Scenario Based Software Testing Using UML Activity Diagram

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

Master of Engineering

in

Software Engineering

Submitted By

Md. Aamir Khan

(801331015)

Under the supervision of:

Mr. Vinay Arora

Assistant professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

JUNE 2015

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Scenario Based Software Testing Using UML Activity Diagram*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering/ Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Vinay Arora* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(Md. Aamir Khan)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Vinay Arora)
Assistant
professor, CSE
Department

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. S. Bhatia)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my guide, **Mr. Vinay Arora**, Assistant Professor, Thapar University, Patiala. I thank my supervisor for his time, patience, discussions and valuable comments. He has been very concerned and have aided for all the material essential for the preparation of this thesis report. His enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to **Dr. Deepak Garg**, Head of Computer Science and Engineering Department and **Ms. Damandeep Kaur**, P.G Coordinator, for motivation and inspiration that triggered me for the thesis work.

I also want to express my gratitude to **Dr. S.S. Bhatia**, Dean of Academic Affairs, for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field.

I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their help, cooperation and affection, which made my stay at Thapar University memorable.

Last but not the least, I would like to thank my parents and Almighty for showing me the right direction, without their blessings none of this would have been possible.

I take the sole responsibility of all the content that I have presented here in my thesis

Abstract

The software testing is one of the most critical phase under software development life cycle. Model based software testing is accepted by industries and researchers as a standard for developing the software design. UML 2.0 is widely used to model the functional requirements of the software under test. UML have various diagrams for describing the different aspects of the software system. In this thesis, we propose an automated approach using XML files and priority table to generate test scenarios from the UML activity diagram.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This chapter describes basic concepts related to the software testing, test cases, test scenario, Unified Modeling Language (UML) diagrams and method for generating the test scenario.

1.1 Software Testing

The process of software testing is to analyzing the software product to detect the differences between the existing and required conditions [1]. Testing is an important phase of the software development life cycle and plays an important role in assuring the software quality. It consumes tentatively 40% of the total effort devoted in the development of any software [2]. The effort for testing increases with the increase in the size of software, which makes exhaustive testing very time consuming [3].

Testing can be done before implementation and as well as after implementation of the whole software project. Model based testing is performed before the implementation, and this reduces the investment of the effort and time for software. Model based testing incorporates the usage of UML diagrams.

1.2 Types of Testing

1.2.1 Black Box Testing

Black box testing is defined as an approach for testing the software, where the functionality of software application is tested. In black box testing we don't really concern about the internal structure or working of the software application [4].

1.2.2 White Box Testing

White box testing is a method for testing the internal structure and working of the software application. For performing the white box testing tester must have the knowledge of technology that has been used for making the software application. Here, the testing process involves various coverage criterias [4].

1.2.3 Unit Testing

In this method of software testing an individual module of the software is tested. The unit testing is basically performed by the programmers and not by the testers, because it requires the detailed knowledge of internal program code and design [4].

1.2.4 Alpha Testing

Alpha testing is the method for testing the software at development site by users or by an independent test team. In this approach the developer observes the users and gives a look at the problems [4].

1.2.5 Beta Testing

In beta testing the software application is actually given to the intended users for the trial. The aim of beta testing is to uncover the flaws or issue if any from users point of view, which we don't want in our released version of software application [4].

1.3 Test Case

According to IEEE standard 160.12-1990 [5], a test case is a set of inputs, execution conditions, and the expected outputs to check whether the software application compliance with the required specification or not. The objective of test case is to uncover the error in the software product, improve the software quality that further reduce the software support and maintenance cost *etc.* Test cases for a program for checking the number is even or not are shown in Table 1.1.

Table 1.1 Example test cases in a program that checks the divisibility of number by two

Test case (input)	Execution condition	Anticipated output
4	Check Inputted number is even or not	Even
5		Odd
3		Odd

1.4 Test Scenario

Test scenario can be thought of as sequence of events in the software system, which describes the overall behavior. Test scenario can be derived from different UML diagrams like use case, collaboration diagram, sequence diagram and activity diagram. Each test scenario can be said to be the representative of different requirement goals of the software system**.** Test scenario for an example activity diagram shown in Figure 1.1 is depicted in the Table 1.2.

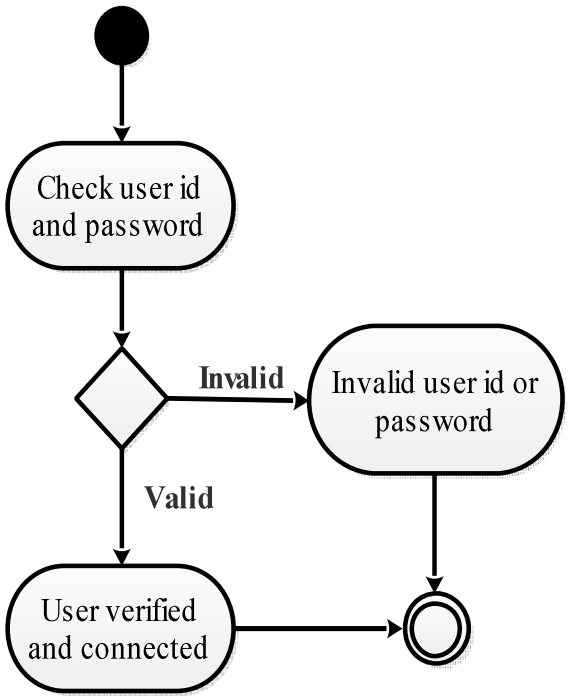


Figure 1.1 Activity diagram for user login module [6]

Table 1.2 Test scenarios *w.r.t* an activity diagram given in Figure 1.1

Serial no.	Test scenario
1	Start-> Enter user id & password-> Check pin-> Invalid user id or password-> Enter user id & password-> Check pin-> User verified and connected-> stop
2	Start-> Enter user id & password-> Check pin->User verified and connected-> stop

1.5 UML Diagrams

UML [7] is a standard way to represent the system in the form of graphical constructs, which makes easy to interpret the different aspects of the system. These models can be directly transformed in any programming language to build the code. UML can also be used for drafting the requirement specifications for the domains like banking, finance, internet, healthcare, *etc.* UML diagram that can be used for generating the test scenario are described below.

1.5.1 Sequence Diagram

Sequence diagram is a variant of interaction diagram that is modeled to show how the different processes for the system interaction. Here, the object interactions are arranged on the basis of the time sequence. Objects used in the diagram should be named, life line represents the time for which the objects are eligible to communicate with each other [8]. An example of sequence diagram is shown in Figure 1.2.

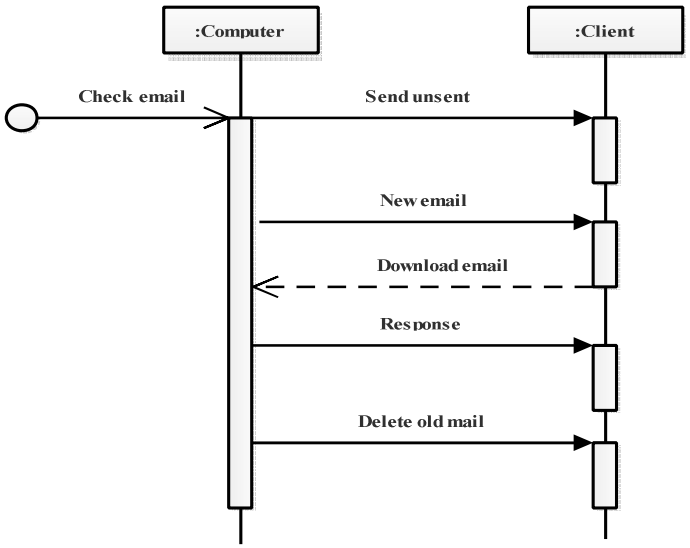


Figure 1.2 Sequence diagram for e-mail message sequence [8]

1.5.2 Collaboration Diagram

Collaboration diagram is used to model the organization of different objects in conjunction with the send and receive message passing construct. Collaboration diagram can be thought of as isomorphic to sequence diagram, except the presence of life lines. An example of collaboration diagram is shown in Figure 1.3

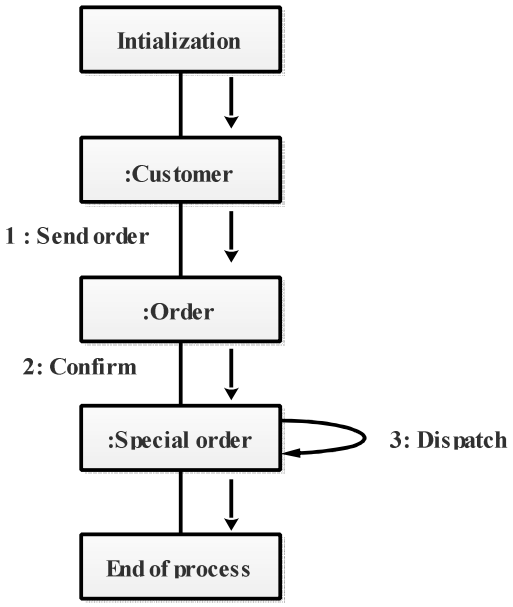


Figure 1.3 Collaboration diagram for an order management system [9]

1.5.3 Statechart Diagram

Statechart diagram is used to represent the dynamic aspect of the system under development. Statechart diagram shows the different states of an object during its whole lifetime and events that make the object to change their state. This makes Statechart diagram more suitable for modeling reactive system, where working is totally dependent on the external events [10]. An example Statechart diagram is shown in the Figure 1.4 below.

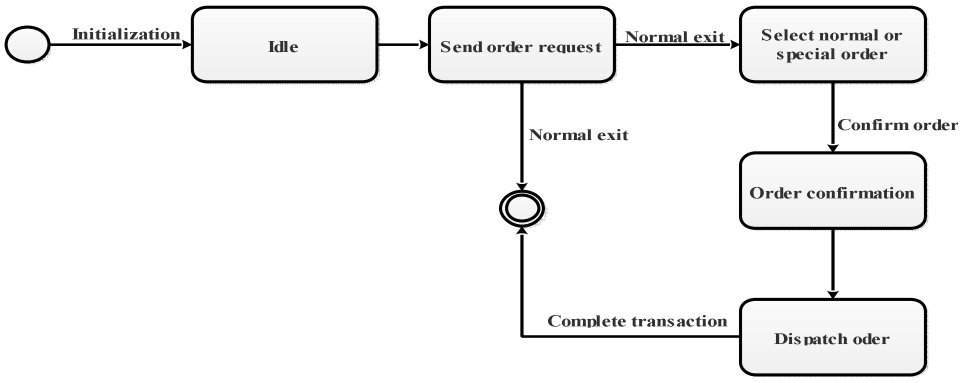


Figure 1.4 Statechart diagram for an order management system

1.5.4 Activity Diagram

Activity diagram is used as a graphical model to represent the dynamic behavior of a software system. Activity diagram can depict sequential, loop, branch and concurrent flow of execution in the system. An example for activity diagram is shown in the Figure 1.5 below [11]. The customer can order product using software application. After placing an order for one item the system will direct the user to the module related to re-order item/s. If user is not intended to purchase more item/s then the system will direct the customer to the billing module. Here, the information about the items and shipping information is collected. At last the final bill will be generated. The information gathering activities will execute simultaneously. After collecting the information, validation of bill information will be done. If information is not valid then system activates the module to collect the billing information. After completion of the billing process the receipt is generated for the purchased item. After this the order get assembled and finally shipped by the system.

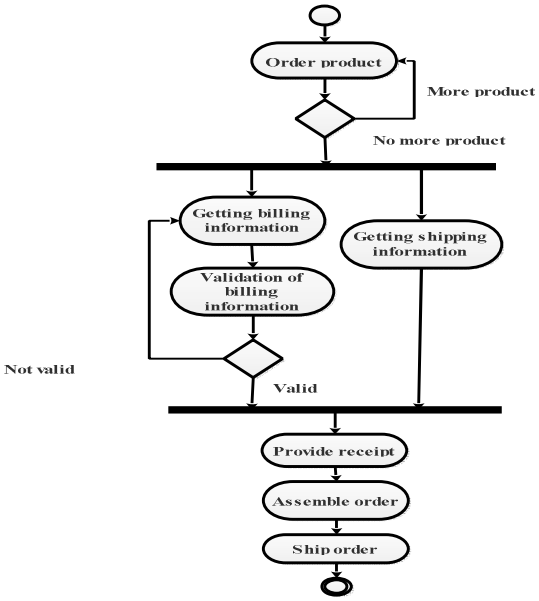


Figure 1.5 Activity diagram for purchasing items [11]

1.6 XML

XML stand for extensible markup language, where markup language means language consists of different tags. It defines set of rules for encoding that are needed to be followed while documentation. The XML code is human as well as machine readable

format. XML is greatly used as an intermediate representation of the UML diagram. The XML document is simple, useable, general and used across the internet.

```
<url>
 <loc>http://www.example.com/en</loc>
  <xhtml:link
    rel="alternate"
    hreflang="gr"
    href="http://www.example.com/gr" />
  <xhtml:link
    rel="alternate"
    hreflang="en"
    href="http://www.example.com/en" />
</url>
<url>
 <loc>http://www.example.com/gr</loc>
  <xhtml:link
    rel="alternate"
    hreflang="gr"
    href="http://www.example.com/gr" />
  <xhtml:link
    rel="alternate"
    hreflang="en"
    href="http://www.example.com/en" />
</url>
```

Figure 1.6 Example of XML code [12]

Chapter 2
Literature Survey

Test cases and test scenarios are an important artifact for testing the software system, and it should be generated from the artifacts like code or the UML model. UML has been accepted as a standard by many software organizations which is used to model the functional requirement of a system. Model based testing is defined as the process of generating the test cases and test scenarios from UML diagram. Model based testing is significant as it locates the error/s in the design phase (beginning of the development process) itself and prevents them from propagating to the later stages of the software development life cycle. In this chapter a systematic review of the research papers have been done to show various available techniques for generating the test cases and test scenarios from UML activity diagram. Table 2.1, summarize various viable, open source and freely available tools for test case and test scenario generation (that use activity diagram).

2.1 Test Case Generation

This section presents the various findings (in brief) present in the research papers related to generation of the test case from UML activity diagram.

2.1.1 Using Depth First Search

R.K. Swain *et al.* [13] proposed a method for generating the test cases from UML Activity diagram in which, an activity flow graph (AFG) was first derived from an activity diagram. And DFS was used to generate the test cases. Researchers mentioned the combination of sequence diagram with an activity diagram as the future work in the direction proposed in [13]. An example activity diagram and its corresponding activity flow graph have been shown in the Figure 2.1 and Figure 2.2 respectively.

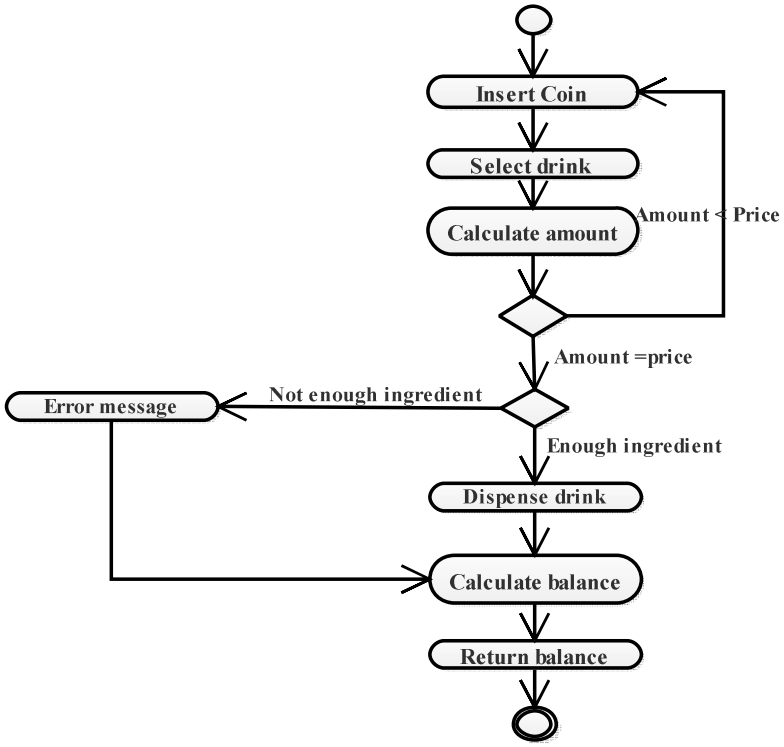


Figure 2.1 Activity diagram for purchasing beverage [13]

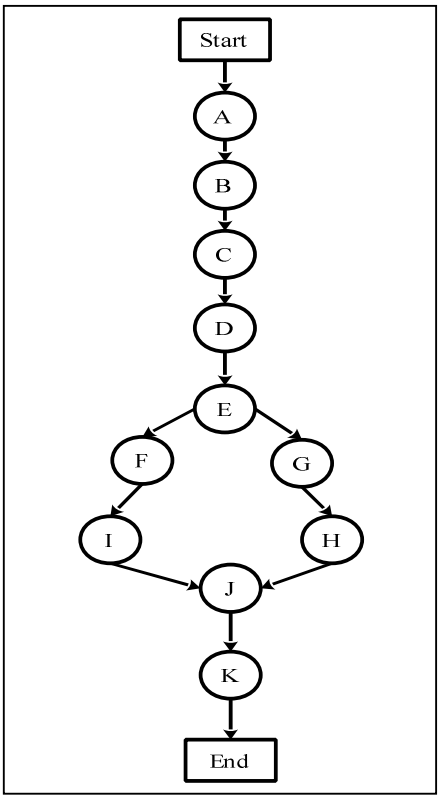


Figure 2.2 Activity flow graph for purchasing beverage activity diagram [13]

X. Chen *et al.* [14] proposed a method that automatically generates test cases from UML activity diagram. Proposed approach constitutes three sub-phases, where, the classes of java program were mapped to the corresponding activity nodes in the activity diagram first and then, the code was instrumented. Finally, a data classifier has been constructed. This data classifier used as a feedback, which helps in creation of new test inputs for covering untouched paths. The researchers used simple activity path coverage criteria to generate the test scenarios.

C.A. Sun *et al.* [15] proposed a method for generating the test cases from an UML activity diagram for the concurrent applications. Here, activity diagram was firstly got converted into an intermediate form called Binary Extended Tree (BET). DFS was applied on BET to generate test scenarios. To derive the test cases for each scenario the category partition method has been used. As a future work, researches planned to extend the approach by incorporating the transformation rules. An example of a BET is provided in the Figure 2.3 below.

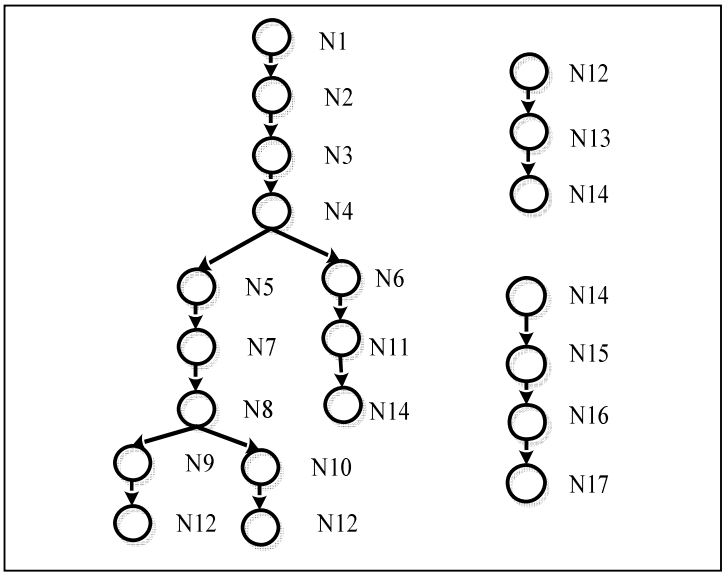


Figure 2.3 Binary extended AND_OR tree [15]

W. Linzhang *et al.* [16] proposed a method to generate the test scenario/s from UML diagram activity diagram that used the basic path coverage criteria. For finding the basic path from the activity diagram, a retrospective DFS algorithm was used. After generating the test scenarios, the grey box method was applied to extract all the input sequence/s, object method sequence/s and the output sequence/s.

P. Boghdady *et al*. [17] proposed a method for generating test cases from UML activity diagram. Here, activity diagram was firstly got converted into an XML file, which further generate an activity dependency table (ADT). Using this ADT, an Activity Dependency Graph (ADG) was generated. Test paths were then generated with the help of ADG and ADT. The generated test paths were validated using the concept of cyclomatic complexity. Finally, the validated test paths were used to generate the final test cases. Snapshot of the approach for generating the test cases (using ADG and ADT) is provided in Figure 2.4 below.

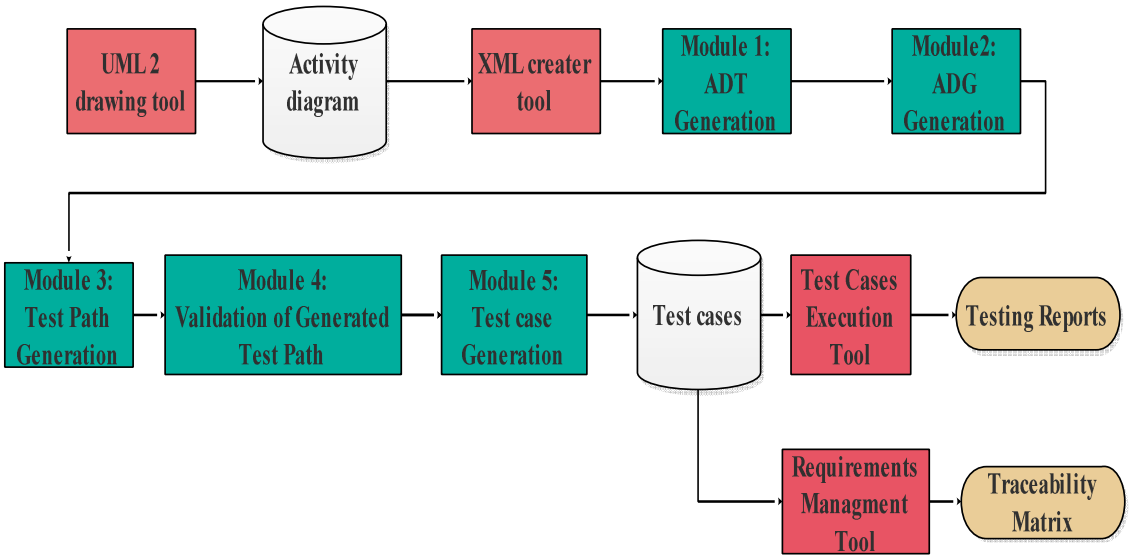


Figure 2.4 Approach for generating test cases that use ADG and ADT [17]

P. Boghdady *et al*. [18] introduced the methodology for generating the test cases where, a XML file is first generated from an activity diagram, and an activity dependence table (ADT) was generated from the XML. Using ADT researcher generated the activity dependence graph (ADG). Using depth first search on ADG all basic paths were extracted. For validating the test path cyclomatic complexity was used that provided the lower bound related to the count of generated tests for satisfying the hybrid coverage criteria. At last the test cases were generated using the test path. Count of test cases was reduced after performing the validation check. Figure 2.5 depicts the steps for generating the test cases as given by the researchers in [18].

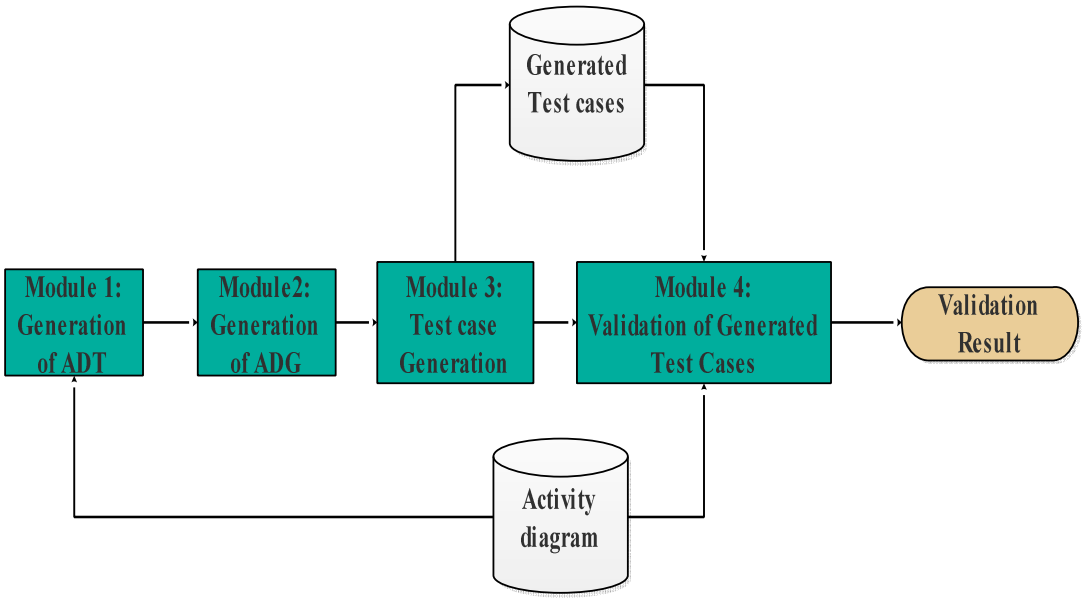


Figure 2.5 Steps for generating the test cases [18]

H. Kim *et al.* [19] introduced a method in which an activity diagram was first got converted into an input output explicit activity diagram (IOAD), which further got converted into a directed graph. For generating the test case/s, the DFS was used on the directed graph. Approach provided by the researchers here suffered from a serious drawback related to the path explosion.

Using condition-classification tree method, S. Kansomkeat *et al.* [20] proposed an approach to generate the test case/s from an UML activity diagram. Here, the control flow information related to the guard condition and the decision point were gathered to construct a condition-classification tree. Test cases were generated using DFS on the condition-classification tree structure.

2.1.2 Using Breath First Search

X. Fan *et al.* [21] provided a method using an activity diagram composition tree (ADCT). ADCT represents the hierarchy of different activity diagrams. Activity diagram represented at the leaf level was considered as a sub activity of the main activity diagram (at the level above it). For each activity diagram the test cases were generated using the composition tree by using a bottom up approach. Where, the test case for the sub-activities were created first and followed by the parent activity

diagram. An example composition tree for an activity diagram has been shown in the Figure 2.6 below. Researcher commented on the adequacy improvement as future scope for the proposed approach.

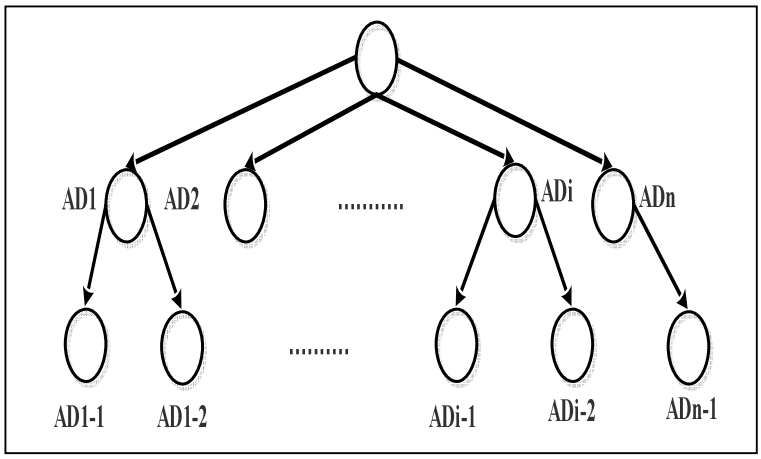


Figure 2.6 Composition tree for an activity diagram [21]

2.1.3 Using Modified Depth First Search

C. Mingsong *et al.* [22] proposed a technique for generating the test cases for the JAVA programs. Here, the generation of test cases includes usage of the program execution trace (PET) and coverage criteria.

R. Chandler *et al.* [23] proposed an approach that used an activity diagram and convert it into an XMI file format. DFS was applied to the XMI file to generate the test scenarios. Approach presented by the researchers here, removed the path explosion problem. Technique given by the researchers was effective and efficient in the usage based scenario.

B.N. Biswal *et al.* [24] proposed a method for generating the test scenario/s from an activity diagram. Here, the DFS algorithm (in conjunction with sequence diagram and class diagram) was used to achieve the test case adequacy. To search for the setting and interaction categories the category partition method to analyze the sequence diagram and class diagram. Approach proposed by the researchers was capable for handling the complicated fork and join pairs efficiently. Figure 2.7 given below depicts the process for generating the test cases using an activity diagram.

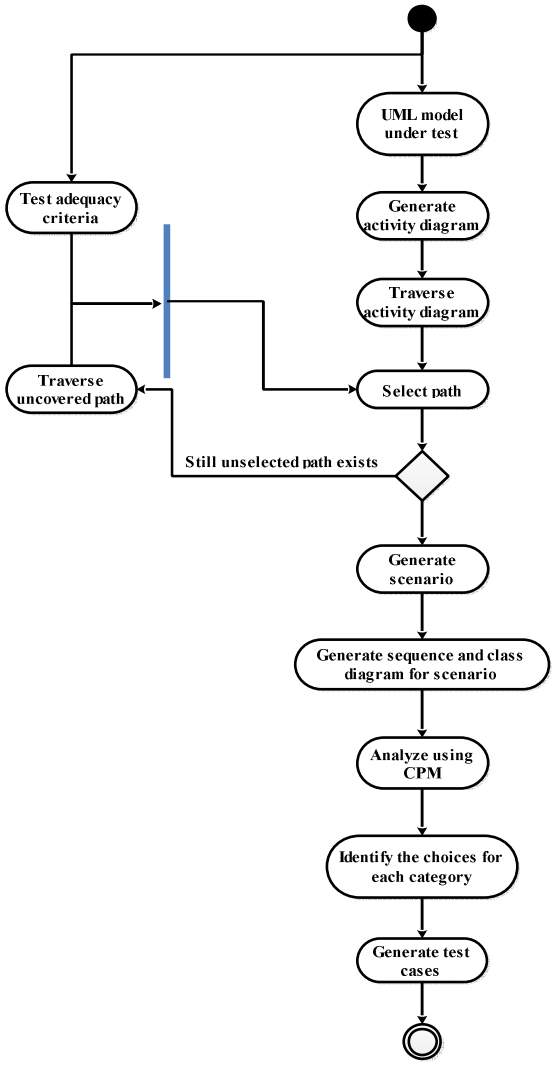


Figure 2.7 Test case generation using an activity diagram [24]

2.1.4 Using Heuristic Approach

K.H. Chang *et al.* [25] proposed a method for generating test cases for the JAVA program. Researcher first randomly generated the large number of test cases and then compared it with Program Execution Trace (PET) according to the simple coverage criteria then they produced the reduced number of test cases. PET derived by instrumenting the code with test cases to get the traces. The heuristic rule based approach allows the frame work to include more test case generation knowledge and test requirement.

A.K. Jena *et al*. [26] proposed an approach for generating the test cases from an activity diagram, where an activity flow table and an activity flow graph were created and DFS was applied to find out the test paths and generate test cases for corresponding test paths. In last the genetic algorithm was applied to optimize the test cases. Figure 2.8 and 2.9 depicts the activity diagram of ATM cash withdrawal and its corresponding flow graph respectively.

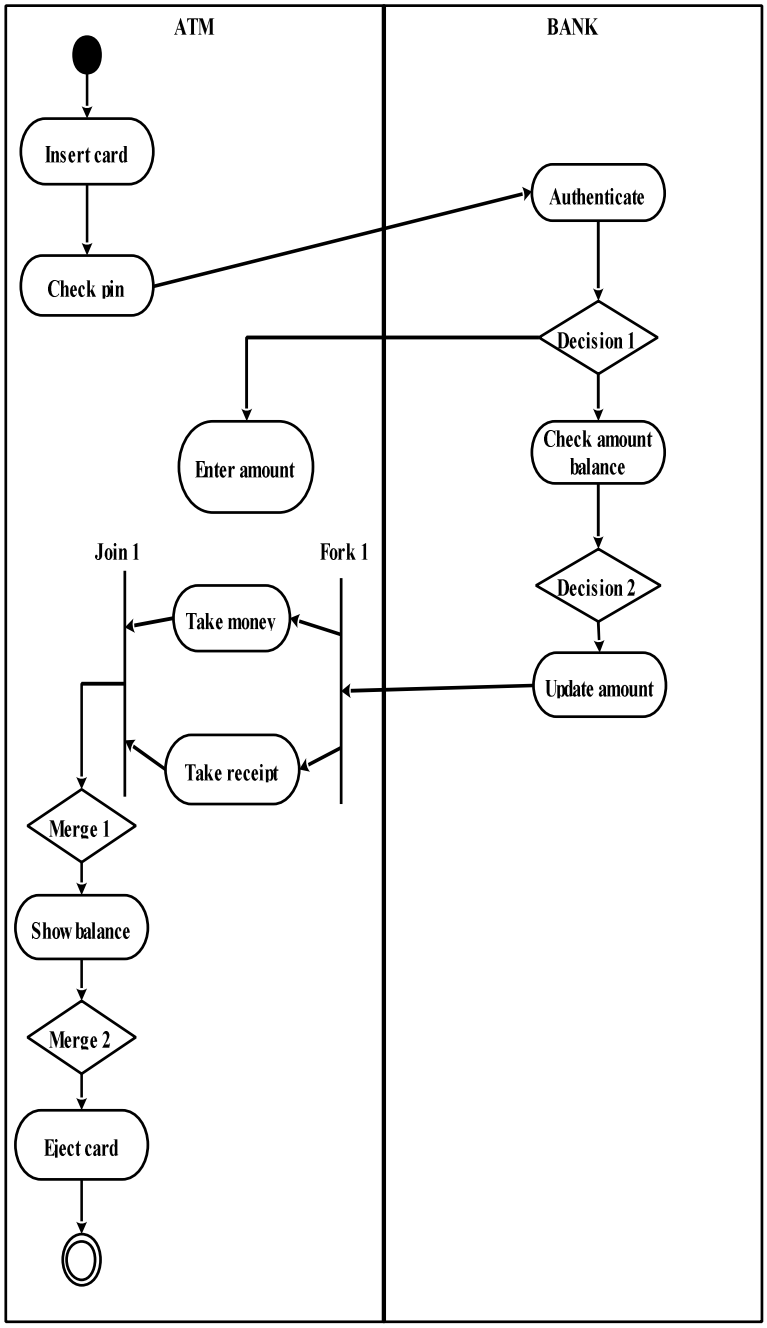


Figure 2.8 Activity diagram for ATM Machine [26]

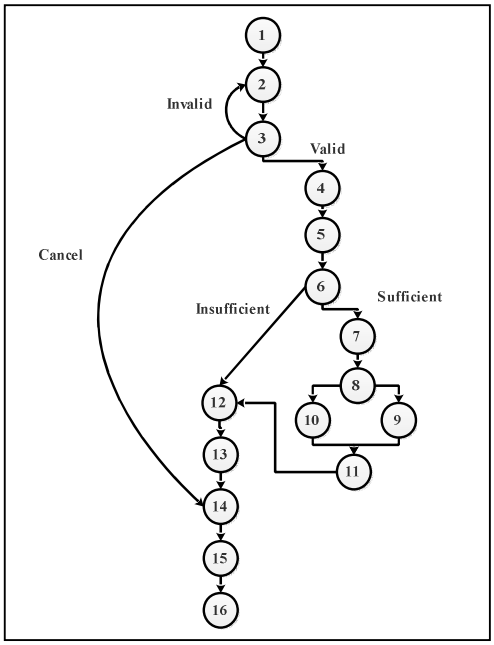


Figure 2.9 Flow graph for ATM machine activity diagram [26]

2.1.5 Other Approaches

P.E. Patel *et al.* [27] compared the two algorithms that correspond to the generation of the test case/s from UML activity diagram. Researchers had used the coverage criterion called activity coverage criterion.

L. Li *et al.* [28] proposed a method to generate the test cases from UML activity diagram using extension theory (extenices). In this an activity diagram was first converted into an Euler circuit. After this an Euler circuit algorithm was used for finding the basic coverage criteria like transition coverage and activity coverage. Extenices theory reduced the number of test cases without affecting the fault detection percentage.

S. Tiwari *et al.* [29] proposed an approach for generating the validation test cases using software fault tree analysis (SFTA). In this a UML activity diagram was transformed to a software success tree (SST). SST further got transformed into a software fault tree (SFT), on which the MOCUS algorithm was applied to get the minimum cut set. Figure 2.10 represents the snapshot of the approach proposed in [29] for generating the test cases.

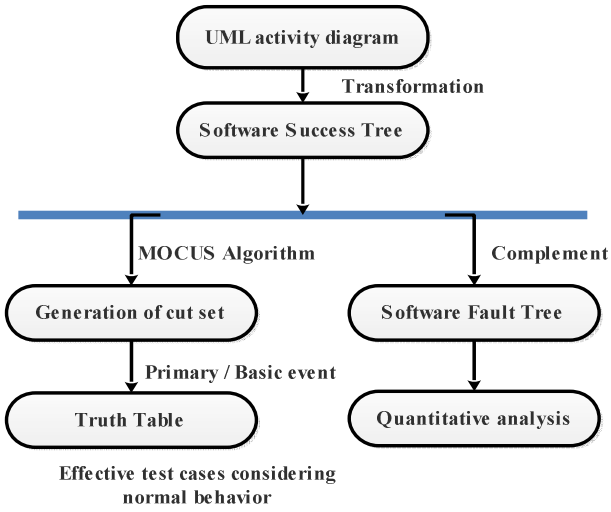


Figure 2.10 Approach for generating test cases [29]

2.2 Test Scenario Generation

This part of the chapter presents the literature survey on generating the test scenario/s from UML activity diagram.

2.2.1 Using Depth First Search

H. Mohanty *et al*. [30] used UML activity diagram for generating the test scenarios for the concurrent system, where activities are supposed to be of interleaved nature. Priority based algorithm was used for generating the test case.

P. Nanda *et al.* [31] also proposed a method to generate the test scenario from UML activity diagram. The approach involved converting the activity diagram into flow chart first and then traversal of the flow chart using path coverage criteria. To cover all the path of the activity diagram DFS algorithm was applied.

P. Kaur *et al.* [32] proposed an approach for prioritizing the test scenarios. In this approach an activity diagram was converted into a control flow graph. DFS was applied to generate the test scenarios that satisfy the basic path criteria. In this process some redundant paths were generated, and to remove the redundant paths breath first search was applied. After collecting all non redundant test scenarios, priority was assigned to each test scenario based on the path complexity. Figure 2.11 and 2.12

represents an activity diagram of shipping order and its corresponding control flow graph respectively.

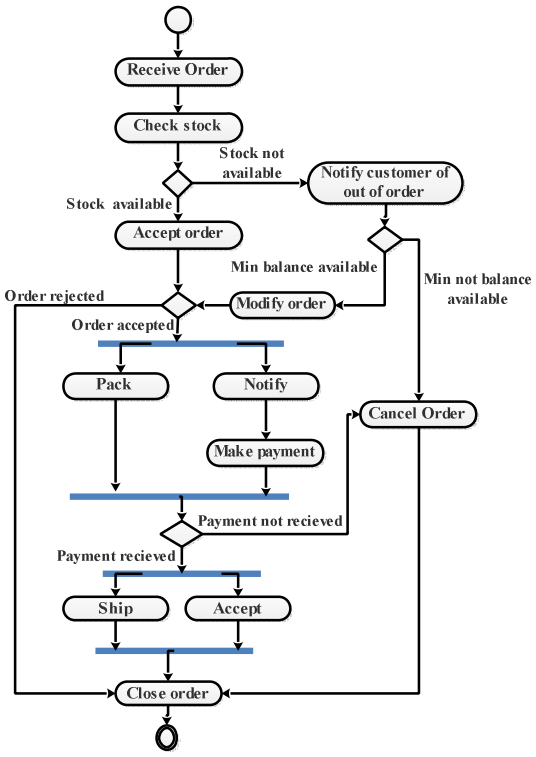


Figure 2.11 Activity diagram for shipping order system [32]

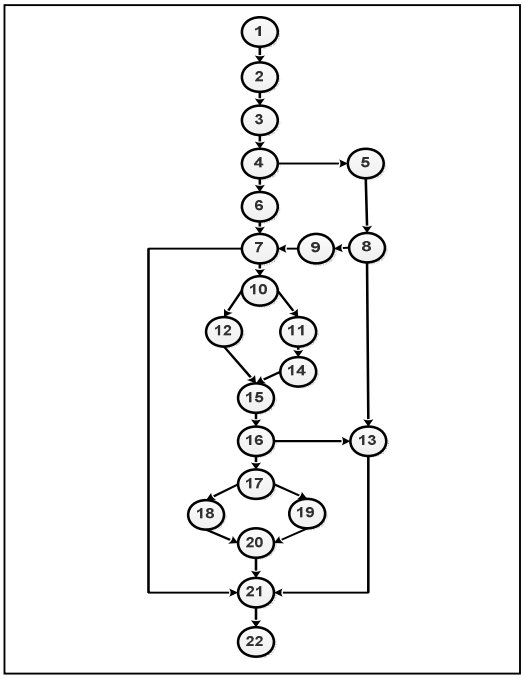


Figure 2.12 Flow graph for shipping order system [32]

2.2.2 Using Heuristic Approach

D. Xu *et al.* [33] proposed a method to generate a test scenario from UML activity diagram using an adaptive agent. The researcher presented the proposed approach by taking an example of a bacterium called Maxococcus Xanthus.

2.3 Testing Tools

This section gives an overview related to the tools available for generating the test scenario as well as test cases (tools mentioned here are categorized into commercial, free and open source). Using these testing tools we can generate test cases and verify the models.

Table 2.1 Testing Tools

S. No.	Tool Name	Description
1.	BEACON for Simulink (B4S) [34]	• B4S is a product of Applied Dynamics International. • It efficiently generates safe code for low risk application as well as for critical application. • B4S can generate code and also do have the modeling power as well.
2.	Safety Test Builder [35]	• Safety test builder is a product of chaisTek. • Automatically generates the test cases for simulink and Sateflow based implementation. • Reliability of safety test builder helps to reduces the testing time
3.	Simulink Design Verifier [36]	• By MathWorks. • Use formal method in model based design and error detection. • Generate test case after analyzing the algorithm and logic in simulink and stateflow model.
4.	T-VEC [37]	• Generate test cases and analyze the model constraint using an advance algorithm. • Test cases are effective for both decision and computational errors. • Helps in analyzing the simulink model
5.	J-Unit [38]	• J-Unit is a framework used to write repeatable tests. • It is a unit testing framework.
*6.	AgitarOne Agitator [39]	• By Agitar Technologies • Developed for testing java application • Provide better way to understand user code
7.	Java Path Finder (JPF) [40]	• It is an model checker, that is used for finds bugs in java programs • It is a framework for runtime java verification.
8.	TefKat [41]	• It is suitable for data transformation and Model Driven Development. • TefKat plug-in includes a source level debugger and a syntax

		aware editor
9.	MOFScript [42]	• MOFScript tool is used for model to text transformation. • It is metamodel-independent languages that use any kind of metamodel and its instance for test generation. • Open source.
10.	ConTest [43]	• By IBM • Discovers and eliminate bugs related to concurrency from distributed and parallel software. • By discovering bugs early in testing process it helps in improving quality and cost of development.
11.	Quick Test Professional (QTP) [44]	• By HP • It an automated tool for functional testing • It is suitable for regression testing.
12.	CHESS [45]	• By Microsoft • It is used to discover and regenerate heisenbugs. • It is suitable for stress testing.
13.	UMLTGF [16]	• Automatically generates test cases from UML activity diagram. • UML specification is easily imported and parsed.
14.	TSGDA [31]	• Automatically generate test scenario from UML activity diagram. • Uses heuristics approach.
15.	TSGen [11]	• Automatically generate test scenario from UML activity diagram. • It includes different coverage criteria.
16.	AGTCG [22]	• User can edit, analyze and construct activity diagram. • Automatic test case generation tool. • Java program is instrumented according to given activity diagram.
17.	UMLTOTC [28]	• Tool is developed in C#. • Generate test cases from UML activity diagram. • Works on the principle of Euler circuit.
18.	NuSMV model checker [46]	• NuSMV is an extension of SMV. • Open source. • Developed for verifying industrial design.

Chapter 3
Gap Analysis and Problem Statement

This chapter tells about the gaps encountered during the research by reviewing the already existing literature in the area of automatic test scenario generation from UML activity diagram. Based on the gap analysis, the problem statement has been framed for carrying the already existing research work in another and yet unexplored direction.

3.1 Gap Analysis

Based on the literature survey related to the test scenario generation using an UML activity diagram, the following gaps have been identified:

- A thorough thought should be given for devising an efficient algorithm that can accommodate the test coverage criterias like transition coverage and activity state coverage [27, 28].

- An approach can be formulated that may efficiently reduce the space complexity while generating the test scenarios from an UML activity diagram [30].

- An approach for generating the test cases from UML activity diagram can be extended to work with other UML diagrams like statechart, sequence and collaboration diagram as well [13, 26].

3.2 Problem Statement

After a thorough review of the literature related to the generation of test case/s, generation of test scenario/s, and the prioritization of test scenario/s in conjunction with UML activity diagram, this has been analyzed that the space complexity used by algorithm can be reduced for storing the information *w.r.t* activity diagram further. Already existing techniques include the usage of a 2-D array for the storage, which is sparse in nature, which leads to wastage of storage space. Proposed approach has replaced the usage of a 2-D array by an adjacency list. An adjacency matrix (2-D

array) takes the memory of an order n×n, whereas the memory usage for an adjacency list is of the order (E+V), where E is the number edges and V is the vertices, which may save the required memory space [49]. In case of the concurrent activities, the proposed approach has by passed the explicit storage of the various combinations present in a concurrent module by implicitly generating the combinations during path generation itself. This activity may reduce the storage needed for the concurrent system and hence trim down the space complexity.

Chapter 4
Methodology

The technique proposed for generating the test sequences from a UML activity diagram involves the steps as shown in figure 4.1. The approach proposed here is an extension of the technique proposed in [30]. The proposed techniques here is depth first search with a module for handling fork and join nodes. Experimental work conducted on various subject systems prove that the space complexity of the algorithm proposed here is less as compared to the approach presented in [30].

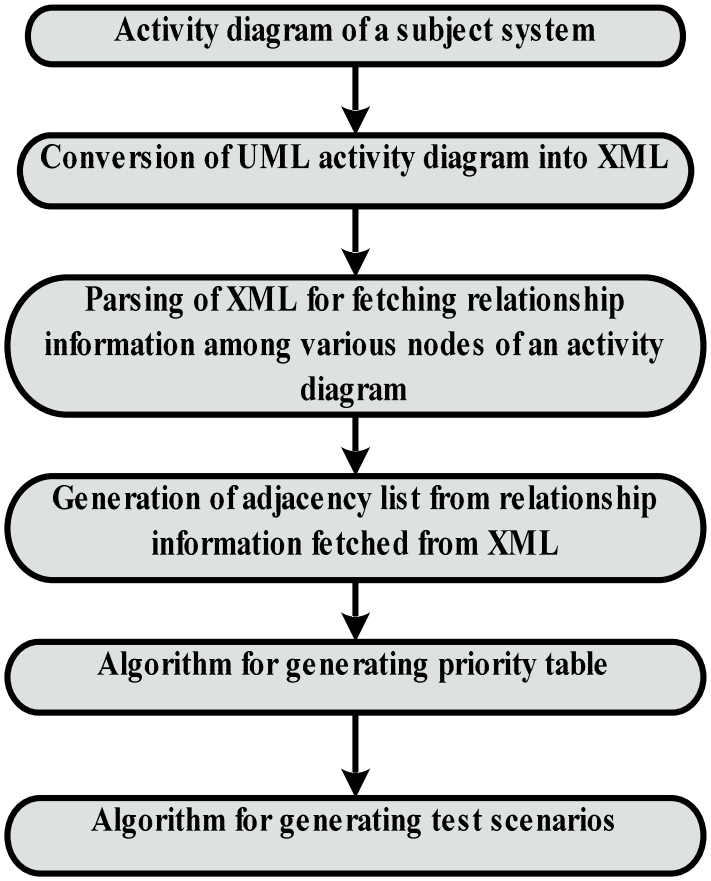


Figure 4.1 Steps under the proposed approach

4.1 Steps Under Proposed Methodology

4.1.1 Generation of an Activity Diagram for a Subject System

Dynamic behavior of a system can be described by using a UML activity diagram and can be depicted as a flow chart, which shows the flow of execution in a program.Tools like Visual paradigm, Rational rose, UMLet, StarUML, Altova, *etc.* are available for generating the UML diagrams. In the proposed approach, Visual paradigm11.2 tool has been used for generating the activity diagram for a particular scenario.

4.1.2 Conversion of a UML Activity Diagram Into Its Respective XML Representation

Although UML diagrams are language independent for depicting the scenario available in the document having functional requirement, but these cannot be traversed for fetching any kind of information. In the proposed approach, UML is converted into its equivalent XML for retrieving any particular information. Visual paradigm 11.2 has been used for converting a UML activity diagram into its corresponding XML code.

4.1.3 Traversal of XML for Getting the Source and Destination Node for Each Activity Node Present In the Activity Diagram

In the proposed approach, the XML file as generated through Visual paradigm is parsed using the user defined module developed in Java 7 (using JDK 1.6.2). Algorithm 1 given below represents the pseudo code used for traversing the XML code and generating the array list having the information related to the source and destination node for all the nodes available in the activity diagram.

Algorithm for fetching source and its respective destination node/s from XML

List of variables:
source [ ]: Array list used to store the source node.
destination [ ]: Array list used to store the destination node

Input: XML code corresponding to an activity diagram
Output: Activity dependence table (ADT)

1. Declare two array list namely source [ ] and destination [ ] for storing the source node and destination node respectively.
2. Traverse XML code and for each node of an activity diagram, look for *from* and *to* identifiers and append the values obtained in the respective array list *i.e.* form entries will go into source [ ] and to will reside into destination [ ].
3. For each node source [ ] and destination [ ] array list will constitute an ADT.

4.1.4 Generation of Adjacency List

Adjacency list is a representation of graph is a collection of unordered list, one for each node in a graph. Each list shows the set of neighbor of its node [wiki]. Approach proposed here uses adjacency list to represent an activity diagram as graph, which make the traversal easier. Here, hash map has been used to create an adjacency list.

Algorithm for generating adjacency list

Variable list:

AL<key, value> = hash map
key = activity name
valueOf = function used for fetching value present at a particular index
Index = integer to store index
NL = name list

Input: source [ ], destination [ ], NL
Output: Adjacency list (ALST)

1. For each activity node in NL do
2. Add the node as key to AL<key, value>
   Look for the *index* of source [ ], where of key entity == *valueOf* (source [index])
   If *index* found
   Look for the value at the same index in destination [ ]
   Add value in the ALST for selected key

End For

4.1.5 Generation of the Priority Table

Priority Table (PT) is a data structure used for storing the priority of the activity nodes. Here, PT is generated using an adjacency list of an intermediate activity diagram, which will be used for the validation of generated test scenario.

Algorithm for generating priority table

Variable list:

1. PT<key, value> is a hash map where, key = activity name, and value = priority
2. AL<key, value> is a hash map, where key = activity name, and value = list consisting of all activity nodes connected to key
3. Priority = An integer used to assign priority to activity nodes
4. NL = Name list
5. NID = Id of node (present in an activity diagram)

Input: Adjacency list, NL, NID
Output: Priority table

```
1.      For each node in NL do
2.        Look for the index of the selected node
3.          Based on similar index, Get the node from NID
4.            Get the Id of the node obtained in step 3.
5.      If node is fork (F) do
            5.1    Set X ← node/s in the list AL[ ] of F do
            5.2 For each x in X
            5.3   If x not in AL[value] of node_type = accept, then
            5.4          node[ ] = x
            5.5          PT[x]=priority
            5.6   End if
            5.7  End for
            5.8  Assign same priority number to the elements present in the array node [ ]
            5.9    Select any element from node [ ]
            5.10       While selected_element != Join do
            5.11       Assign priority to the nodes present in the AL [selected_element],
                              p [ ] = priority +1;//storing all the elements in the list of selected_element.
            5.12      selected_element = select any element from AL [selected_element]
            5.13       End while

6.      End if
7.      End for
```

4.1.6 Generation of the Test Scenario

Test scenario is defined as Test scenario is a statement that guides what to test and used to generate test cases. Test scenario can be easily derived from use cases and they are classified as high level test requirement. Use case diagram, sequence diagram, collaboration diagram, and activity diagram can be used to generate the scenarios.

Algorithm for generating test scenario/s

List of variable
PT<key, value> = hash map priority table that stores the priority of nodes.
AL<key, value> = hash map adjacency list that used to represent activity diagram as graph.
Path [ ] = array used to store the nodes that resulted in the test scenario/s

Input: PT<key, value>, AL<key, value> *w.r.t.* activity diagram
Output: Test scenarios

1. Fetch start node from AL
2. path[ ] ← Add node
3. If node is stop (ending) node then
4. Go to step 17
5. Else // when node is not a stop or ending node
6. Flag for the selected node = (active and not processed)
7. If n ∈ N ≠ fork then
8. For all nodes (K) ∈ list from AL *w.r.t* node n (active and not processed) do
9. Change Flag of node n to (active and processed)
10. Go To step 2
11. End for
12. End if
13. If n ∈ N = = fork then
14. Path [ ] ← Nodes (K) present in the list from AL [ ] of fork node (n) and go to step i.
    i. Generate different combinations for Nodes (K) present in the list from AL [ ] of fork node (n)
    ii. For each generated combination
    iii. Fetch the last element of path [ ] and Go to step 2.
    iv. End for
15. End if
16. End if
17. Look for the fork node/s in path [ ]
18. If fork found then
19. Look for the index value of the fork node
20. Start from the index obtained above and loop while node != join do
21. If priority of path[i] < path[i+1] then
22. Continue

```
23.              Else
24.                Set flag=0;
25.                break;
26.              End if
27. End while
28.   If flag = = 0 then
29.        Invalid path
30.   Else
31.       Vaild path
32.   End if
```

Chapter 5
Implementation and Results

For the methodology proposed in the Chapter 4, implementation has been done using Java 7 (jdk1.7.0_79), to generate the test scenario from a UML activity diagram. Snapshot *w.r.t.* each step has been mentioned below with a brief explanation.

5.1 Generation of UML Activity Diagram

Tool named Visual paradigm 11.2 has been used to generate the required activity diagram from requirement specification.

Figure 5.1 Original activity diagram

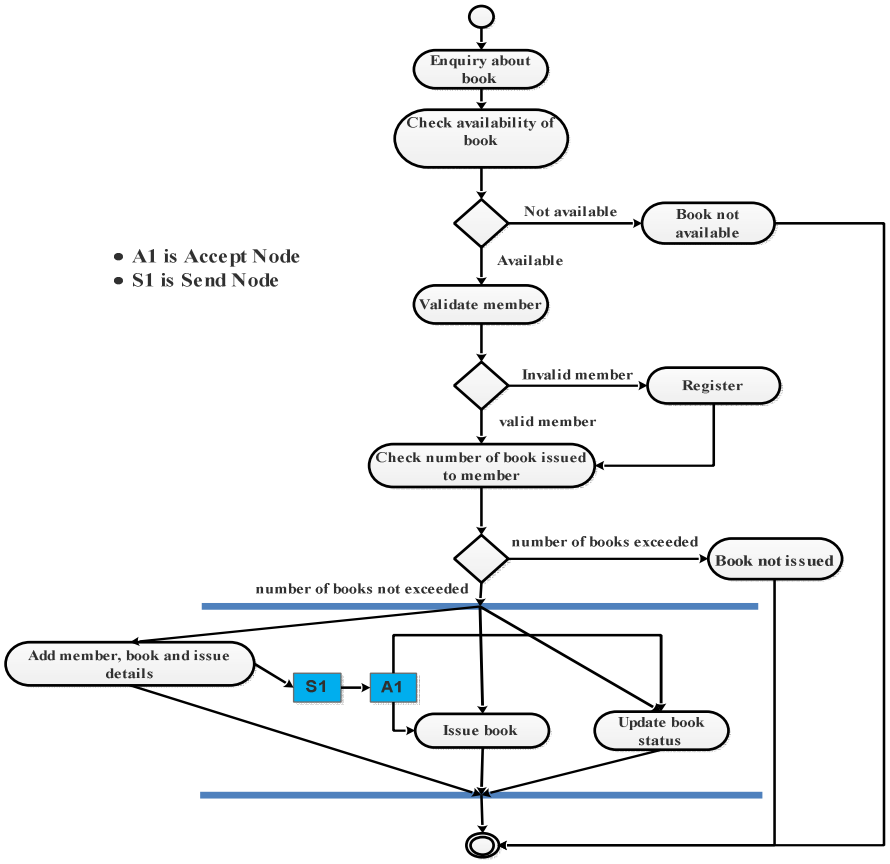


Figure 5.2 Intermediate activity diagram

5.2 Generation of XML

Visual Paradigm has inbuilt function that export the XML file for that particular activity diagram.

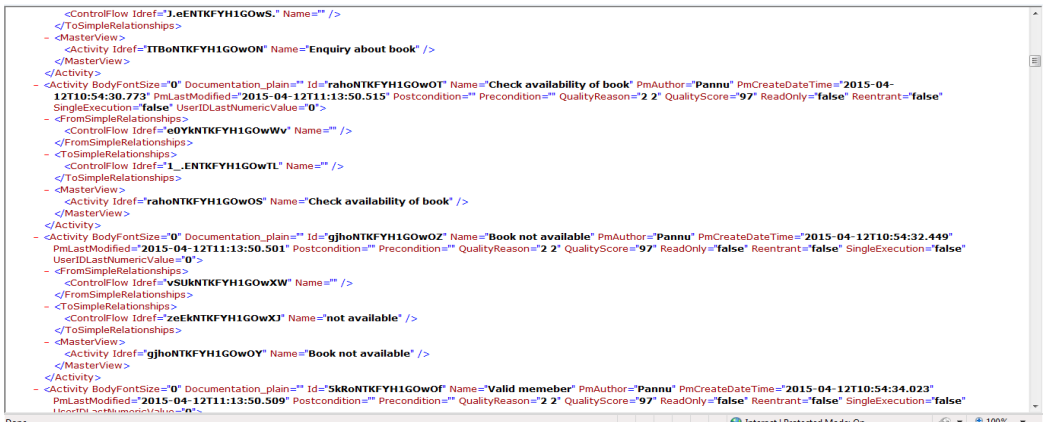
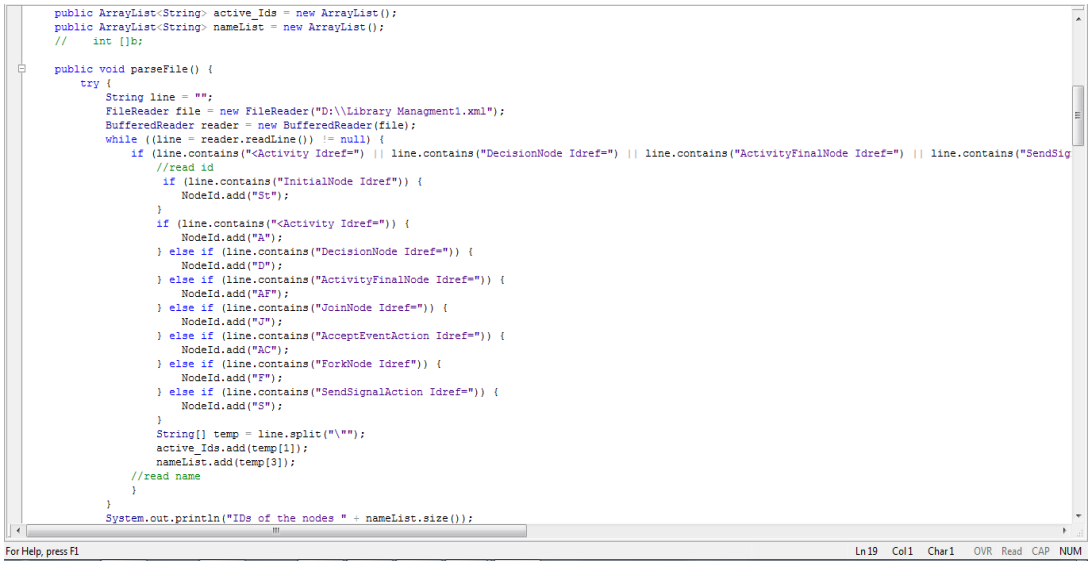


Figure 5.3 XML file of activity diagram

5.3 Module for Parsing Xml

This module of the program parses the XML file and extracts the information like nodes and relationship among the nodes.



```
public ArrayList<String> active_Ids = new ArrayList();
public ArrayList<String> nameList = new ArrayList();
//    int []b;

public void parseFile() {
    try {
        String line = "";
        FileReader file = new FileReader("D:\\Library Managment1.xml");
        BufferedReader reader = new BufferedReader(file);
        while ((line = reader.readLine()) != null) {
            if (line.contains("<Activity Idref=") || line.contains("DecisionNode Idref=") || line.contains("ActivityFinalNode Idref=") || line.contains("SendSig
                //read id
                 if (line.contains("InitialNode Idref")) {
                    NodeId.add("St");
                }
                if (line.contains("<Activity Idref=")) {
                    NodeId.add("A");
                } else if (line.contains("DecisionNode Idref=")) {
                    NodeId.add("D");
                } else if (line.contains("ActivityFinalNode Idref=")) {
                    NodeId.add("AF");
                } else if (line.contains("JoinNode Idref=")) {
                    NodeId.add("J");
                } else if (line.contains("AcceptEventAction Idref=")) {
                    NodeId.add("AC");
                } else if (line.contains("ForkNode Idref")) {
                    NodeId.add("F");
                } else if (line.contains("SendSignalAction Idref=")) {
                    NodeId.add("S");
                }
                String[] temp = line.split("\"");
                active_Ids.add(temp[1]);
                nameList.add(temp[3]);
            //read name
            }
        }
        System.out.println("IDs of the nodes " + nameList.size());
```

For Help, press F1 Ln 19 Col 1 Char 1 OVR Read CAP NUM

Figure 5.4 XML parser

5.4 Generation of Adjacency List

Adjacency list is generated while parsing the XML file. This Adjacency list is used to represent the activity in the form of graph.

```
1. Enquiry about book ---->>>Check availability of book----->x
2. Check availability of book---->>>DecisionNode2----->x
3. Validate member---->>>DecisionNode3----->x
4. Check Number of book Issued to member---->>>DecisionNode1----->x
5. Issue book---->>>join----->S1----->x
6. Add member, book and issue detail---->>>join----->x
7. Update book status---->>>join----->x
8. Book not issued---->>>STOP----->x
9. Book not available---->>>STOP----->x
10. Register member---->>>check Number of book Issued to member----->x
11. DecisionNode1---->>>Book not issued----->Fork----->x
12. DecisionNode2---->>>Book not available----->Validate member----->x
13. DecisionNode3---->>>Register member----->check Number of book Issued to
member----->x
14. Fork---->>>Update book status----->Issue book----->Add member, book and issue
detail----->x
15. join---->>>STOP----->x
16. START---->>>Enquiry about book ----->x
17. STOP---->>>x
18. A1---->>>Add member, book and issue detail----->Update book status----->x
19. S1---->>>A1----->x
```

Figure 5.5 Adjacency list of intermediate activity diagram

1. Enquiry about book ---->>>Check availability of book----->x
2. Check availability of book---->>>DecisionNode2----->x
3. Validate member---->>>DecisionNode3----->x
4. check Number of book Issued to member---->>>DecisionNode1----->x
5. Issue book---->>>join----->x
6. Add member, book and issue detail---->>>join----->x
7. Update book status---->>>join----->x
8. Book not issued---->>>STOP----->x
9. Book not available---->>>STOP----->x
10. Register member---->>>check Number of book Issued to member----->x
11. DecisionNode1---->>>Book not issued----->Fork----->x
12. DecisionNode2---->>>Book not available----->Validate member----->x
13. DecisionNode3---->>>Register member----->check Number of book Issued to member----->x
14. Fork---->>>Update book status----->Issue book----->Add member, book and issue detail----->x
15. join---->>>STOP----->x
16. START---->>>Enquiry about book ----->x
17. STOP---->>>x

Figure 5.6 Adjacency list of original activity diagram

5.5 Module for Generating Adjacency List

This module of the program generates the adjacency matrix. This is used by allpath module to generate the test scenario

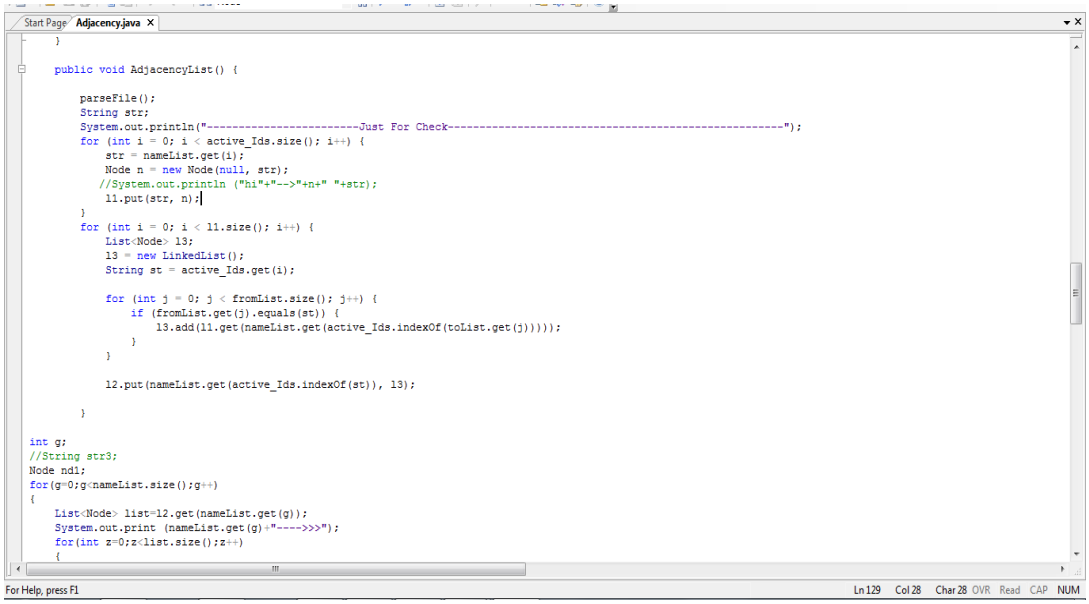


Figure 5.7 Module for generating adjacency list

5.6 Priority Table

Priority table is used for validating the generated test scenario. It is used to store priorities of activities nodes that are connected to fork activity node.

Table 5.1 Priority table

Activity Node	**Priority**
FORK	1
ISSUE DETAIL	2
ADD MEMBER, BOOK AND ISSUE DETAIL	3
UPDATE BOOK STATUS	3
JOIN	4

5.7 Module for Generating Priority Table

Module for generating priority table is made using Java7 and the snapshot for the same is shown below.

```
        return Nodename;
    }

    public HashMap<String,Integer> priority() {

        int i, j;
        String name = " ";
        String name1 = " ";
        String name2 = " ";
        Node n = null;

        for (i = 0; i < nameList.size(); i++) {
            if (nameList.get(i).equals("START")) {
                name = nameList.get(i);
                l4.put(name, greatness);
            }
        }

        for (i = 0; i < l2.size(); i++) {
            List<Node> list = l2.get(name);
            for (j = 0; j < list.size(); j++) {
                n = list.get(j);
                Set set = l1.keySet();
                Iterator iter = set.iterator();
                while (iter.hasNext()) {
                    name1 = iter.next().toString();

                    if (l1.get(name1).equals(n)) {

                        if (NodeId.get((nameList.indexOf(name1))).equals("F"))
                            {

                            l4.put(name1, ++greatness);
                            greatness++;
                            name2 = forkPriority(l4, name1);
                           }
                            else {
```

Figure 5.8 Snapshot of priority table generation module

5.8 Java Module for Generating Test Scenario

Module for generating test scenario uses DFS for applying all path coverage criteria. This module is used to generate the test scenarios. You will see the generated scenarios in the next part.

```
public void allPath(Map<String, List<Node>> l4, int count, String name, Node n, Set set, Map<String, Node> l5, Map<String, Integer> l6) {

    b[count] = n;
    String name1 = "";
    if (is_Empty(l4, name) == 1) {
        //System.out.println ("yes");
        print(b, count, l6);
        return;
    } else {

        for (int h = 0; h < nameList.size(); h++) {

            if (nameList.get(h).equals(name)) {
                if (NodeId.get(h).equals("F")) {
                    name1 = "F";
                }
            }
        }
        if (name1.equals("F")) {
            int k = count + 1;
            for (int j = 0; j < l4.get(name).size(); j++) {
                count++;
                b[count] = l4.get(name).get(j);
               // System.out.println("hiiiiiii"+l4.get(name).get(j)+" ");
                }
                //System.out.println("k"+count+" "+k);

                p.exchange(b, k, count, set, l4, l5, this, l6);
           // System.out.print("exchange return");
            } else {
                // System.out.println("else"+" "+name);
                for (int j = 0; j < l4.get(name).size(); j++) {
                    Node v = l4.get(name).get(j);
                    Set s = l1.keySet();
                    Iterator t = s.iterator();
                    while (t.hasNext()) {
```

Figure 5.9 Test scenario generating module

5.9 Test Scenarios

Test scenario as the result of the above module. The below figure shows the output test scenarios, which are basically the different paths in the activity diagram. Using these test scenario test cases can be made.

1. START---->Enquiry about book ---->Check availability of book---->DecisionNode2---->Book not available----> STOP---->X
2. START---->Enquiry about book ---->Check availability of book---->DecisionNode2---->Validate member----> DecisionNode3---->Register member---->check Number of book Issued to member---->DecisionNode1---->Book not issued---->STOP---->X
3. START---->Enquiry about book ---->Check availability of book---->DecisionNode2---->Validate member----> DecisionNode3---->Register member---->check Number of book Issued to member---->DecisionNode1----> Fork---->Issue book---->Update book status---->Add member, book and issue detail---->join---->STOP---->X
4. START---->Enquiry about book ---->Check availability of book---->DecisionNode2---->Validate member----> DecisionNode3---->Register member---->check Number of book Issued to member---->DecisionNode1----> Fork---->Issue book----> Add member, book and issue detail---->Update book status----> join---->STOP---->X
5. START---->Enquiry about book ---->Check availability of book---->DecisionNode2---->Validate member----> DecisionNode3---->check Number of book Issued to member---->DecisionNode1---->Book not issued----> STOP---->X
6. START---->Enquiry about book ---->Check availability of book---->DecisionNode2---->Validate member----> DecisionNode3---->check Number of book Issued to member---->DecisionNode1---->Fork---->Issue book----> Update book status---->Add member, book and issue detail---->join---->STOP---->X
7. START---->Enquiry about book ---->Check availability of book---->DecisionNode2---->Validate member----> DecisionNode3---->check Number of book Issued to member---->DecisionNode1---->Fork---->Issue book----> Add member, book and issue detail---->Update book status----> join---->STOP---->X

Figure 5.10 Test scenarios

5.10 Example Activity Diagram Taken from an Online Repository

To show the working of presented approaches two example activity diagram have been taken from online repository. The first one is facebook login activity diagram and second is online shopping activity diagram.

5.10.1 Facebook Login Example and Its Result

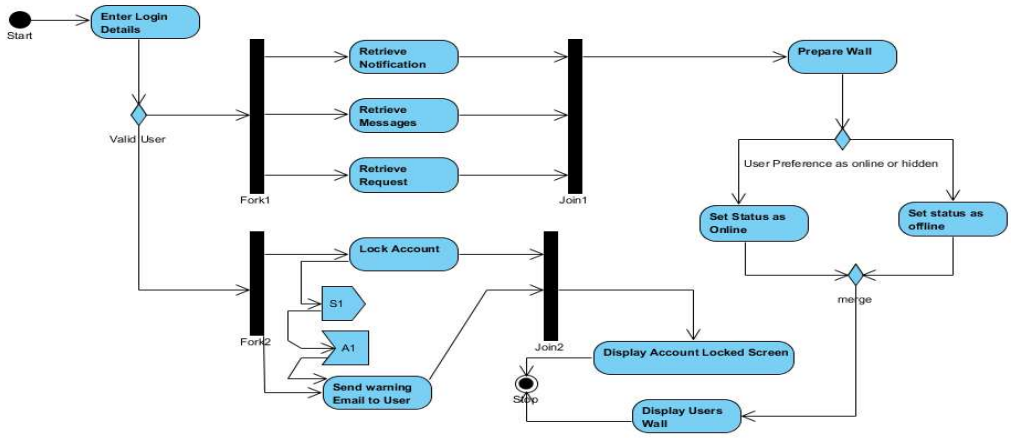


Figure 5.11 Facebook login activity diagram (original activity diagram) [47]

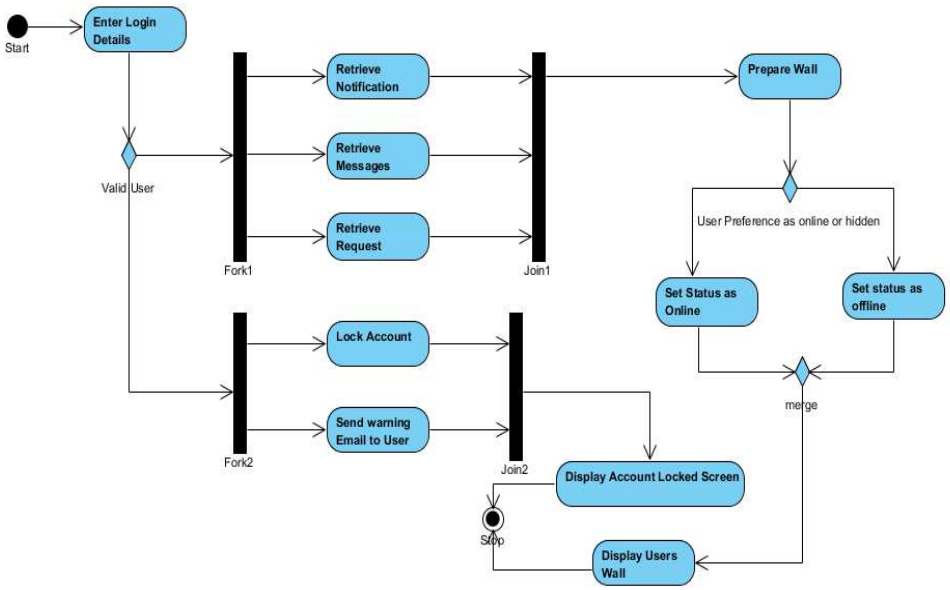


Figure 5.12 Facebook login activity diagram (intermediate activity diagram) [47]

1. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Notification---->Retrieve Messages---->Retrieve Request---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set Status as Online---->merge ---->Display Users Wall---->Stop---->X
2. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Notification---->Retrieve Messages---->Retrieve Request---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set status as offline---->merge ---->Display Users Wall---->Stop---->X
3. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Notification---->Retrieve Request---->Retrieve Messages---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set Status as Online---->merge ---->Display Users Wall---->Stop---->X
4. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Notification---->Retrieve Request---->Retrieve Messages---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set status as offline---->merge ---->Display Users Wall---->Stop---->X
5. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Messages---->Retrieve Notification---->Retrieve Request---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set Status as Online---->merge ---->Display Users Wall---->Stop---->X
6. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Messages---->Retrieve Request---->Retrieve Notification----> Join1---->Prepare Wall---->User Preference as online or hidden---->Set Status as Online---->merge ---->Display Users Wall---->Stop---->X
7. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Messages---->Retrieve Notification---->Retrieve Request---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set status as offline---->merge ---->Display Users Wall---->Stop---->X
8. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Messages---->Retrieve Request---->Retrieve Notification----> Join1---->Prepare Wall---->User

Preference as online or hidden---->Set status as offline---->merge ---->Display Users Wall---->Stop---->X

9. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Request---->Retrieve Notification---->Retrieve Messages---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set Status as Online---->merge ---->Display Users Wall---->Stop---->X
10. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Request---->Retrieve Messages---->Retrieve Notification----> Join1---->Prepare Wall---->User Preference as online or hidden---->Set Status as Online---->merge ---->Display Users Wall---->Stop---->X
11. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Request---->Retrieve Notification---->Retrieve Messages---->Join1---->Prepare Wall---->User Preference as online or hidden---->Set status as offline---->merge ---->Display Users Wall---->Stop---->X
12. Start---->Enter Login Details---->Valid User---->Fork1---->Retrieve Request---->Retrieve Messages---->Retrieve Notification----> Join1---->Prepare Wall---->User Preference as online or hidden---->Set status as offline---->merge ---->Display Users Wall---->Stop---->X
13. Start---->Enter Login Details---->Valid User---->Fork2---->Send warning Email to User---->Lock Account---->Join2---->Display Account Locked Screen---->Stop---->X

Figure 5.13 Test scenarios of facebook login activity diagram

5.10.2 Online Shopping Example and Its Result

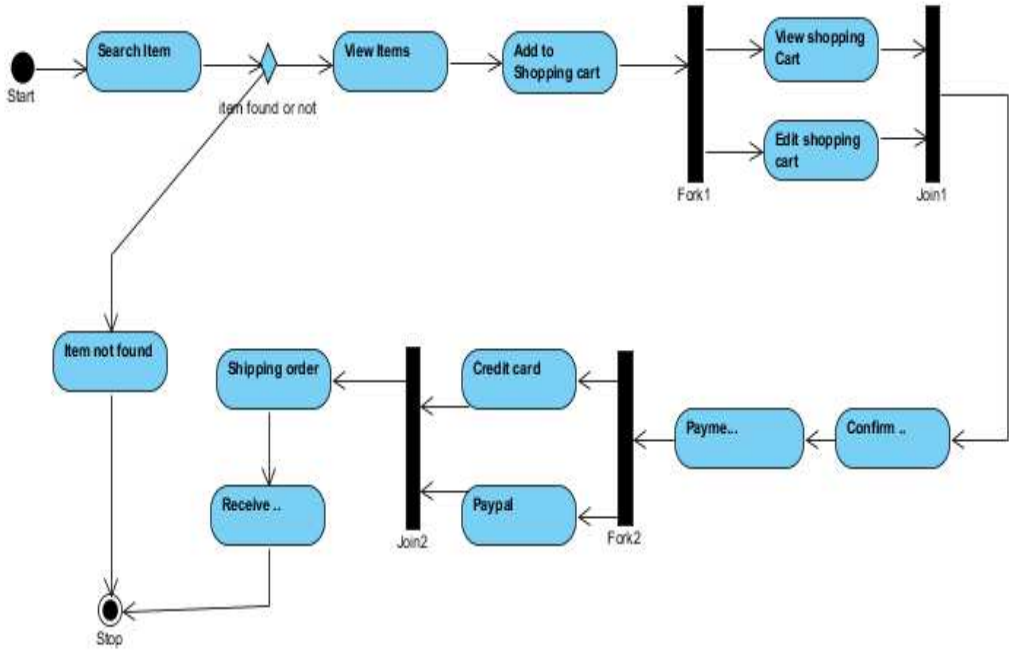


Figure 5.14 Online shopping activity diagram (original activity diagram) [47]

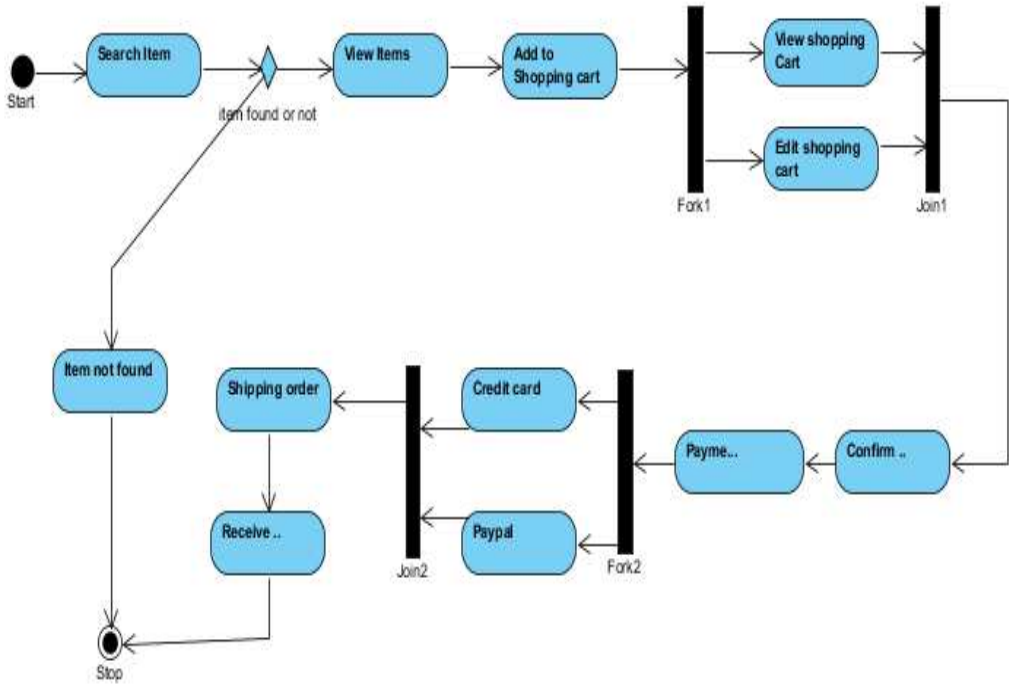


Figure 5.15 Online Shopping Activity Diagram (Intermediate Activity Diagram) [47]

1. Start---->Search item---->Item found or not---->View item---->Add to shopping cart---->Fork1---->View shopping cart---->Edit shopping card---->Join1---->Confirm order---->Payment---->Fork2 ---->Credit card---->Paypal---->Join2---->Shipping order---->Receive order--->Stop
2. Start---->Search item---->Item found or not---->View item---->Add to shopping cart---->Fork1---->View shopping cart---->Edit shopping card---->Join1---->Confirm order---->Payment---->Fork2 ----> Paypal---->Credit card----> Join2---->Shipping order---->Receive order--->Stop
3. Start---->Search item---->Item found or not---->View item---->Add to shopping cart---->Fork1----> Edit shopping card---->View shopping cart----> Join1---->Confirm order---->Payment---->Fork2 ---->Credit card---->Paypal---->Join2---->Shipping order---->Receive order--->Stop
4. Start---->Search item---->Item found or not---->View item---->Add to shopping cart---->Fork1----> Edit shopping card---->View shopping cart----> Join1---->Confirm order---->Payment---->Fork2 ----> Paypal---->Credit card----> Join2---->Shipping order---->Receive order--->Stop
5. Start---->Search item---->Item found or not---->Item not found---->Stop

Figure 5.16 Test scenarios of online shopping activity diagram

5.11 Space Complexity

For a computer program the space complexity is defined as a measure of the amount of working storage that is needed by an algorithm. Figure 5.18 represents the example activity diagram concerning PIN check for an ATM. Adjacency matrix as well as adjacency list were generated as an internal representation for storing intermediate values. The respective adjacency matrix and adjacency list obtained for Figure 5.18 are shown in Figure 5.19 and Figure 5.20 respectively. The size of integer is 4 bytes and there are 49 entries in case of adjacency matrix and there are 13 entries in case of adjacency list. So, this can be computed that the memory requirement is 196 bytes when adjacency matrix has been used and its 52 bytes, when adjacency list has been deployed.

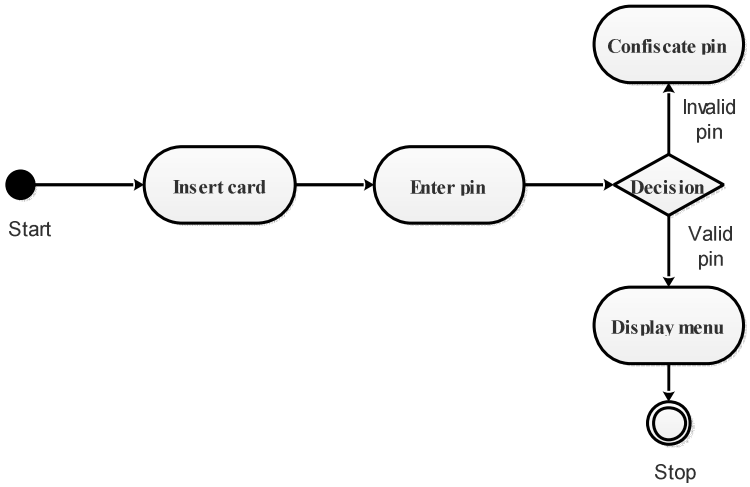


Figure 5.17 Example activity diagram for pin check [48]

Table 5.2 Activity nodes and its corresponding integer values

Activity node	Integer value
Start	1
Insert Card	2
Enter pin	3
Decision	4
Confiscate pin	5
Display menu	6
Stop	7

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0
4	0	0	0	0	1	1	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0

Figure 5.18 Adjacency matrix

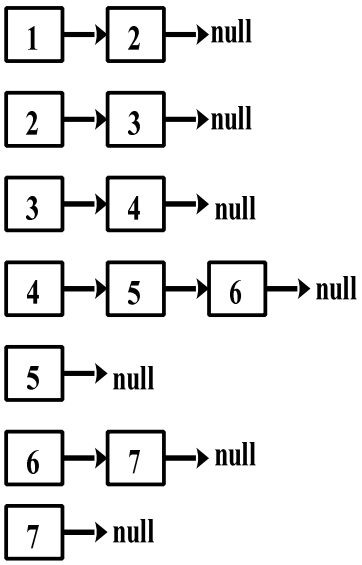


Figure 5.19 Adjacency list

Chapter 6
Conclusion and Future Work

The conclusion of proposed approach for generating the test scenario/s from UML activity diagram is presented in this chapter. Some points, that can be considered in future for extending this research work, are also mentioned in this section.

6.1 Conclusion

Experimental results have shown that the algorithm as proposed for generating the test scenario/s from UML activity diagram has reduced the space complexity. In a system when the count of the concurrent processes was n, the space consumed by the 2-D array was n!, whereas the proposed approach consume only n units of space. The proposed approach is different from the existing approach in following ways:

1. In previous approach adjacency matrix is used to represent the activity diagram as a graph, whereas in current approach adjacency list is used.

2. In this approach XML file is generated for the activity diagram to make the activity diagram parsing task easy.

3. In the previous approach to store all the combination of concurrent activity nodes 2D array is used, whereas in this approach combinations are not stored at all, combinations are generated in the path itself.

6.2 Future Scope

In future the proposed approach may be extended in conjunction with the following:

1. Test scenario/s for the concurrent module having the constructs like locking mechanism.
2. Test value generation using the concept of def-use chaining with the test path/test scenario generation.

References

[1] "Testing Introduction" Internet: http://agile.csc.ncsu.edu/SEMaterials/IntrotoTesting.pdf [As accessed on Apr. 2015].

[2] I. Javanović, "Software Testing Methods and Techniques." The IPSI BgD Transactions on Internet Research, Vol. 5(1), pp.31-41, Internet Journal, 2009.

[3] P.Kaur, P.Bansal, and R.Sibal, "Prioritization of test scenarios derived from UML activity diagram using path complexity". In Proceedings of the CUBE International Information Technology Conference, pp. 355-359, ACM, 2012.

[4] "Types of software testing" Internet:http://www.softwaretestinghelp.com/types-of-software-testing [As accessed on Mar. 2015].

[5] "IEEE standard glossary of software engineering terminology" IEEE Std 610.12-1990, pp.1-84, Dec. 31 1990.

[6] "Activity diagram for user login" Internet: http://www.google.com [As accessed on Apr. 2015].

[7] G. Booch, J. Rumbaugh and I. Jacobson, Unified Modeling Language User Guide, 1st ed. Addison Wesley, 1998.

[8] "Sequence diagram" Internet: http://en.wikipedia.org/wiki/Sequence_diagram [As accessed on Apr. 2015].

[9] "Interaction diagram" Internet: http://www.tutorialspoint.com/uml/uml_interaction_diagram.htm [As accessed on Mar. 2015].

[10] "Statechart diagram" Internet:http://www.tutorialspoint.com/uml/uml_statechart_diagram.htm [As accessed on Mar. 2015].

[11] C. Sun, B. Zhang, J. Li2,"TSGen: A UML Activity Diagram-based Test Scenario Generation Tool," *In International Conference on Computational Science and Engineering*, Vancouver, Canada, Vol. 2, pp. 853-858, Aug. 2009.

[12] "XML." Internet: https://en.wikipedia.org/wiki/XML [As accessed on Apr. 2015].

[13] R.K. Swain, V. Panthi and P.K. Behera, “Generation of test cases using UML activity diagram,” *In International journal of computer science and informatics*, PP. 2231-5292, 2013.

[14] X. Chen, N. Ye, P. Jiang, L. Bu and X. Li," Feedback-directed test case generation based on UML activity diagrams," *In 5th International Conference on Secure Software Integration and Reliability Improvement Companion SSIRI-C’11*, pp. 9-10, June 2011.

[15] C.A. Sun, "A Transformation-Based Approach to Generating Scenario-Oriented Test Cases from UML Activity Diagrams for Concurrent Applications," *In Computer Software and Applications COMPSAC'08*, Turku, Finland, pp.160-167, Aug. 2008.

[16] W. Linzhang, Y. Jiesong, Y. Xiaofeng , H. Jun, L. Xuandong, and Z. Guoliang, “ Generating test cases from UML activity diagram based on gray-box method, ” *In 11th Asia-Pacific Software Engineering Conference, APSEC’04*, Busan, Korea, pp. 284-291, Nov. 2004.

[17] P.N. Boghdady, N.L. Badr, M. Hashem and M.F. Tolba,” An enhanced test case generation technique based on activity diagrams,” *In International Conference on Computer Engineering and Systems ICCES’11*, pp. 289-294.

[18] P.N. Boghdady, N.L. Badr, M. Hashem and M.F. Tolba,” A proposed test case generation technique based on activity diagrams, “*In International Journal of Engineering and Technology*, June. 2011.

[19] H. Kim, S. Kang, J. Baik, I. Ko, “Test Cases Generation from UML Activity Diagrams.”, *In 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Qingdao, China, pp. 556-561, Jul. 2007.

[20] S. Kansomkeat, J. Offutt, “ Generating Test Cases From UML Activity Diagrams Using Condition Classification Tree Method,” *In 2nd International Conference on Software Technology and Engineering*, Puerto Rico, United States, pp. V1-66, Oct. 2010.

[21] X. Fan, J. Shu, L. Liu and Q.J. Liang,” Test case generation from UML subactivity and activity diagram,” *In Second International Symposium on Electronic Commerce and Security ISECS'09*, Vol. 2, pp. 244-248, May. 2009.

[22] C. Mingsong, Q. Xiaokan, and L. Xuandong, "Automatic test case generation for UML activity diagrams," *In Proceedings of the 2006 International*

Workshop on Automation of Software Test, Shanghai, China, pp. 2-8, May 2006

[23] R. Chandler, C.P. Lam, and H. Li. "AD2US: An automated approach to generating usage scenarios from UML activity diagrams," *In 12th Asia- Pacific Software Engineering Conference, APSEC'05*, Taipei, Taiwan, Dec. 2005.

[24] B.N. Biswal, P. Nanda, and D.P. Mohapatra, "A novel approach for scenario-based test case generation," *In International Conference on Information Technology ICIT'08*, Bhubaneshwar, India, pp. 244-247, Dec. 2008.

[25] K.H. Chang, W.H. Carlisle, J.H. Cross II & D.B. Brown, "A heuristic approach for test case generation," *In Proceedings of the 19th annual conference on Computer Science*, New York, USA, pp. 174-180, Apr. 1991.

[26] A.K. Jena, S.K. Swain and D.P. Mohapatra, "A Novel Approach for Test Case Generation from UML Activity Diagram," *In International Conference on Issues and Challenges in Intelligent Computing Techniques*, Delhi, India, pp. 621-629, Feb 2014.

[27] P.E. Patel and N.N. Patil," Testcases Formation Using UML Activity Diagram, "*In International Conference on Communication Systems and Network Technologies CSNT'13*, Gwalior, India, pp. 884-889, Apr. 2013.

[28] L. Li, X. Li, T. He and Xiong, "Extenics-based Test Case Generation for UML Activity Diagram," *In Procedia Computer Science*, pp. 1186-1193, June 2013.

[29] S. Tiwari, and A. Gupta, "An Approach to Generate Safety Validation Test Cases from UML Activity Diagram," *In Software Engineering Conference APSEC'13*, Bangkok, Thailand, Vol. 1, pp. 189-198, Dec 2013.

[30] P. G. Sapna and H. Mohanty, "Automated Scenario Generation Based on UML Activity Diagrams," *In International Conference Information Technology ICIT'08*, Bhubaneswar, India, pp. 209-214, Dec 2008.

[31] P. Nanda, D.P. Mohapatra and S.K. Swain, "Generation of Test Scenarios Using Activity Diagram," *In Proceedings of SPIT-IEEE Colloquium and International Conference*, Mumbai, India, Vol. 4, pp. 69-73, Feb 2008.

[32] P. Kaur, P. Bansal and R. Sibal, "Prioritization of test scenarios derived from UML activity diagram using path complexity," *In Proceedings of the CUBE International Information Technology Conference ACM*, Pune, Maharashtra, India, pp. 355-359, Sept 2012.

[33] D. Xu, L. Huaizhong, and C.P. Lam," Using adaptive agents to automatically generate test scenarios from the UML activity diagrams, " *In 12th Asia-Pacific Software Engineering Conference APSEC'05*, Taipei, Taiwan, pp. 8-pp, Dec. 2005.

[34] "B4S." Internet: http://www.adi.com/products/b4s/ [As accessed on Mar. 2015].

[35] "Safety Test Builder." Internet: http://www.chiastek.com/products/stb.html [As accessed on Apr. 2015].

[36] "Simulink Design Verifier." Internet: http://www.mathwork.in/products/slidesignverifier/ [As accessed on Apr. 2015].

[37] "T-VEC Tester for Builder." Internet: http://t-vec.com/solutions/simulink.php [As accessed on Apr. 2015].

[38] P. Louridas,"JUnit: unit testing and coiling in tandem," Software, vol. 22,no. 4, pp.12-15, 2005.

[39] "AgitatorOne." Internet: http://www.agitar.com/solutions/products/agitorone.html [As accessed on Mar. 2015].

[40] K. Havelund and T. Pressburger," Model Checking Java programs using java path finder," International Journal on Software Tools for technology transfer, Vol. 2, no. 4,pp. 366-381,2000.

[41] "Tefkat" Internet: http://tefkat.sourceforge.net/ [As accessed on Apr. 2015].

[42] "MOFScript Home Page." Internet: http://www.eclipse.org/mt/mofscript/ [As accessed on Apr. 2015].

[43] "ConTest." Internet: http://www.research.ibm.com/haifa/project/verification/contest/ [As accessed on Mar. 2015].

[44] "Quick Test Professional." Internet: http://qtpfaq.wordpress.com/2007/08/03what-is-quick-test-professional-tool/ [As accessed on Apr. 2015].

[45] "CHESS." Internet: http://research.microsoft.com/en-us/projects/chess/ [As accessed on Apr. 2015].

[46] "NuSMV." Internet: http://nusmv.fbk.eu/ [As accessed on Apr. 2015].

[47] “Online Repository.” Internet: http://creately.com/diagram/example/i34didjl1/onlne+repository [As accessed on Mar. 2015].

[48] Russ Miles, Kim Hamiltom, *A Pragmatic Introduction to UML*, 1st ed. O’Reilly Media, 2006.

[49] Narsimha Karumanchi, *Data Structures and Algorithms Made Easy*, 2nd ed. CareerMonk Publication, 2011.

List of Publications

Communicated:

[1] Md. Aamir and Vinay Arora, "Priority Based Test Scenario Generation from UML Activity Diagram in Software Engineering," 8th International Conference on Contemporary Computing (IC3), Noida, IEEE, 2015.

Youtube Link: http://youtu.be/VABpDx9q6Tw