Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived fromUML State chart Diagrams

A.Jayakiruba
Associate Professor, Department of computer Science, Bishop Cotton Christian College, Bangalore

Abstract

Software testing is an important part of the software development process. Testcase generation is the center of testing process. To decrease the cost of software testing and to increase the reliability of the testing processes, a new method has been created to automate the testing process. Different UML diagrams include various important pieces of information that can be successfully used in a testing procedure.This paper proposes a new approach to prioritize test case scenarios by identifying the critical path clusters using genetic algorithm. The test case scenarios are derived from the UML state chart diagrams. The testing efficiency is optimized by applying the genetic algorithm on the test data. The information flow metric adopted in this work for calculating the information flow complexity associated with each node of state chart diagram.The state chart diagram is converted into an intermediate graph called as State Dependency Graph (SDG) for test case generation.If the software requirements change, the software needs to be modified. Hence, to take care of requirements change, a stack based approach for assigning weights to the nodes of state chart diagram has also been proposed.

Keywords— **software testing, genetic algorithm, state chart diagram, SDG, test case.**

I. INTRODUCTION

Testing activities consist of designing test cases.However, software testing is a time consuming and expensive task [4]. Testing can be carried out earlier in the development process so that the developer will be able to find the inconsistencies and ambiguities in the specification and hence will be able to improve the specification before the program is written [8]. It is still a major problem to meet the requirement specification for the systematic production of highquality software. Many researchers are doing research on to find effective test cases to minimize time and cost. Hence, it is important to generate test cases based on design specifications [9].Evolutionary testing is an emerging methodology for automatically producing high quality test data [1]. Genetic algorithms (GA) are well known form of the evolutionary algorithms conceived by John Holland in United States during late sixties [1], [6]. Unified Modeling Language has become the de facto standard for object-oriented modeling and design. It is a major problem to meet the requirement specification for the Systematic production of high-quality software. However, it is a non-trivial task to manually construct the state model of a system. Therefore, with continually increasing system sizes, the issue of automatic design of system test cases is assuming prime importance [10]. The UML state model of an actual system is usually extremely complex and comprises of a large number of states and transitions. Possibly for this reason, state models of complete systems are rarely constructed by system developers [10]. In case of component-based software development, test case generation based on program source code proves to be inadequate, where even the source code may not be available to the developers. Hence, it is important to generate test cases based on design specifications [9].With this motivation, I fix my objective on test case generation, automatically, using UML state chart

diagram. In this paper, I have proposed a technique for prioritization of test case scenarios derived from state chart diagram of UML using the concept of basic information flow (IF) metric, stack and GA. The paper is divided into 5 sections. Section 2 describes the basic structure of GA. In section 3, the proposed approach is discussed while section 4 describes the test case scenarios derived from state chart diagram. Section 5 concludes the paper andgives an overview of the future work.

II. GENETIC ALGORITHM

GA is an evolutionary algorithm. It has emerged as a practical, robust optimization technique and search method. A GA is a search algorithm that is inspired by the way nature evolves species using natural selection of the fittest individuals. The possible solutions to the problem being solved are represented by a population of chromosomes. A chromosome is a string of binary digits and each digit that makes up a chromosome is called a gene. This initial population can be totally random or can be created manually using processes such as greedy algorithm. The pseudo code of a basic algorithm for GA is as follows [5]:

Initialize (population)
Evaluate (population)
While
(Stopping condition not satisfied)
{
Selection (population)
Crossover (population)
Mutate (population)
Evaluate (population)
}

A GA uses three operators on its population which are described below:

A. Selection:

A selection scheme is applied to determine how individuals are chosen for mating based on their fitness. Selection generates the new population from the old one, thus starting a new generation. Each chromosome is evaluated in present generation to determine its fitness value. This fitness value is used to select the better chromosomes from the population for the next generation.

B. Crossover or Recombination:

After selection, the crossover operation is applied to the selected Chromosomes. It involves swapping of genes or sequence of bits in the string between two individuals. This process is repeated with different parent individuals until the next generation has enough individuals. After crossover, the mutation operator is applied to a randomly selected subset of the population.

C. Mutation:

Mutation alters chromosomes in small ways to introduce new good traits. It is applied to bring diversity in the population.

III. PROPOSED APPROACH

This section illustrates the details of my proposed approach for test case prioritization using GA. Theapproach uses state chart diagram for deriving the test case scenarios. In this work, I have derived the test cases form the state chart diagram.

The main constructs used in the state chart diagram are shown in Fig.1. The event with guardcondition and action generates a new state. The state chartdiagram is converted into an intermediate graph called asState Dependency Graph (SDG) for test case generation.

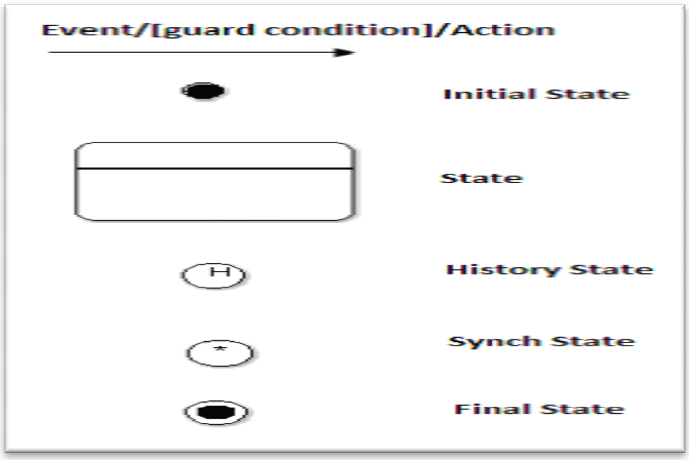

Fig.1 Main Constructs used in State Chart Diagram

A. *Procedure*

The issue of requirements change are taken care byprioritizing the nodes of SDG using the stackbased memory allocation approach and IF metrics. In this approach, the data is pushed or popped only at one end called top of stack.The stack uses last in first out (LIFO) approach. Nodeputed first is removed last from the stack. The top isincremented when node is inserted and decremented whennode is deleted.

The steps involved in identifying the critical pathclusters for astate chart diagram are asfollows:

a) Convert the state chart diagram into SDG.

b) Assign the weights to the nodes of SDG by using stack based weightassignment approach and the Basic IF Model.

In the proposed stack based memory allocationtechnique, for the nodes of the SDG eachnode is assigned a weight, w based on number ofoperations to access element in the stack. But toaccess the node, we have to pop all the data above it. Higher the number of operations required to access the node, higher is the weight and hence the complexity of the node.

In the Basic IF model [6], information flow metrics are applied to the components of system design. In my work, the component is taken as a node in theSDG. The IF is calculated for each node of aSDG. For example, the IF of node A i.e. IF(A) is calculated using equation given below:

$$IF\ (A)=[FANIN\ (A)\ x FANOUT\ (A)]\ (1)$$

Where FANIN (A) is the count of the number of othernodes that can call, or pass control to node A andFANOUT (A) is the number of nodes that are calledby node A. The IF is calculated for each node of aSDG. The weighted nodes in the path are Summed together and the complexity of each path iscalculated.Therefore, the sum of the weight of a node by stackbased weight assignment approach and IF complexitycontributes to the total weight of a node ofSDG.

c) Selection:

The decision nodes of theSDGform the chromosome. A chromosome or test datawill therefore be a binary string where a single bit omultiple bits in a string will correspond to a decisionnode of the graph. Number of bits in the chromosomewill depend upon the number of decision nodes andthe type of graph being used. For example, in SDG ifthere are four decision nodes, a four bit binary stringwill form a chromosome or an individual in thepopulation. The fitness value of each chromosome iscalculated by applying the stack based weightassignment approach and Basic IF model. Thechromosomes with high fitness value are selected asthe parents for the reproduction. The fitness value ofeach chromosome is calculated by using the formulagiven below:

$$F=\sum_{i-1}^{n} w_i \quad (2)$$

Where, W_i is weight of i^{th} node in a path under consideration and n is number of nodes in a current path.Weight of i^{th} node is the sum of IF complexityand stack based complexity given by equation givenbelow:

W_i=IF(i) STACKBASEDWEIGHT(i) (3)

d) Crossover:

There are number of techniques ofcrossover, but all require swapping of genes orsequence of bits in the chromosome. It involvesswapping between two individuals or test data in ourcase. In this work, I have assumed the probability ofcrossover is taken as 80% [5]. The random number r is generated from 0 to 1. Crossover is done if r < 0.8condition is satisfied.

e) Mutation:

Mutation is done to introduce new traits in the population to avoid the localoptima. In mutation the bits are flipped from 0 to 1and vice versa. Here, I have assumed theprobability of mutation as 20%. The random numbers generated from 0 to 1. If r < 0.2 condition issatisfied, then the bits of the test data are mutatedrandomly.

The algorithm for the proposed approach to generate testscenarios for state chart diagram isshown in Fig.3 while the weightassignment algorithm is shown in Fig.2.

ALGORITHM 1

1. for each activity node a_i, i = 1...n ,

a) Push nodes of CFG on the stack using DFS and BFS approach.

b)Determine the maximum size, $s_{max}$of thestack.

c)fori= 1 to s_{max}, assign w= s_{max}– k to each node of the CFG, where $s_{max}$is maximum size of stack and k is number of nodes above current node.

d)for each decision node, d_i

(i) Assign the same weight, W to branching nodes.

(ii) Insert the next neighbor nodes of branching nodes and update top.

(iii) Neglect the branching nodes anddecision nodes that have been inserted previously.
end

e)Assign same weight, two concurrent nodes.

end

Fig. 2 Algorithm for the weight assignment

ALGORITHM 2

1. Convert the state chart diagram into SDG.

2. Use the fork nodes in SDG to generate the test data or chromosome population randomly.

3. For each test data i = 1.... n,

a) Traverse the SDG by applying Depth first search (DFS).

(i) Find the neighbor node for the current node having next higher depth dby applying DFS.

(ii)Update the top pointer,size sand kof the stack, where kis number of nodes above the current node.

(iii) Assign weight, two the nodeby applying the weight assignment algorithm.

b)Calculate the fitness value of each test data by using equation (2) and stack based weight assignment approach.

c)Select initial test data by ranking the fitness of the chromosomes.

d)If initial population is not enough randomly generate them.

If r< 0.8, perform crossover
Else if r< 0.2, perform mutation
end if
end if
end

4. If test data for all the paths have not been covered, then repeat the GA process.
5. Else end system.

Fig. 3 Proposed Algorithm for generating test scenarios from state chart diagram

IV. TEST CASE SCENARIOS DERIVED FROM STATE CHART DIAGRAM

I have applied this approach on the state chart diagram of the student enrolment system as shown in Fig4.

In student enrolment system, student enrolled is an event, seat available is a guard condition and add student is an action.

If the guard condition becomes true then action is performed and open for enrolment state is generated. The intermediate graph is called as SDG (Fig.5).

In Fig.5, INIT represents start process, PR represents proposed state, SC represents scheduled state, OE represents open for enrolment state, FU represents Full state and CE represents closed to enrolment state respectively.

The events are shown as e1, e2, e3.....e12 where e1 is scheduled, e2 is registration open,e3 is student apply for enrolment, e4 is enrolment closed, e5 is student apply for waiting list, e6 is close enrolment, e7 is student dropped, e10 is seat allotment and e8, e9, e11, e12 are cancelled events respectively.

As shown in Fig., there are four decision nodes namely 2, 3, 4, and 6 respectively which will form the chromosomes in this case study.

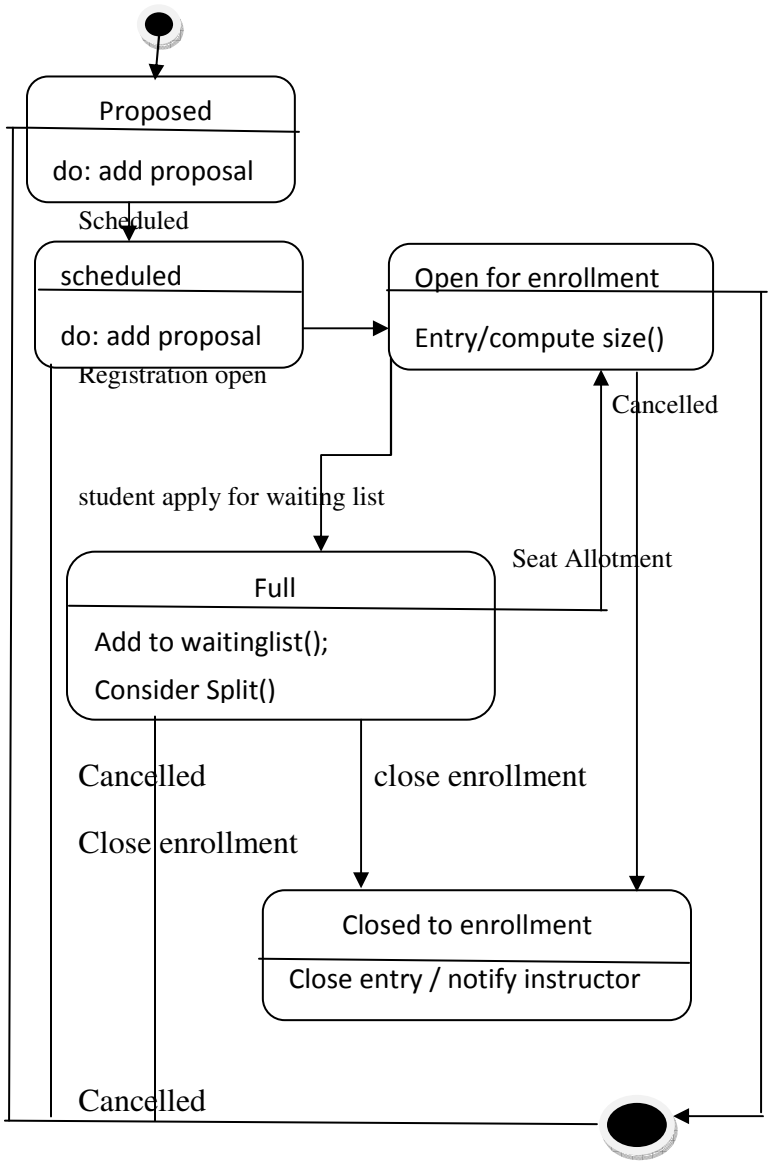

Fig 4. State chart diagram of student enrollment system

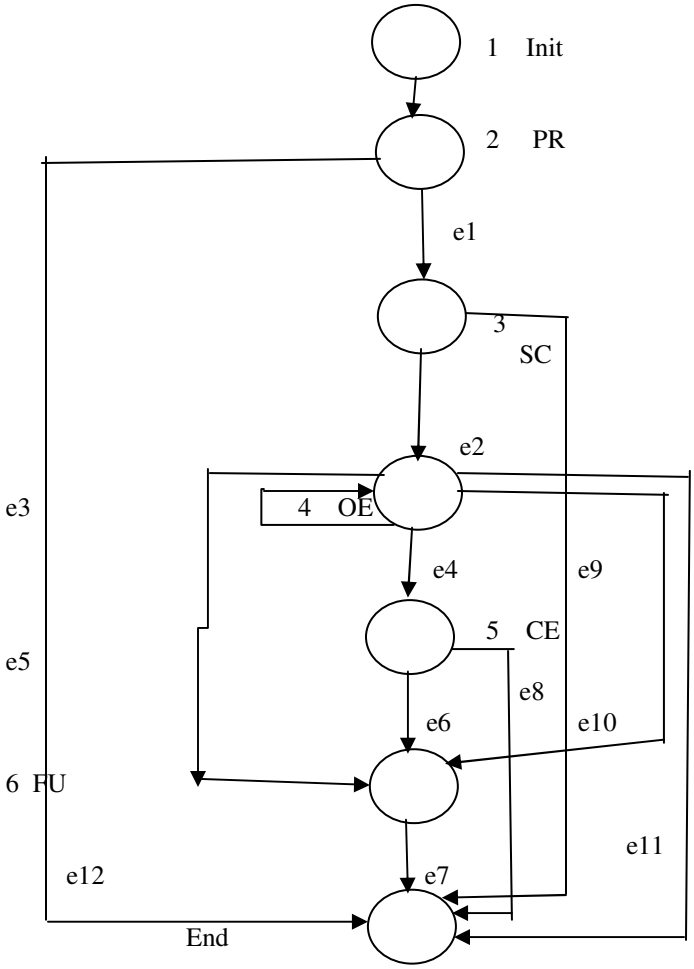

Fig. 5 SDG of student enrolment system

Nodes	K	Size, s	Weight=Smax-k
7	5	6	6-5=1
5 or 6 or 7	4	5	6-4=2
4	3	4	6-3=3
3 or 7	2	3	6-2=4
2	1	2	6-1=5
1	0	1	6-0=6

Fig. 6 Stack based weight assignment to nodes of SDG

The events e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11 and e12 represent the edges and the states represent the corresponding nodes in the SDG. The

test data corresponding to decision nodes 2, 3, 4 and 6 are shown in Table 8. These test data show the events representing edges in the SDG and are part of test case scenarios. For e.g. data involving events e1, e2, e5 and e7 willlead to path consisting of nodes 1, 2, 3, 4, 6 and 7. In Table 8 we have considered all the test data leading to valid as well as invalid paths.

TABLE1: TEST CASES FOR SDG

S.No		Decision Nodes			
		2	3	4	6
1.		e1	e2	e4	e7
2.		e1	e2	e4	e10
3.		e1	e2	e5	e7
4.		e1	e2	e5	e10
5.		e1	e2	e11	e7
6.		e1	e2	e11	e10
7.		e1	e9	e4	e7
8.		e1	e9	e4	e10
9.		e1	e9	e5	e7
10.		e1	e9	e5	e10
11.		e1	e9	e11	e7
12.		e1	e9	e11	e10
13.		e12	e2	e4	e7
14.		e12	e2	e4	e10
15.		e12	e2	E5	e7
16.		e12	e2	E5	e10
17.		e12	e2	11	e7
18.		e12	e2	e11	e10
19.		e12	e9	e4	e7
20.		e12	e9	e4	e10
21.		e12	e9	e5	e7
22.		e12	e9	e5	e10
23.		e12	e9	e11	e7
24.		e12	e9	e11	e10
25.		e1	e2	e3	e7
26.		e1	e2	e3	e10
27.		e1	e9	e3	e7
28.		e1	e9	e3	e10
29.		e12	e2	e3	e7
30.		e12	e2	e3	e10
31.		e12	e9	e3	e7
.32.		e12	e9	e3	e10

In this e.g.as clear from Fig.5 in a state, maximum four events can take place. Therefore each event in SDG is represented in two bit format. In our case, the events e1 and e12 at node 2 are represented by 00 and 01, events e2 and e9 at node 3 are represented by chromosome 00 and 01, events e11, e5, e4 and e3 at node 4 are represented by 00, 01, 10 and 11 respectively. Events e7 and e10 at node 6 are represented by 00 and 01 respectively. Therefore, each event in SDG is represented in two bit format. As there are four decision nodes in the SDG, the chromosome here will consist of a binary string of 8 bits. Here, the loops are traversed at most once. Therefore, node having self-loop is considered only once in a path. For example, the test case scenario involving events e1, e2, e3, e7 will follow the path consisting of nodes 1, 2, 3, 4, 5 and 7 .The node 4 containing self-loop is traversed only once in the path and the next node to be followed after loop is the nearest neighbor having shortest distance i.e. next node to be followed after node 4 is 5 as the distance between node 4 and node 5 is small as compared to node 6 and 7.

The sum of stack based priority number (A) and the IF complexity of each node (B) is equal to the total complexity of each node. The IF complexity of each node of SDG is shown in Table 9 where, FI is Fan In and FO is Fan Out for the corresponding node. The test data representation involving decision nodes is shown in Fig.6using two bit format for each decision node.

e1 e2 e4 e7

0 0	0 0	1 0	0 0

Test case 1

e1 e2 e4 e7

e1 e2 e4 e10

0 0	0 0	1 0	0 0

Test case 2

.

.

.

e12e9 e3 e7

0 1	0 1	1 1	0 0

Test case 31

e12e9 e3e10

0 1	0 1	1 1	0 1

Test case 32

Fig.7 Test case representation using two bit format

TABLE2: COMPLEXITY OF NODES OF SDG

Node	Complexity based on pop operation, (A)	Fan In(a) * Fan Out(a),(B)	Total Complexity=(A+B)
1	6	0	6
2	5	2	7
3	4	2	6
4	3	6	9
5	2	2	4
6	2	3	5
7	4+2+1=7	0	7

We start the process by randomly generating the initial population as shown in Table 10.

Initial Population: - 00011000, 01001000, 01000101, 00000100 where X is the test data, F(X) is the fitness value, r is random number generated from 0 to 1. F'(X) is new computed fitness value after crossover and mutation operation. C is crossover and M is mutation operation. The test data 00011000 will traverse the edges e1, e9, e4, e7 and hence will follow the path 1, 2, 3 and 7 and the corresponding fitness value of test data is 6 + 7 + 6 + 7 = 26. The test data 01001000 will traverse the edges e12, e2, e4 and e7 and hence will follow the path 1, 2 and 7 and the corresponding fitness value is 6 + 7 + 7 = 20.

The test data 01000101 will follow the edges e12, e2, e5, e10 and will follow the path 1, 2 and 7 and the corresponding fitness value is 6 + 7 + 7 = 20. Similarly the test data 00000100 will follow the edges e1, e2, e5, e7 and hence will follow the path 1, 2, 3, 4, 6 and 7 and the fitness value computed is 6 + 7 + 6 + 9 + 5 + 7 = 40.

TABLE3: ITERATION 1

S NO	X	F(X)	R	C	M	F'(X)
1	00011000	26	.829	00011000	00011000	26
2	01001000	20	.876	01001000	01001000	20
3	01000101	20	.712	01000100	01000100	20
4	00000100	40	.686	00000101	00000101	54

TABLE 4: ITERATION 2

S. NO	X	F(X)	R	C	M	F'(X)
1	00011 0000	26	.917	0011 0000	00110 000	26
2	01001 000	20	.126	0100 1000	01000 100	20
3	01000 100	20	.814	0100 0100	01000 100	20
4	00000 100	54	.562	0000 0100	00000 100	54

TABLE 5: ITERATION 3

S. NO	X	F(X)	R	C	M	F'(X)
1	00110000	26	.833	00110000	00110000	26
2	01000100	20	.415	01000100	01000100	20
3	01000100	20	.124	01000100	01000100	20
4	00000100	54	.612	00000100	00000100	54

TABLE6: ITERATION 8

S. NO	X	F(X)	R	C	M	F'(X)
1	011000100	20	.841	01000100	00100000	20
2	00000101	54	.912	00000101	00000000	54
3	00001001	39	.104	00001001	00000101	54
4	00000101	54	.809	00000101	00010101	54

TABLE7:ITERATION 9

S. NO	X	F(X)	R	C	M	F'(X)
1	01000100	20	.971	01000100	01000100	20
2	00000101	54	.501	00000101	00000101	54
3	00001001	54	.412	00000101	00000101	54
4	00000101	54	.972	00000101	00000101	54

In this example, after the 9th iteration as shown in Table 7, the test data e1, e2, e5, e10 have the highest fitness value i.e. 54. So, the path corresponding to the chromosome 00000101 should be the one which must be tested first. Therefore the path consisting of nodes 1, 2, 3, 4, 6, 4, 5 and 7 consisting of events e1, e2, e5 and e10 should be the one that will be tested first.

V. CONCLUSION AND FUTURE WORK

In this paper a GA based approach is proposed for identifying the test path that must be tested first. Test paths or scenarios are derived from State chart diagram. The proposed approach makes use of IF model and GA to find the path to be tested first.

My future work involves applying the proposed approach on other UML diagrams like activity diagram, sequence diagram and using this technique for white box testing and object oriented testing.

REFERENCES

[1] "Towards automated support for deriving test data from UML state charts"L Briand, J Cui, Y Labiche - UML 2003-The Unified Modeling Language …, 2003

[2] UML behavioral model based test case generation: a surveyMShirole, R Kumar - ACM SIGSOFT Software Engineering Notes, 2013

[3]"Automated-generating test case using umlstate chart diagrams" SupapornKansomkeat, WanchaiRivepiboon*Proceedings of SAICSIT, pp.296-300, 2003.*

[4]Somerville, I. "Software engineering" 7th Ed. Addison Wesley.

[5]Goldberg, D.E "Genetic Algorithms: in search, optimization and machine learning", Addison Wesley, M.A (1989).

[6]MahaAlzabidi, Ajay Kumar and A. D. Shaligram "Automatic Software structural testing by using Evolutionary Algorithms for testdata generations", *IJCSNS International Journal of Computer science and Network security, Vol.9, No. 4 (2009).*

[7]Sapna P.G. and Hrushikesha "Automated Scenario Generation based on UML Activity diagrams", *International conference on information technology, pp.209-214, IEEE (2008).*

[8] KansomkeatS. andRivepiboon, W 2003. Automatedgenerating test case using UML statechart diagrams. *In Proc. SAICSIT 2003, ACM,2003, PP.296-300.*

[9] Samuel,P., Mall,R., and Bothra,A. K.2008. Automatic test case generation using Unified Modeling Language(UML) state diagrams. *IET Software, 2(2), 2008, pp.79 –93.*

[10] Sharma,M. and Mall,R.2009. Automatic generation of test specifications for coverage of system state transitions. *Information and SoftwareTechnology, (51), 2009, pp.418 –432*