solution which generates test cases at the early stages of system development life cycle which captures maximum number of requirements. As requirements are best captured by use cases our focus lies on generating test cases from use case diagrams.

Keywords: Test cases, Use cases.

I. INTRODUCTION

In software engineering, the most common definition of a test case is a set of conditions or variables under which a tester will determine if a requirement or use case upon an application is partially or fully satisfied. Creating test cases adds an extra level of precision to a use case. Test cases are commonly designed based on program source code. This makes test case generation difficult especially for testing at cluster levels. Further this approach proves to be inadequate in component-based software development, where the source code may not be available to the developers. It is therefore desirable to generate test cases automatically from the software design documents, rather than code or code-based specifications. Test generation from design documents has the added advantage of allowing test cases to be available early in the software development cycle, thereby making test planning more effective. Further, in design based tests the generated test data is independent of any particular implementation of the design. The benefit of creating a test case is that it assists in identifying the important concepts in use case. In this paper a new scheme has been proposed to generate the test cases from a use case diagram efficiently.

II. PROBLEM STATEMENT

Once a use case has been captured, the first part of design is the validation of the use case to ensure that it can be implemented. Creating test cases adds an extra level of precision to a use case. Some of the problems which we come across are that the generated test cases might be incomplete, each case not describing enough details of use, they might not be updated when the requirements are changed, inaccurate, missing of entire functionality. An additional benefit of this approach is that creating the test cases assists in identifying the important concepts in each use case. In OO development we use classes to represent concepts, so this early nomination of candidate classes is useful.

III. LITERATURE SURVEY

Test cases are key to the process because they identify and communicate the conditions that will be implemented in test and are necessary to verify successful and acceptable implementation of the product requirements. They are all about making sure that the product fulfills the requirements of the system. So generating the test cases automatically from use cases is an advantage for software testing.

Use case development begins early on, so it gives opportunity to capture the key product functionality are available in early iterations. Use cases tell the customer what to expect, the developer what to code, the technical writer what to document, and the tester what to test. For software testing, creation of test cases is the fundamental step, and then the test procedures are designed for the test cases, and finally the test scripts. UML (Unified Modeling Language) is the most widely used language for generating use cases, many researchers are using UML diagrams such as state-chart diagrams, use-case diagrams, sequence diagrams, and etc to generate test cases and this has led to Model based test case generation. Even though variety of approaches have been proposed, with the advent of modeling tools like Rational Rose, for a decade there has been constant research on generating test cases based on specifications and design models. For easy understanding, we have classified test case generation approaches mainly into two categories. 1. Specification based test case generation. 2. Model based test case generation.

IV. RELATED WORK

Many researchers and practitioners have been working in generating optimal test cases based on the specifications;

There are several discussions on how to use cases may help the testing process to be done early in the development lifecycle. Jacobson et al. explained tests can be derived from use cases in three types: First, tests of the expected flow of event; second, tests of unusual flow of events; and third, test of any requirements attached to a use case. Unfortunately, Jacobson et al. did not discuss on how to choose test cases and how to know when you are done. Binder , Heumann and Wood et al. derive test cases directly from requirements in natural language. Wood et al.state that the most integral part of use case for generating test cases is the Event Flow (basic flow and alternate flow). The next step is creating the scenario based on the Event Flow before it can be used to generate the test cases. However, automating the testing operation can reduce the cost and improve the reliability and effectiveness of software testing.

V. APPROACH USED

Several approaches have been proposed for test case generation, mainly random, path-oriented, goal-oriented and intelligent approaches. Random techniques determine test cases based on assumptions concerning fault distribution. Path-oriented techniques generally use control flow information to identify a set of paths to be covered and generate the appropriate test cases for these paths. These techniques can further be classified as static and dynamic. Static techniques are often based on symbolic execution, whereas dynamic techniques obtain the necessary data by executing the program under test. We make use of this dynamic technique in developing this system.

VI. ASSUMPTIONS

The pre-conditions for each scenario are assumed to be got from the user of the system .For effective generation of test cases, use case scenarios are to be derived from each use case. At least one scenario can be derived from each use case and for each scenario at least one test case will be generated. A use-case scenario is an instance of a use case, or a complete "path" through the use case.

VII. PROPOSED SYSTEM

The main aim of this proposed system is to generate test cases from use cases**.** Through the years a number of different methods have been proposed for generating test cases. Test cases can also be derived from system requirements. One of the advantages of producing test cases from use cases is that they can be created earlier in the development life cycle and be ready for use before the programs are constructed. Additionally, when the test cases are generated early, Software Engineers can often find inconsistencies and ambiguities in the requirements specification and design documents. This will definitely bring down the cost of building the software systems as errors are eliminated early during the life cycle. The most important part of a use case for generating test cases is the flow of events. The two main parts of the flow of events are the **basic flow of events** and the **alternate flows of events**. The basic flow of events should cover what "normally" happens when the use case is performed. The alternate flows of events cover behavior of an optional or exceptional character relative to normal behavior, and also variations of the normal behavior.

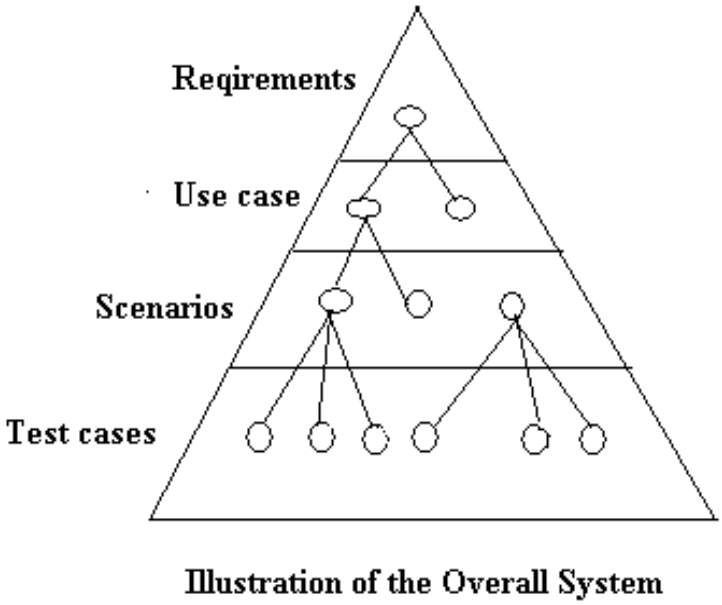

Fig 1. Overall System

Use cases describe functional requirements. In addition, every use case maps to many scenarios. Mapping use cases to scenarios then, is a one to many relationships. Scenarios map to test case also in a one to many relationship.

VIII. COMPONENTS OF THE SYSTEM

There are three basic components involved in this system: use case diagram, use case scenario and test cases.
In a use case diagram, the ovals represent use cases, and the stick figures represent "actors," which can be either humans or other systems. The lines represent communication between an actor and a use case. Each use case represents a big chunk of functionality that will be implemented, and each actor represents someone or something outside our system that interacts with it.

The most important part of a use case for generating test cases is the flow of events. The two main parts of the flow of events are the basic flow of events and the alternate flows of events. The basic flow of events should cover what "normally" happens when the use case is performed. The alternate flows of events cover behavior of an optional or exceptional character relative to normal behavior, and also variations of the normal behavior. You can think of the alternate flows of events as "detours" from the basic flow of events.

A use-case scenario is an instance of a use case, or a complete "path" through the use case. End users of the completed system can go down many paths as they execute the functionality specified in the use case. Following the basic flow would be one scenario. Following the basic flow plus alternate flow 1A would be another. The basic flow plus alternate flow 2A would be a third, and so on.

A test case is a set of test inputs, execution conditions, and expected results developed for a particular objective: to exercise a particular program path or verify compliance with a specific requirement, for example. The purpose of a test case is to identify and communicate conditions that will be implemented in test. Test cases are necessary to verify successful and acceptable implementation of the product requirements (use cases).

IX. SYSTEM DESIGN

The workspace for drawing use case diagram is created in which all the requirements for drawing the UCD are included. Once all the basic requirements are collected the initial step is to draw the use case diagram. The first step is to collect the details about each use case. A significant amount of detail goes into fully specifying a use case. All the details collected are stored in a data base and are retrieved from the data base whenever needed. The second thing to be concentrated before we use use cases for generating test cases is the use case scenarios. A use case scenario is an instance of a use case or a complete path through the use case. A scenario is a

combination of basic flows and alternate flows. Following the basic flow would be one scenario. One basic flow and one alternate flow would be another, one basic flow plus two alternate flows would be another and goes on...From these use case scenarios, test cases for the given use case will be generated.

X. PROPOSED SYSTEM FRAME WORK

In a software development project, use cases define system software requirements. Use case development begins early on, so real use cases for key product functionality are available in early iterations. Test cases can also be derived from system requirements. Once all the requirements are collected the system is now ready to draw a use case diagram for which the test cases can be generated based upon their textual descriptions. One of the advantages of producing test cases from specifications and design is that they can be created earlier in the development life cycle and be ready for use before the programs are constructed.

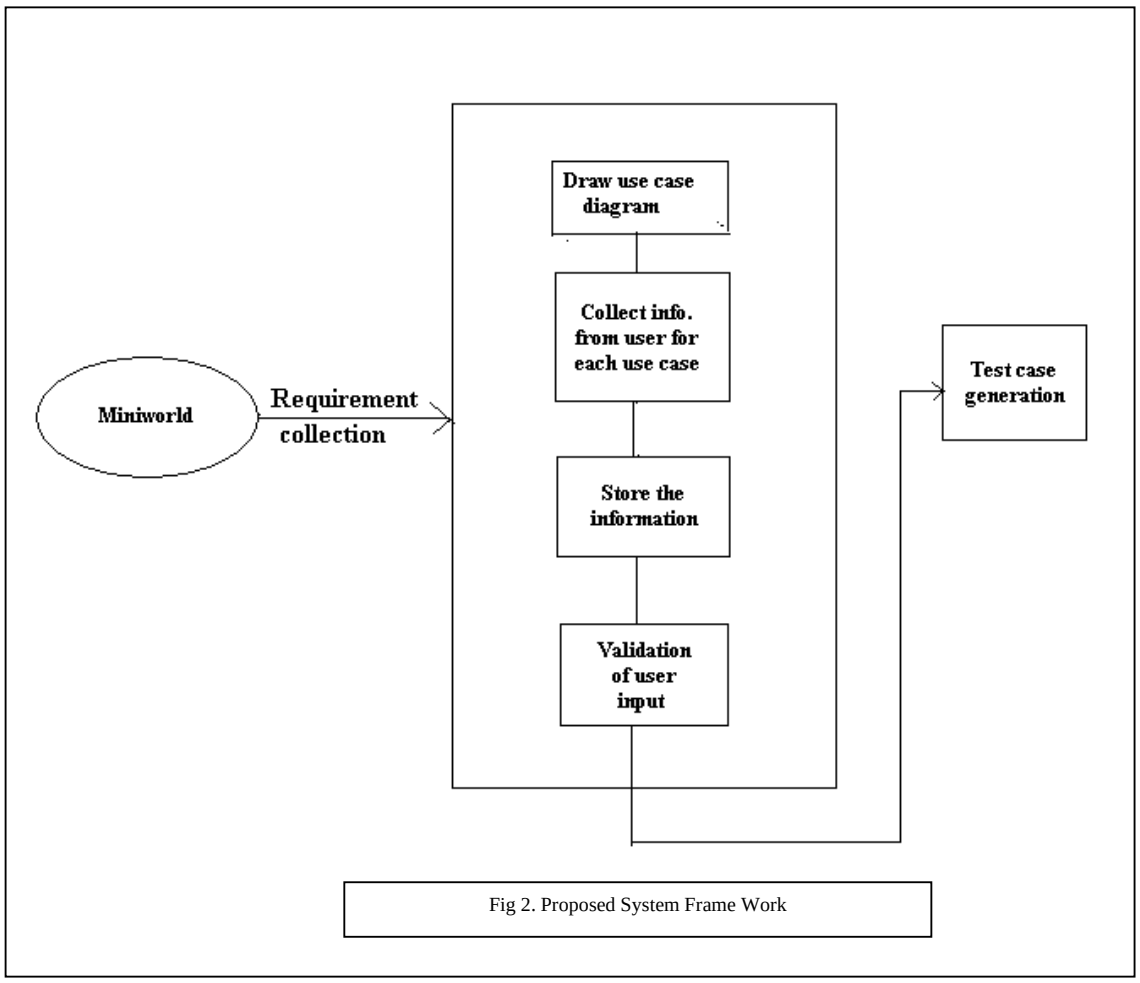

Fig 2. Proposed System Frame Work

XI. SYSTEM METHODOLOGY

1. Designing of the work space for drawing Use case diagram.

2. Getting the use case diagram from the user and storing all the details internally.

3. Generating the scenarios based upon the use cases.

4. Generating the test cases.

DFS

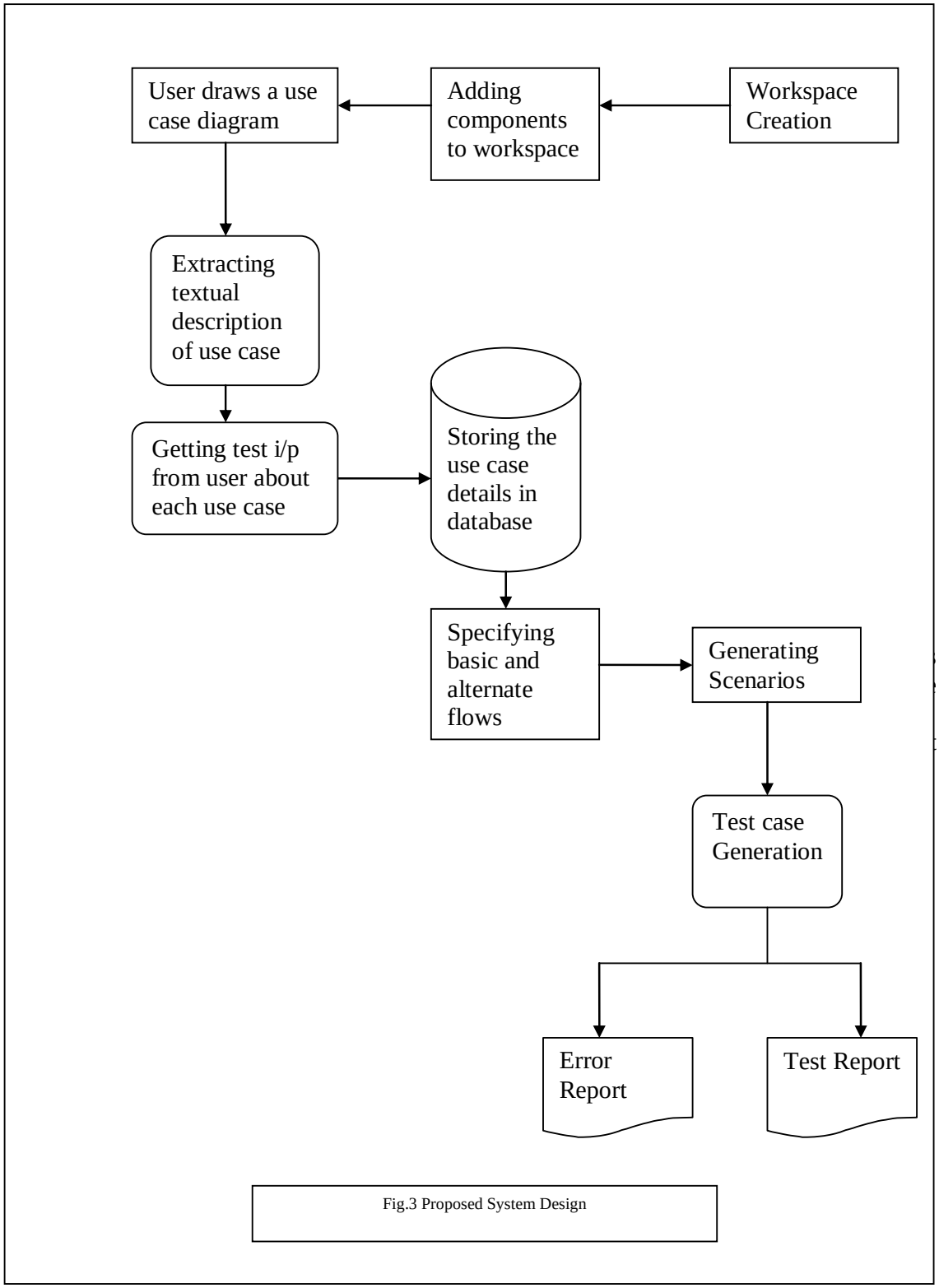

Fig.3 Proposed System Design

These details are then stored in a data base. Test cases are generated based upon the basic flows which are the normal happenings of the use case and the alternate flows which are the exceptional things associated with the use case. The combination of basic and alternate flows forms several scenarios. Each use case will have at least

one scenario and each scenario will have at least one test case. Based upon the several scenarios, test cases will be generated.

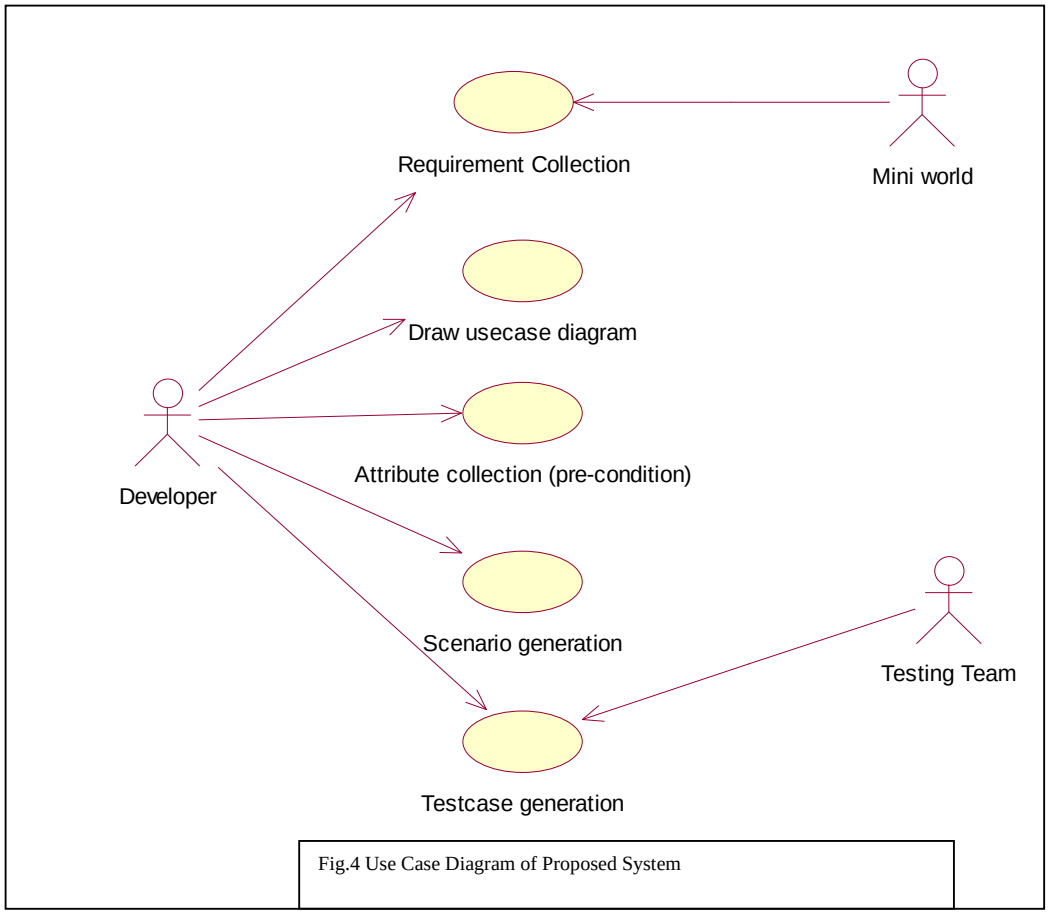

Fig.4 Use Case Diagram of Proposed System

In UML specification, requirements analysis and design are usually done using diagrams. One particular diagram (a use-case diagram) is used to specify requirements of the system. In a use-case diagram, two important factors are used to describe the requirements of a system. They are actors and use cases. Actors are external entities that interact with the system and use cases are the behavior (or the functionalities) of a system. The use cases are used to define the requirements of the system. These use cases represent the functionalities of the system. Most often, each use case is then converted into a function representing the task of the system. Therefore, we can convert from each of the use case into one test case or many test cases.

The relationship of the conversion is either one to one or one to many. However, if we have many use cases, then we will have many test cases.

A. Workspace Creation Classification

The component type is a module.

Definition

A workspace is created to facilitate the users to draw use case diagrams.

Responsibilities

User should be able to draw use case diagrams in this environment. This workspace is provided with all the requirements needed to draw use case diagrams.

Constraints

The workspace should be designed in such a way that the users will not face any difficulties or problems while they perform some actions in the workspace like drawing, editing or saving the drawn use case diagrams. The interface between the user and the workspace should be good.

Composition

The workspace consists of enough area to draw the use case diagrams. Separate sections should be there in which the various symbols which build the use case diagrams are placed.

Uses/Interactions

Designing the workspace stands at the first level in generating the test cases. The users will be able to draw use case diagrams and the details about each use case are also stored in a data base which can be retrieved later.

Resources

Java net beans are used to design the workspace using the swing concepts.

Processing

Once when the user draws a use case diagram, all the details about each use case is taken as input from the user and is stored in the data base.

B. Deriving Scenarios for each use case

Classification

The component type is a module.

Definition

A use-case scenario is an instance of a use case, or a complete "path" through the use case. Scenario is built using the basic flow and alternate flows. Basic flows are the normal happenings of the use case where the alternate flows are its exceptional cases. Scenario is nothing but the combination of basic flows and alternate flows.

Responsibilities

A scenario should clearly explain all the possible basic flows and alternate flows associated with each use case. Each scenario will have at least one test case.

Constraints

Each use case should have at least one scenario and each scenario should have at least one test case. Usually more than one scenario is built for each use case. Following the basic flow would be one scenario, one basic flow plus one alternate flow would be another, one basic and two alternate flows would be another and goes on...

Composition

For any use case the first scenario starts with the basic flow and is later combined with the alternate flows. Each scenario will have a name associated with it.

Uses/Interactions

Deriving scenarios plays an important role in generating test cases from the use case diagrams. Based upon the various scenarios derived for each use case, several test cases will be generated. Scenarios depend upon the details of each use case which is given as a test input.

Resources

Scenarios will be generated based upon the details of the use case.These information is stored in the data base which can be retrieved later for generating test cases.

Processing

The user will give all the details about each use case which are stored in a data base. Based upon those details basic flows and alternate flows are found out whose different combination built up the scenarios.

C. Generating Test cases

Classification

The component type is a module.

Definition

A test case is a set of test inputs, execution conditions, and expected results developed for a particular objective. Test cases are necessary to verify successful and acceptable implementation of the product requirements (use cases).

Responsibilities

Test cases are key to the process because they identify and communicate the conditions that will be implemented in test. They are all about making sure that the product fulfills the requirements of the system.

Constraints

For the efficient generation of test cases, a set of scenarios are needed. And for generating each scenario, detailed explanation about each use case is must. For each scenario at least one test case should be there.

Composition

Test cases are generated in the form of a tabular column which consists of several scenarios, the basic and alternate flows and finally the result showing whether a particular test case is successful or not.

Uses/Interactions

Generating test case is an important activity for any software system to be carried out. Test case generation from use case adds an extra advantage for any system as testing is carried out at the early stages of software development life cycle.

Resources

The resources for generating test cases are the several scenarios which are built using the details. These details are given as a test input by the user.

Processing

A test case matrix is developed based upon the scenarios which will have all the valid and invalid condition. This is the first step in generating test cases and finally data values are given to check whether they are valid or not and the corresponding results are generated.

XII. SYSTEM IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system and is giving confidence on the new system for the users that it will work efficiently and effectively. It involves careful planning, investigation of the current system and constraints on implementation, design of methods to achieve the change over,

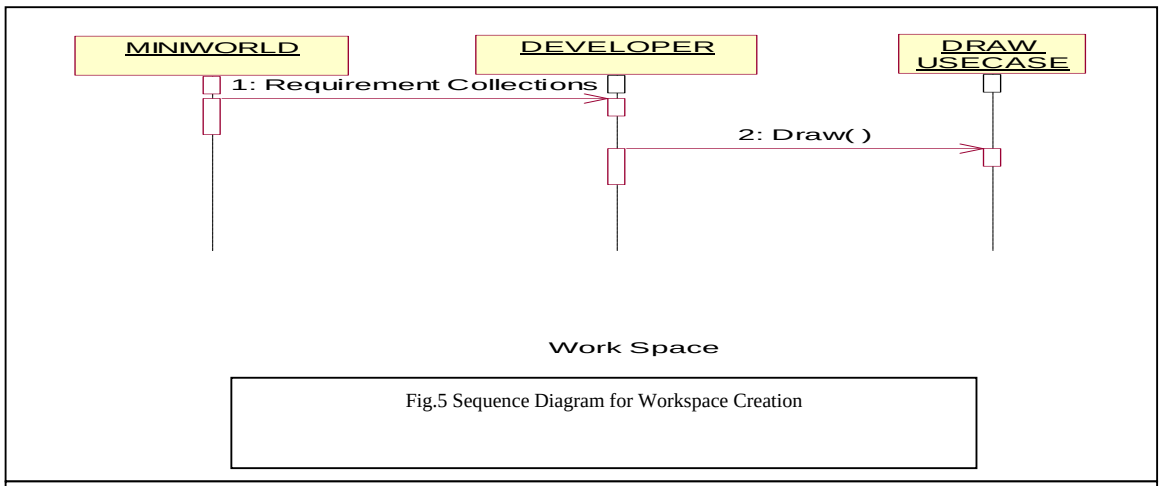

Fig.5 Sequence Diagram for Workspace Creation

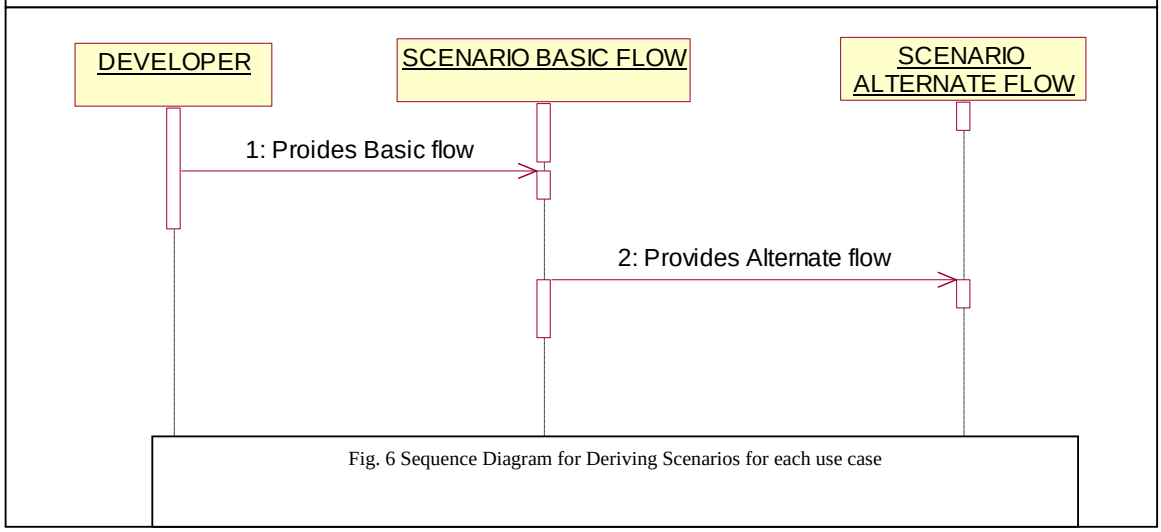

Fig. 6 Sequence Diagram for Deriving Scenarios for each use case

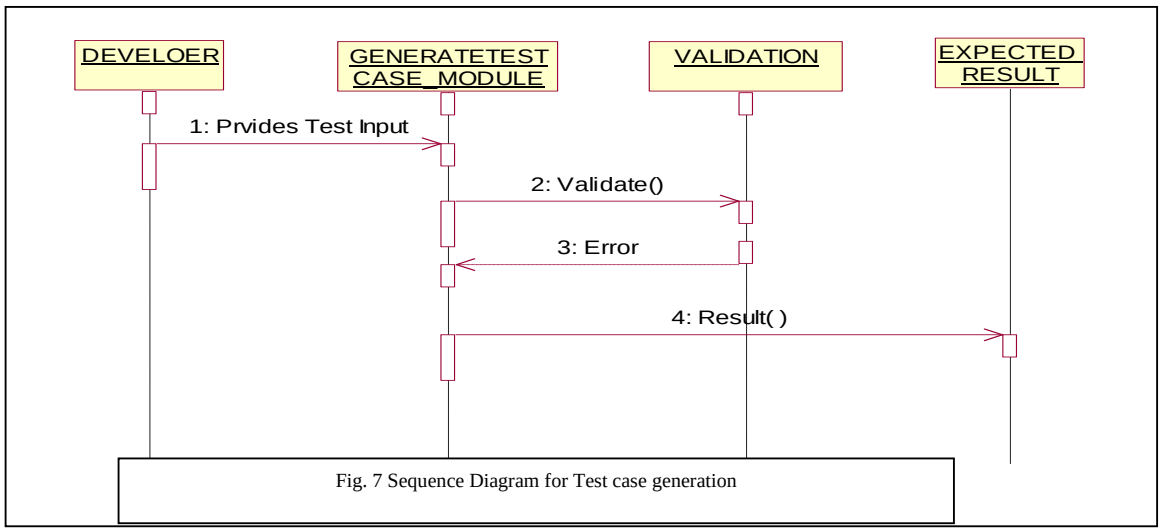

Fig. 7 Sequence Diagram for Test case generation

an evaluation, of change over methods. At the beginning of the development phase a preliminary implementation plan is created to schedule and manage the many different activities that must be integrated into plan. The implementation plan is updated throughout the development phase, culminating in a changeover plan for the operation phase. The major elements of implementation plan are test plan, training plan, equipment installation plan.

Starting out with the definition of use cases, a simplified overall view of the system's required functionality is constructed. Due to expressional limitations, use case diagrams are by themselves of little use – unless the individual use cases are refined in a commonly textual fashion.

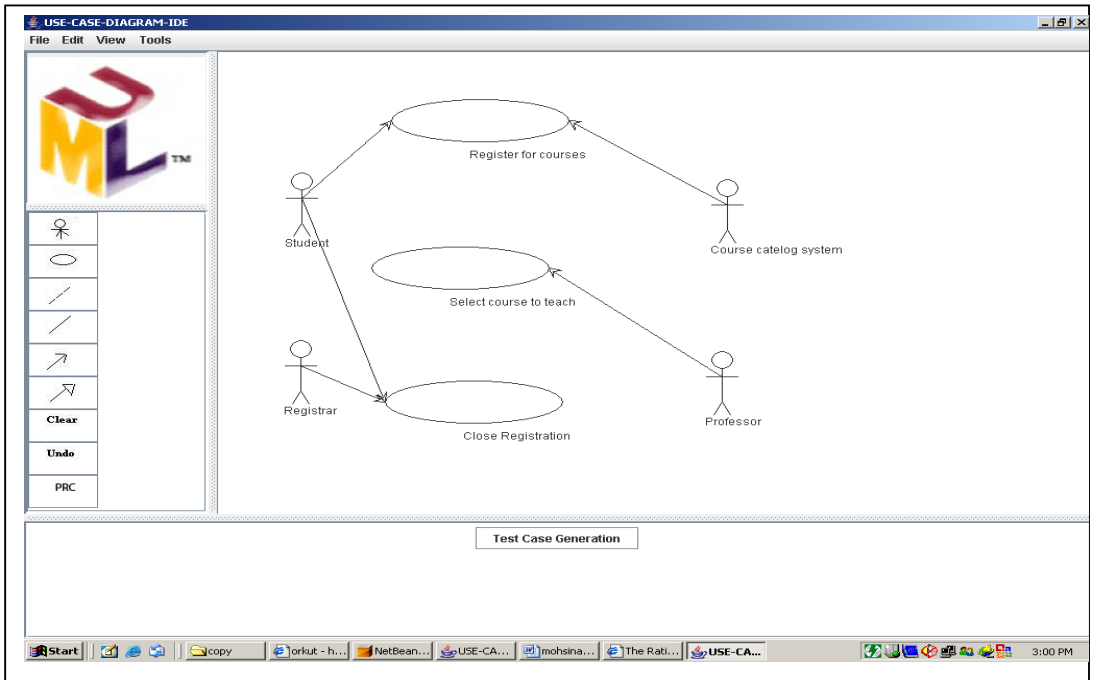

Fig. 8 Use case diagram-Course Registration

specification and generate test cases. In current practice, use cases are associated with the front end of the software development lifecycle and test cases are typically associated with the latter part of the lifecycle. By leveraging use cases to generate test cases, however, testing teams can get started much earlier in the lifecycle, allowing them to identify and repair defects that would be very costly to fix later, ship on time, and ensure that the system will work reliably.

XIII. Conclusion and future work

Many researchers and practitioners have been working in generating optimal test cases based on the specifications still 100% testing is impossibility.

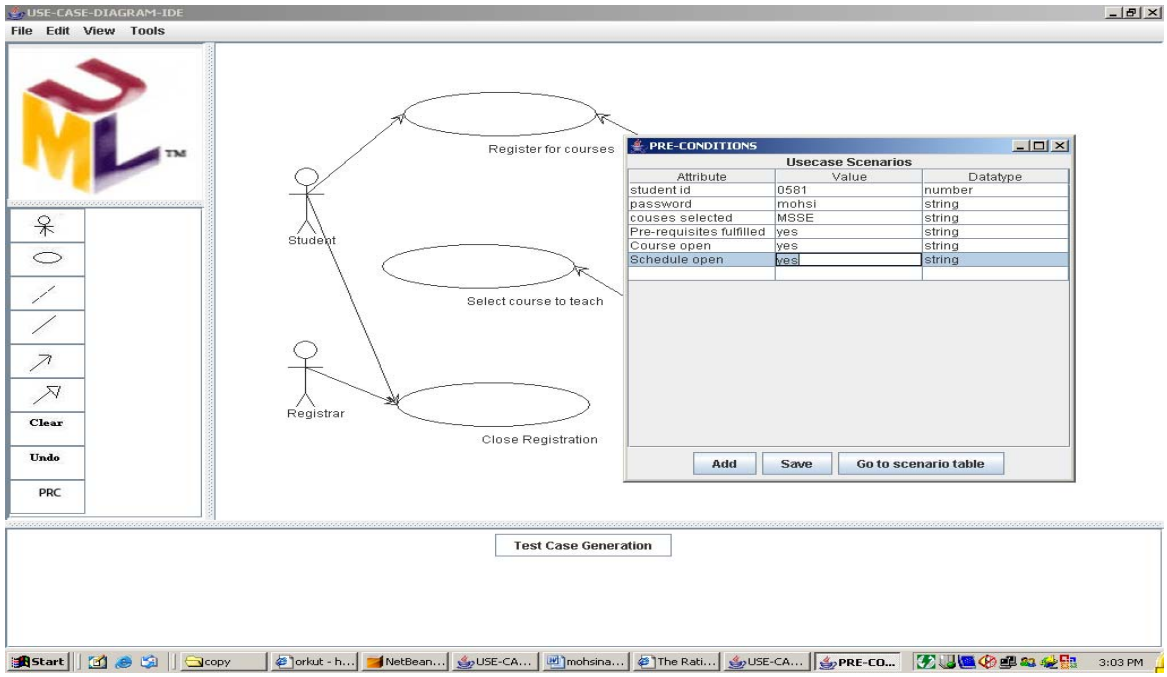

Fig. 9 Attribute values for "Register for courses" Use case

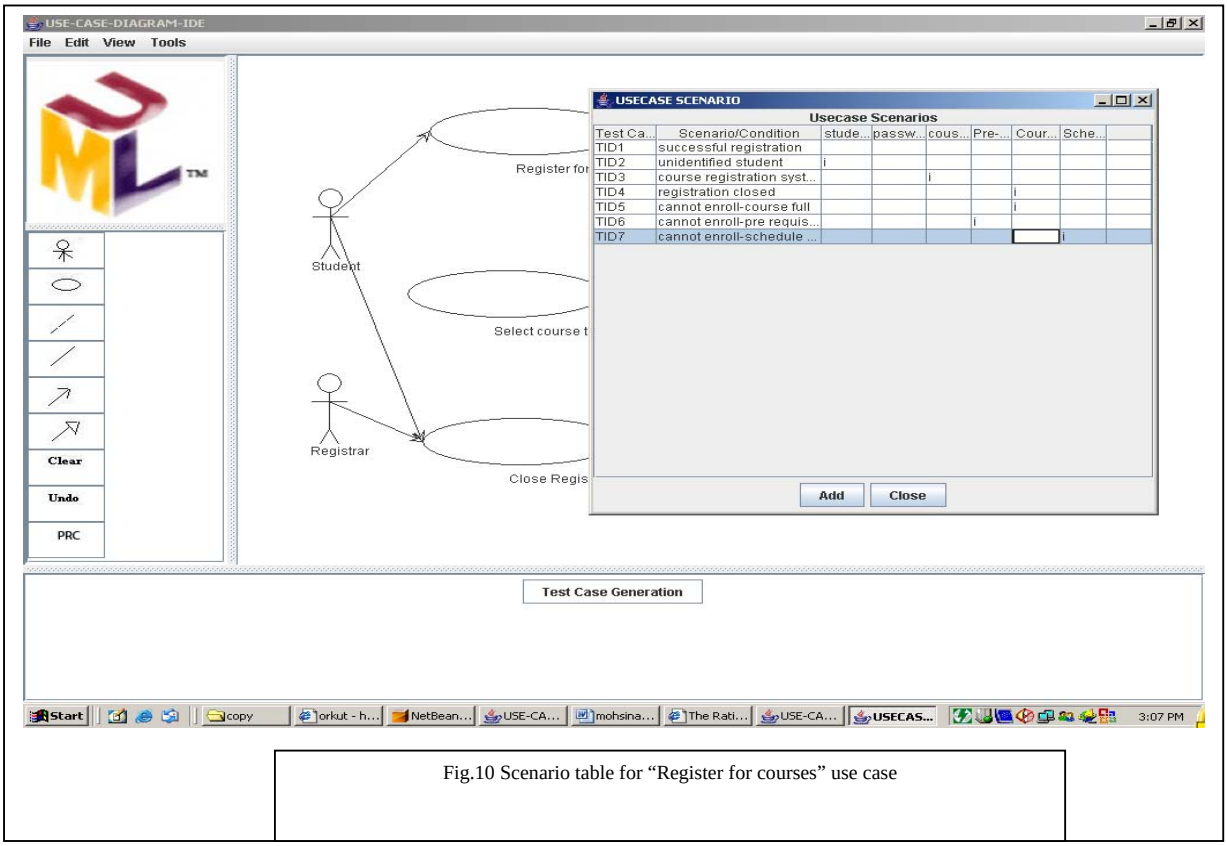

Fig.10 Scenario table for "Register for courses" use case

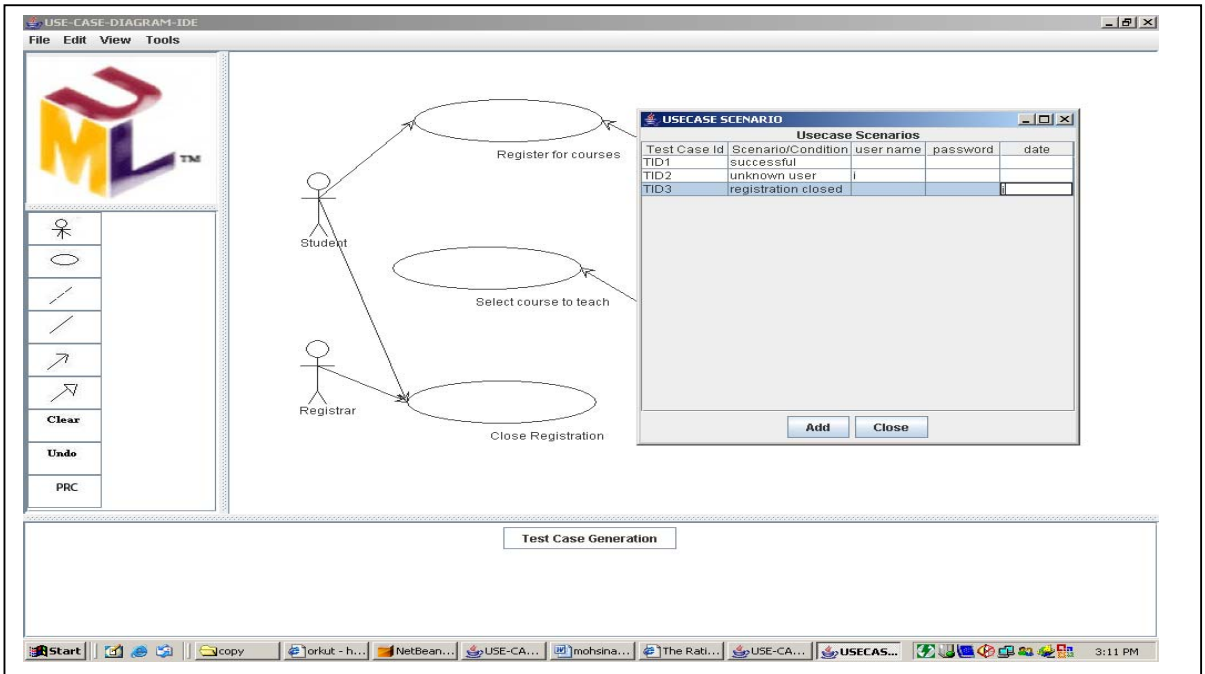

Fig. 11 Scenario table for "select course to teach" use case

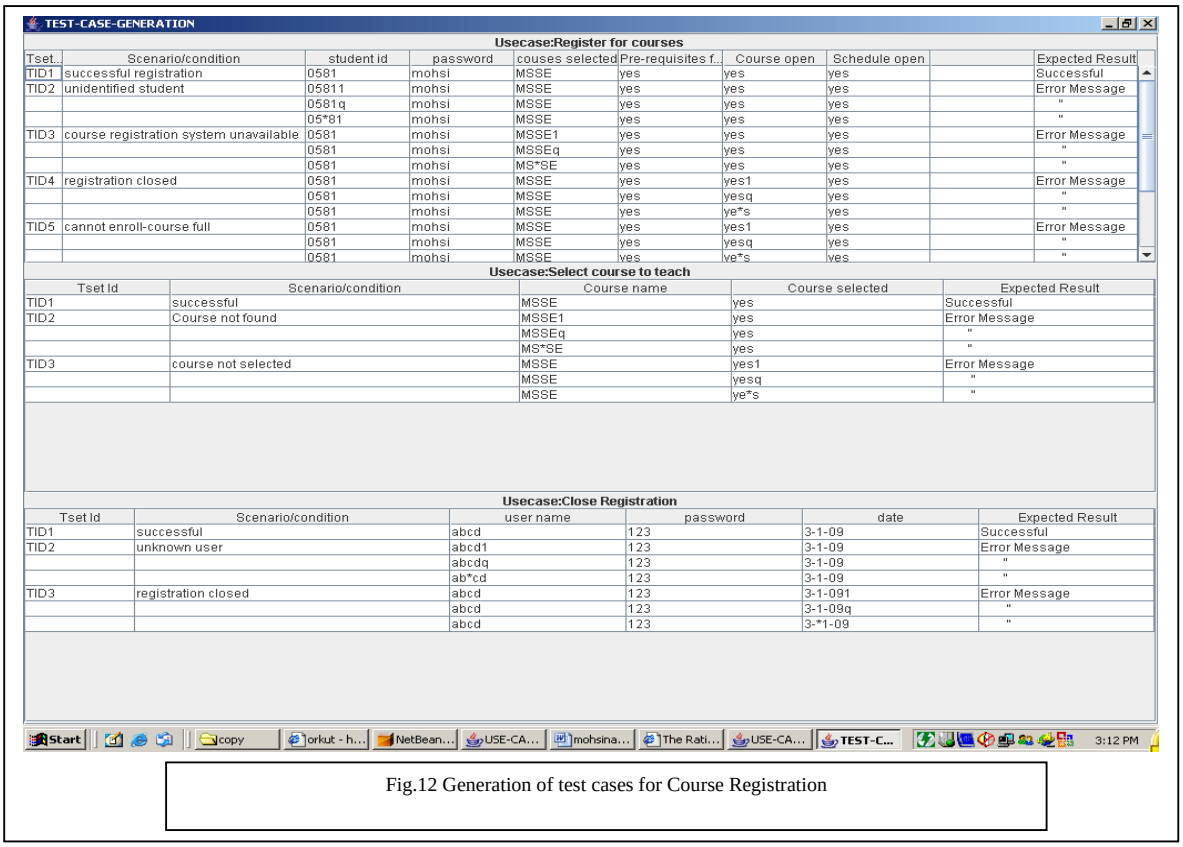

Fig.12 Generation of test cases for Course Registration

Next steps of our future work will include generating the scenarios for each use case automatically based on the information given in the use case description. By combining scenario-based requirement descriptions with system behavior descriptions for test case generation, the efficiency of the generated test cases can be increased.

14. References

[1] Jim Heumann,Generating test cases from use cases, The Rational edge-June 2001.

[2] M.Prasanna, S.N. Sivanandam R.Venkatesan R.Sundarrajan, A survey on automatic test case generation. Academic Open Internet Journal, Volume 15, 2005

[3] Noraida Ismail, Rosziati Ibrahim, Noraini Ibrahim Automatic generation of test case from use case diagram, Indonesia June 17-19, 2007.

[4] Erik Kamsties, Klaus Pohl, Sacha Reis, Andreas Reuys, Testing Variabilities in Use Case Models,published in van der Linden,F.(Ed.):Software Product-Family Engineering-5th International Workshop(Siena,Italy,November 2003)

[5] Jean Hartmann, Marlon Vieira, Herb Foster, Axel Ruder, UML-based Test Generation and Execution, jeanhartmann@siemens.com

[6] M. Balcer, W. Hasling, and T. Ostrand, “Automatic Generation of Test Scripts from Formal Test Specifications”, Proceedings of ACM SIGSOFT'89 - Third Symposium on Software Testing, Verification, and Analysis (TAVS-3), ACM Press, pp. 257-71, June 1990.