A Semantic Approach for the Generation of Test Cases from Activity Diagram

Amandeep
Department of Computer Science and Engineering
Chandigarh University, Mohali, Punjab, India

Pratibha
Department of Computer Science and Engineering
Chandigarh University, Mohali, Punjab, India

Ishdeep Singla
Department of Computer Science and Engineering
Chandigarh University, Mohali, Punjab, India

ABSTRACT
Software testing is the process of evaluating a system or its modules in the intent to find that the software is acquiring the efficient requirements or not. In simple words testing is the execution of the system in order to find their gaps, ambiguity and inconsistency. Software testing comprises into three factors: test case generation, test cases execution and test cases evaluation. This paper implemented a semantic approach for the generation of test cases on UML model i.e., Activity Diagram. In this approach an Activity diagram is created then it automatically generated a Activity Dependency table (ADT) from Activity diagram. From the ADT an Activity Dependency Graph (ADG) is introduced. Finally After the automatic generation of ADG a consistent test case are generated. This approach includes the validation of the test cases by their consistency and efficiency. This approach saves the cost, time, efforts and increases the quality of generated test cases.

Keywords
Unified Modeling Language Model (UML), automatic generated test cases, Model Based testing (MBT), Genetic Algorithm, branch coverage criterion

1. INTRODUCTION
Software engineering is an approach to the development, design operation and maintenance of software. Consequently in software engineering main focus is to assure the quality in the product, detect the bugs and prevent system from bugs by testing or analysis. Software development life cycle has its own phases such as designing, coding, testing and maintenance phase. But to test the software and find its hidden bugs/errors/defect has a biggest task in software industry.

Software testing has a vital role in the software development life cycle. It involves the identification of the error/bug/defect without correcting it. In the concept of automation testing where tester writes the script and uses the software to test the software. Model Based Software Testing is a testing strategy that depends upon the extraction of test cases from the models [3, 4, 5, 7]. MBT can detect the faults of different categories in models which cannot be detected in code based testing. Detection of faults from the Model can increases the quality, efficiency and consistency.

Unified Modeling Language (UML) is most dominant modeling language. It easily analyzes the problem domains by capturing the requirements, simply reality; specify the structure and behavior of the system. It also designs the solution of the problem by structure, documentation. Basically, in UML model the artifacts are visualize, specify, construct and document. Activity diagram is one of the most important diagrams of the UML. It represents the dynamic behavior of the system [9]. Moreover, Activity diagrams can be utilized to describe the business and operational step-by-step workflows of components in a system [1, 2]. Activity diagram helps to describe the flow of control of the target system, such as exploring complex business rules and operations, describing in the business process

As per the Generation of test cases on activity diagram depicts the automatic generation of the test cases this covers all the functionalities as well all the scenarios of the software [6, 10]. Test cases for the activity diagram will specify their functionality, activity, behaviors and interaction of the modules. Test cases should be consistent and specified.

Organization: In this paper Section II describes the related work Section III describes the performance evaluation Section IV describes the conclusion and future scope.

2. RELATED WORK
The building block of this approach is depicted below. This approach constructs the Activity diagram on modeling tool like RSA, Magic Draw. To find the dependency between the activities is automatically generate the table that is Activity Dependency Table. From the dependency table a dependency graph is created which is called Activity Dependency Graph. It is concerned with the activity node and the flow of control of the activities in the Activity diagram. To examining the dependency graph Genetic Algorithm is used. ADG covers all the functionalities and scenarios of the Activity diagram. Thus, it applies on the branch criterion for the generation of test cases. This process for the generation of test cases is as follows:

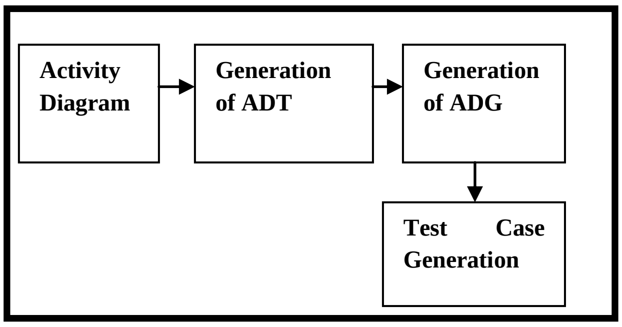

Fig 1: Building block of the Approach

As the figure, module1 defines the modeling of the activity diagram when Hotel Management System is considered module2 depicts the generation of ADT. Whereas, generation of ADG and final outcomes in the form of test cases. This Diagram is explained as below.

2.1 Modeling of Activity Diagram
Modeling of Activity Diagram should be created on the modeling tool like RSA, Magic Draw. In this illustration an

Activity Diagram for the Hotel Management System is considered and described as below with figure 2.

An activity diagram of the Hotel management system is shown in figure 2 firstly as per diagram, the user visited the hotel and met with the receptionist and asked him/her about the requirements which they are giving to the customer. The receptionist checked his/her requirements (like room, rates, facilities etc.) and told him/her their details. Customer provides the requirement details to the receptionist and receptions checked the availability of the rooms. If the requirements matches with the customer requirement then receptionist informed him/her with their satisfaction, then receptionist has generated the bill. After the generation of the bill the customer paid the bill as parallel receptionist managed the database. Moreover, by paying the bill to the receptionist, the receptionist provided the receipt to the customer for the verification that his/ her submission of the bill is successful submitted. Finally the permission is granted and room is allocated to the customer. The control transfer points are handled by using swim lanes. Three swim lanes are considered, one for the customer, receptionist and for the manager who is handling this management. The input and the output of each activity are shown using parameter nodes.

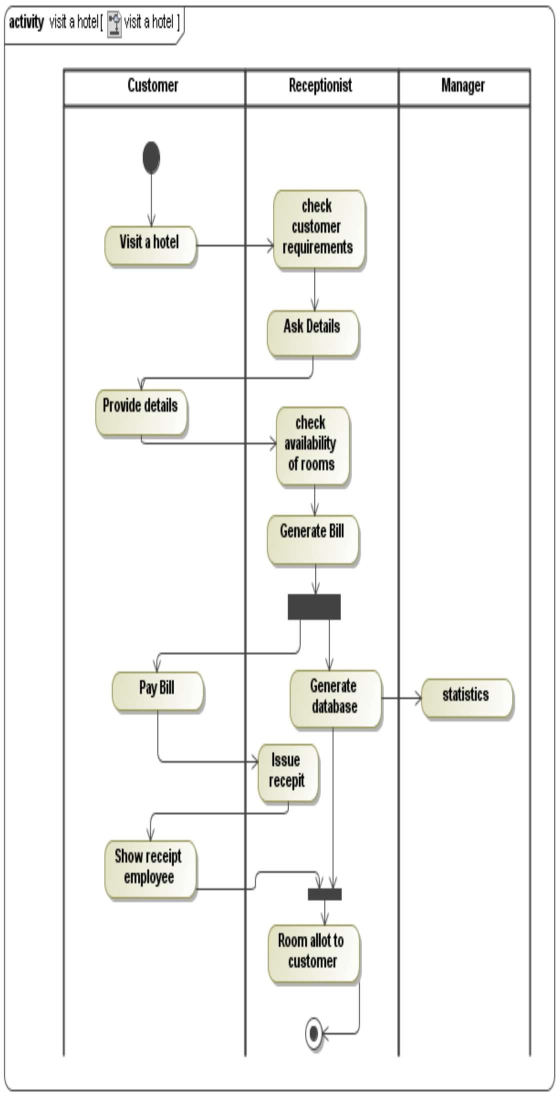

Fig 2: Hotel Management System Activity diagram

2.2 Automatic Generation of ADT

ADT is Activity Dependency Table which is automatically generated from the activity diagram. ADT is fully dependent upon the Activity diagram. The four columns of the ADT are:

i) Symbol: A unique number is assigned to the activities for their verification. From figure 1 first activity is performed by customer is visit a hotel, is assigned as 'A' symbol. As Similar scenarios is taken for other activities.

ii) Activity Name: It describes the activities which are performed by the actors. Example from Figure 2 visit a hotel is a activity which are performed by the customer.

iii) Dependency: It describes the dependency of the one activity on other activity. As shown in Figure 2 symbol 'A' has no dependency but as similar symbol 'B' is depending on the output of the Symbol 'A'.

iv) Controlling entity: Where it manages who is performing the action.

Table 1. Activity Dependency Table

Symbol	Activity Name	Dependency	Controlling Entity
A	Visit a Hotel		Customer
B	Check Customer Requirements	A	Receptionist
C	Ask Details	B	Receptionist
D	Provide Details	C	Customer
E	Check Availability of rooms	D	Receptionist
F	Generate Bill	E	Receptionist
G	Pay Bill	F	Customer
H	Generate Database	F	Receptionist
I	Statistics	H	Manager
J	Issue Receipt	G	Receptionist
K	Show Receipt to Employee	J	Customer
L	Show Receipt to Customer	K,F	Receptionist

From Figure 3 ADT is generated automatically by fetching the XMI code from the Activity diagram a dependency table is generated. A Modeling tool has generated the code file should be by '.XMI' extension. By extracting the relevant information from the XMI code; lexical tree will shows their dependencies behavior and interaction. After getting the dependencies and controlling entities automatically generate ADG.

2.3 Generation of Activity Dependency Graph

ADG is Activity Dependency Graph which is generated from the Dependency Table. ADG is required for the generation of the test cases where a required path is followed. From Dependency table it is not possible to find the exact path of the activities. Activity Dependency Graph is relies on the Dependency table. It is necessary to generate the ADG because it follows an exact path which is easy to generate the test cases. From the output of this table the activity dependency graph is generated.

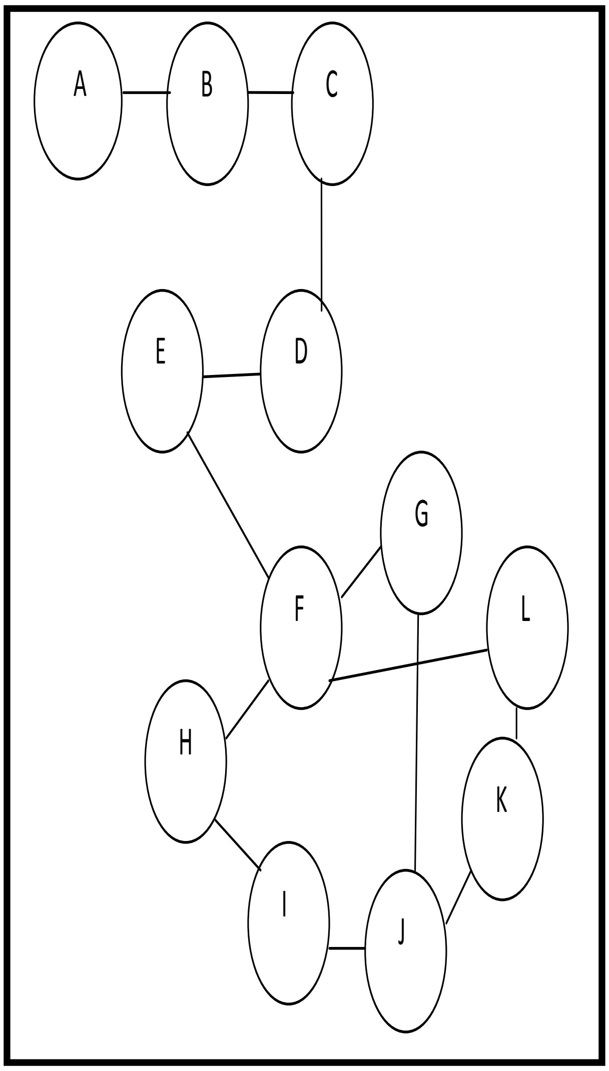

Fig 4: Activity Dependency Graph

From figure 4, ADG has taken the input values from the ADT as such ‘A’ is independent whereas ‘B’ is dependent on the ‘A’. Including ‘B’ depends upon other two activities i.e., ‘C’ and ‘D’. These same procedures are followed in every path.

3. PERFORMANCE EVALUATION

The final Test cases are generated using a particular defined algorithm called Generating TestCasesSuite. Genetic Algorithm is used for the generation of final test cases.

Algorithm Used:

```
1. To Apply Genetic Algorithm, assigning weightage (wi ) to node using the SDT table
a. Start the process with the start message assigns the value as one.
b. Using the dependency table check the next message.
c. Increment the value and assign it to the next.
d. If the current message is decision, then
i. Increment the value and assign it to the true side of the decision.
ii. And also for the false side of the decision.
e. Repeat the step until it reaches the end message.
2. To calculate fitness value
a. for each node calculate the number of incoming node(a) and the number of outgoing
nodes (b)
b. F = Σ n ( ai * bi ) + wi
i=1
3. Select the initial test data by random and calculate the fitness value for test data.
4. Generate the random number r
a. if r<0.8 perform crossover
b. else if r<0.2 perform mutation
5. Repeat this process until all the message paths have been covered
6. Best Test path generated.
7. End.
```

Test Case : A-B-C-D-E-F-H-I-J-K-L

Test Case : A-B-C-D-E-F-H-I-J-G-F

Test Case :A-B-C-D-E-F-G-J-K-L

Test Case : L-K-J-G-F

Fig 5: Generated Test Cases

Generation of test cases in early software development life cycle will reduces the complexity as well provides an efficient software testing planning.

4. CONCLUSION AND FUTURE SCOPE

We have implemented a semantic approach for the generation of test cases automatically from behavioral diagram i.e., Activity Diagram. From the Activity diagram a ADTs and ADGs are generated for the generation of test cases. It also achieves full predicate coverage. It is efficient and effective approach for the generation of test cases. In future, we will expect this approach for other UML diagrams to generate test cases and achieve higher coverage including with the prioritization of the test cases expecting with detection of faults.

5. REFERENCES

[1] B. Berenbach, D. Paulish, J. Kazmeier, A. Rudorfer. 2009. Software and Systems Requirements Engineering in practice. The McGraw-Hill Companies Inc., USA.

[2] Cartaxo, Emanuela G., Francisco G. Oliveira Neto, and Patrícia DL Machado. 2007. Test case generation by means of UML sequence diagrams and labeled transition systems. Systems, Man and Cybernetics ISIC. International Conference on IEEE.

[3] Prasanna, M., and K. R. Chandran. 2009. Automatic test case generation for UML object diagrams using genetic algorithm. International Journal of Advance Soft Computer Application.

[4] B.B. Agarwal, S.P. Tayal, M. Gupta. 2010. Software Engineering and testing. In Infinity Science Press, Jones and Bartlett, Hingham, Toronto.

[5] R. Black.2007. Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional. In John Wiley & Sons, Bulverde, Texas.

[6] Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall. 2010. Test Case Generation Based on Use case and Sequence Diagram. In International Journal of Software Engineering, IJSE Vol.3 No.2.

[7] C. Mingsong, Q. Xiaokang, and L. Xuandong. 2006. Automatic test case generation for UML activity diagrams. In International Workshop on Automation of software test, pp. 2-8.

[8] Magic Draw UML. Available from: www.magicdraw.com..

[9] Debasish Kundu and Debasis Samanta. 2009. A novel approach to generate test cases from UML activity diagrams. In Journal of Object Technology, 8(3):65–83.

[10] Stefania Gnesi, Diego Latella, and Mieke Massink. Formal test-case generation for UML statecharts. In Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer Systems Navigating Complexity in the e-Engineering Age, pages 75–84, Washington, DC, USA,