Empirical Comparison of Test Data Generation Techniques using Activity Diagrams

Ridham Khurana
Guru Gobind Singh Indraprastha University,
New Delhi, India

Anju Saha
Guru Gobind Singh Indraprastha University,
New Delhi, India

ABSTRACT
UML (Unified Modeling Language) is now a leading standard for defining software processes. Test data generation is advantageous in early phases of software development. Activity diagrams are user and developers' friendly because of the ease in their understanding. Many papers have presented techniques for test data generation using activity diagrams. These techniques have their own specific benefits considering required test data to be generated. On application of these techniques on same input i.e. activity diagrams, differences and similarities emerge evidently. These outcomes can provide clarity among testers, so as to decide upon the technique for test data generation depending upon the phase and type of test data required. In this paper we performed a comparative study of the five techniques of test data generation based on activity diagrams using ten examples**.**

General terms
Comparative study

Keywords
Activity Diagrams, Test Data Generation, Comparison.

1. INTRODUCTION
Presently UML is widely used by researchers for test data generation. Various UML diagrams like use case, state chart, class diagram etc are used for test case generation. Benefits from these diagrams are extracted to ensure simplified test data generation before actual testing phase. Multiple methods for test data generation have been developed, keeping in mind a particular scenario or availability of information or requirement of test data. Following the same trend, test data is extracted from activity diagram in many ways depending upon the priority set by the developers that which type of test data is required. In this paper we discuss five techniques given by Kim et al [1], Fan et al [2], Kansoamkeat et al [3], Heinecke et al [4] and Boghdady et al [5].
The paper is structured as follows. Section 2 provides a description of activity diagrams, section 3 discusses the related work of test data generation, section 4 describes the five techniques for test data generation, section 5 describes the example used, section 6 details the application of five techniques on the given example, section 7 provides comparison among techniques and section 8 presents the conclusion of the comparison.

2. ACTIVITY DIAGRAMS
The diagram that presents the sequence of activities to be followed in a system for getting a particular purpose done is called an activity diagram. Activities are represented by an oval and sequences are shown by directed arrows.
Conditional branches are shown with a diamond where one arrow makes an entry and multiple arrows exit. Merging of multiple branches is also done using a diamond. Parallelism can also be drawn using join and forks which are horizontal lines. All these are exemplified in the example activity diagram example figure 1.
Many times people do make comparison in a flow chart and an activity diagram but an activity diagram is different from flow chart in the sense that it can represent parallelism in activities and it does not allow entering or exiting of two arrows from a single activity i.e. an oval in a flow chart.

3. RELATED WORK
Testing is the most important phase in the software development lifecycle as it is the only phase that actually proves the correctness of the product with respect to any given specification. It takes the major portion of time of the entire development. Some may consider it easy but actually even the simplest code in software can take infinite combination of inputs for testing. Hence test data generation has acquired an eminent place in the development life cycle of software. Historically test data generation has been symbolic or dynamic with random, goal oriented and path oriented approach [6]. But strategies have taken a new turn bringing the test case development to requirement and design phases. UML diagrams, a strong tool for system design, is serving as the base for test data generation with each diagram focusing on different views of the system. By view, here we mean the perspective with which a designer/user looks at the system. Usually it is required that a path of execution is traced out using UML diagram and then the constraints in that path are solved with basic methods to extract test data. Standards for solving constraints have been laid out lately and followed using a number of tools but extraction of the execution path is what that has wide branches with UML diagrams as the input. Cle'mentine [7] et al proposed the use of use case diagrams and sequence diagrams to extract test scenarios through her elaborate technique using instantiated use cases and use case transition system in the process. The use of sequence diagrams with use case template and class diagram to generate test data is proposed by Monalisa [8]. Dymek [9] proposed to use relation among different UML diagrams for using test data generated, at a particular phase, in another phase. Samuel et al [10] presented an approach to draw valid sequences of transitions of object's state in the system and generate test data in order to traverse those transitions in object's state. Bandyopadhyay [11] et al suggests following only sequences that go through valid states of the objects of the system and extract test cases. Chen et al [12] introduced an algorithm to generate Interaction Finite Automaton (IFA) from use cases and then test cases from IFA automatically. Using class diagrams with OCL constructs to define constraints and behavior for test data generation using SMT solver is prescribed by Fujiwara et al [13] Apart from UML based test data generation goal oriented test data generation [14] and test data generation using symbolic evaluation [15] are alternate ways using code as input and not UML diagrams. Segmenting

programs [16] in order to reduce cyclomatic complexity for easier and complete coverage of the code while test data generation is proposed by Wang [16]. In this paper, we present empirical evaluation of the working of five techniques for test data generation using activity diagrams and a comparative study is presented further.

DESCRIPTION OF FIVE TECHNIQUES

Technique no 1

Test data generation using IOAD.

This technique converts the activity diagram into an IOAD (input output explicit activity diagram) that focuses on the external interaction of the system and ignores internal processing activities. It is used to derive test paths based on the inputs/outputs given/received from the user because tester rarely knows about the internal processing of the system being tested. Using this strategy and all-path as the coverage criterion all the interactions are exercised for appropriate functioning [1].

Technique no 2

Test data generation using sub activity diagrams

This technique analyses an activity diagram and looks for activities that are not individual activities rather a name for the group of activities and can be expanded as a separate activity diagram. This new activity diagram is inclusive in the previous one therefore called as the sub activity. Using this sub super activity relation fine details in an activity diagram can be tested where any particular activity performs a complete function. This technique uses path coverage with round robin strategy for sub activities included in the super activity diagram so that all combinations of paths are avoided rather only valid and executable combinations are taken [2].

Technique no 3

Test data generation using condition classification tree method

This technique uses all the conditional branches to find out as to which test case covers which of the branches of the conditions in the activity diagram and uses minimal test suite that covers all the branches [3].

Technique no 4

Test data generation for acceptance testing.

Users while acceptance testing, sometimes, do not know what is the expected input to the system and what is the expected output from the system. In order to make the system understandable the activity diagram of the given system is converted into Interaction Flow Diagram that shows the objects (input/output), in which the user will be interested. IFD also show the role played by the user in each activity where any input/output is entering/being receiving from the system, so that the clarity in the system usage is maximized. The IFD is converted to IFG(Interaction Flow Graph) with each loop traversed once for test path coverage and using depth first search on the graph that gives you the all the valid paths for test case generation.[4]

Techniques no 5

Enhanced test case generation technique

Normally activity diagrams are very complex, having the presence of forks, joins, conditional branches and merges, the presence of these symbols make the activity diagram look very clumsy. Therefore in this technique the activity diagram is first converted Activity Dependency Table (ADT) and then into Activity Dependency Graph (ADG) which is a simplified form of any activity diagram without losing any activity. Test paths are extracted from ADG using path coverage criterion with depth first search and are minimized using reduction for loops [5].

4. DESCRIPTION OF THE EXAMPLE

Here the activity diagram of a shipping company is taken Figure 1. The company allows the customer to place order in its system. The system then checks for the availability of stock specified in order. If available then bill is processed and shipping choices are specified. On verifying the authenticity of the payment, mode of delivery is asked. Finally the receiving is generated.

5. APPLICATION OF 5 TECHNIQUES ON THE EXAMPLE.

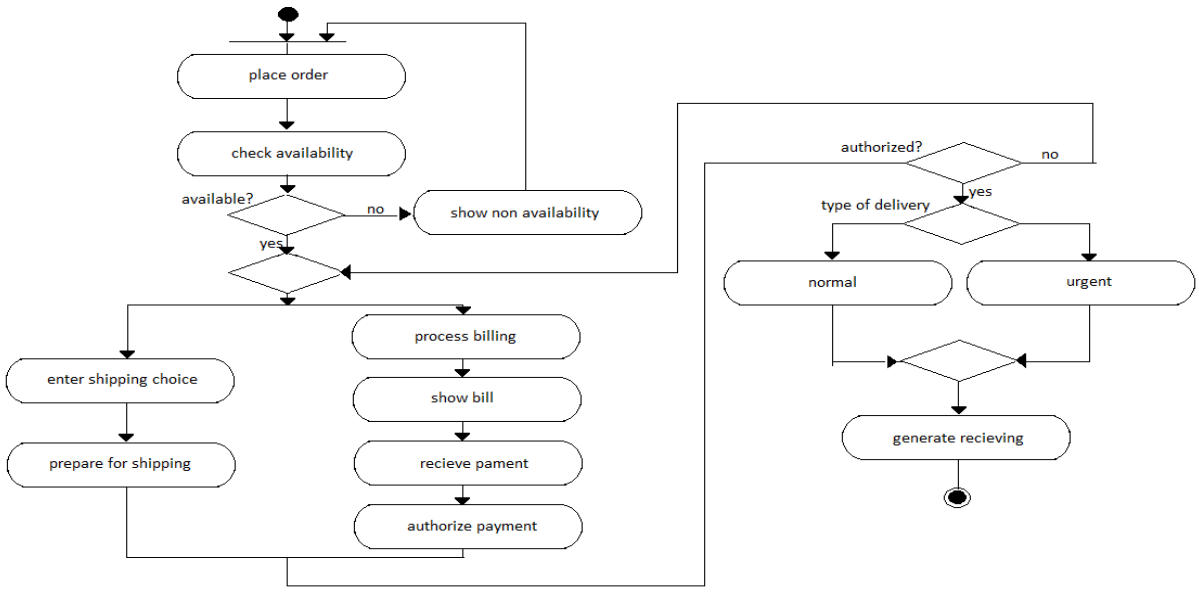


Figure 1. Example activity diagram

Applying Technique 1

Activity Diagram is converted into the form where user interactions are paid attention for test case derivation.

Converting the Figure 1 into IOAD, shown in Figure 2.

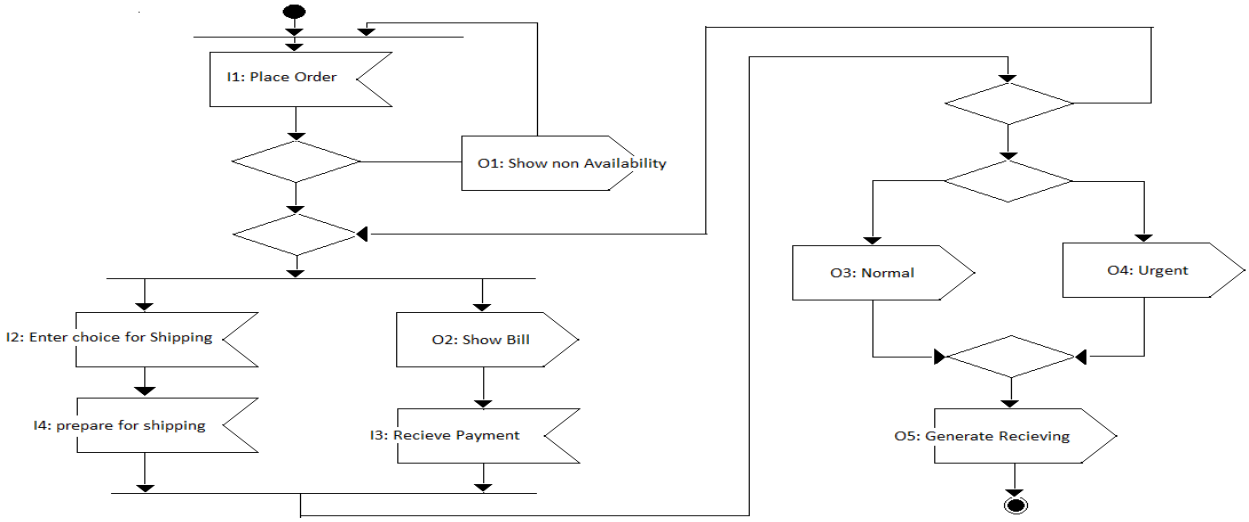


Figure 2. IOAD of the activity diagram in Fig 1.

Using single stimulus principle as proposed in technique 1. The following test paths will be generated

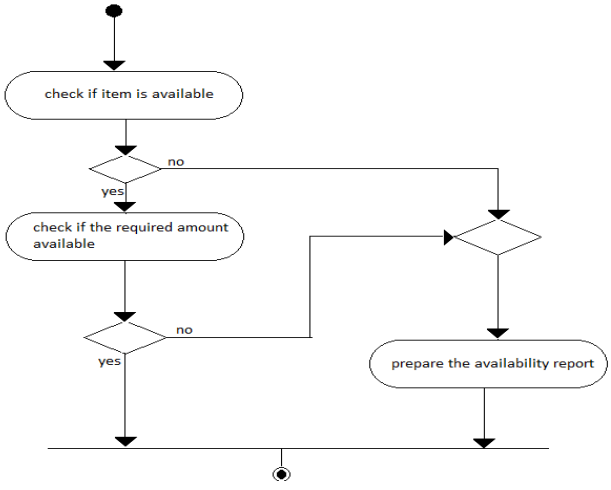
P1: I1-O2-I2-I3-I4-O3-O5
P2: I1-O1-O2-I2-I3-I4-O3-O5
P3: I1-O2-I3-I2-I4-O4-O5
P4: I1-O2-I3-I2-O2-I3-I2-I4-O4-O5

Keeping in mind the concurrent case there can be other paths as well but assuming the single stimulus and user interaction with the system the above test cases are sufficient to provide complete path coverage.

Applying Technique 2

Activity diagram is further expanded to show internals of any activity and more paths are found out that introduce new test cases.

In the given example "Check Availability" is not a simple activity rather it can be further expanded in the following way.



.

Figure 3. Sub activity diagram of "Check Availability" Activity of the Fig 1

There are three paths in this expanded sub activity diagram of "Check Availability". If all path combination technique is employed the then at most 12 test paths can be generated with 4 basic paths and 3 paths of this expanded sub activity making 4*3=12. But using the round robin technique as specified in [2] at most 7 test scenarios are sufficient to provide complete path coverage. In actual any path from activity "Check availability" will cover any one path of "Check availability" activity diagram hence two more cases are sufficient to provide complete coverage rather than combining each path of sub activity diagram Figure 3 with all possible scenarios of the Figure 1.

Applying technique 3

Conditions (diamonds) in the diagram are used to find out minimal test suite that covers all conditional branches.

With three conditions in hand the Figure 4 for the condition classification methodology will be generated. Four test cases with the shown relation among the conditions in the activity diagram are sufficient to provide complete path coverage. If separate test case for each branch of the condition is developed then there can be eight test cases but relation among them has reduced the number to four. This count of test cases is equal to what is produced in technique 1.

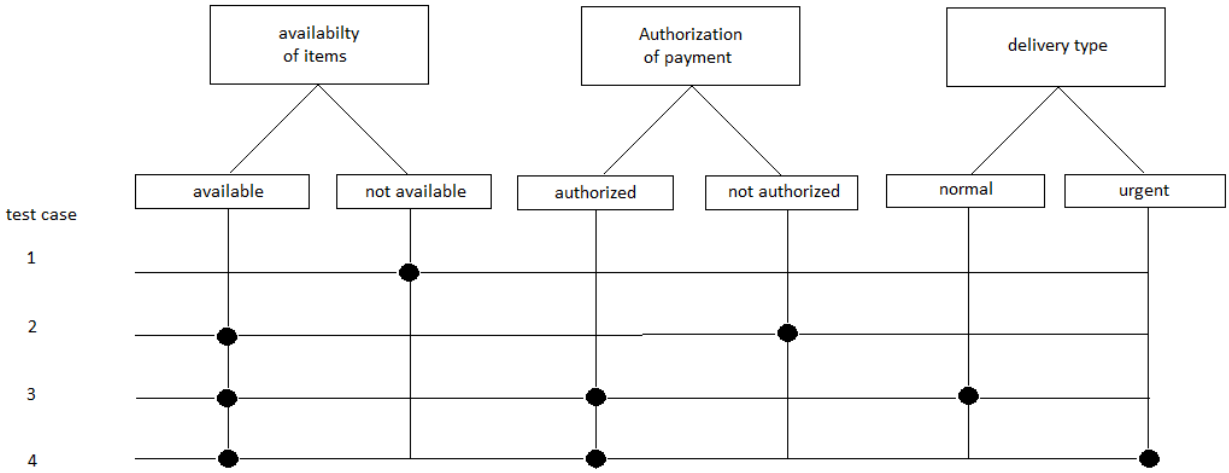


Figure 4. Condition classification of the activity diagram in Figure 1.

Applying Technique 4

Activity diagram is converted to IFD Figure 5 and then to IFG followed by complete path coverage for test data.

This technique converts the activity diagram into Interaction Flow Diagram (IFD) that fully exploits the view, a tester will have, while executing the system by adding various objects and roles a tester plays during an activity's execution.

Further IFD is converted into Interaction Flow Graph (IFG) that is traversed to generate four test paths. One each for two cycles and two for the conditional branches. IFG is not shown here.

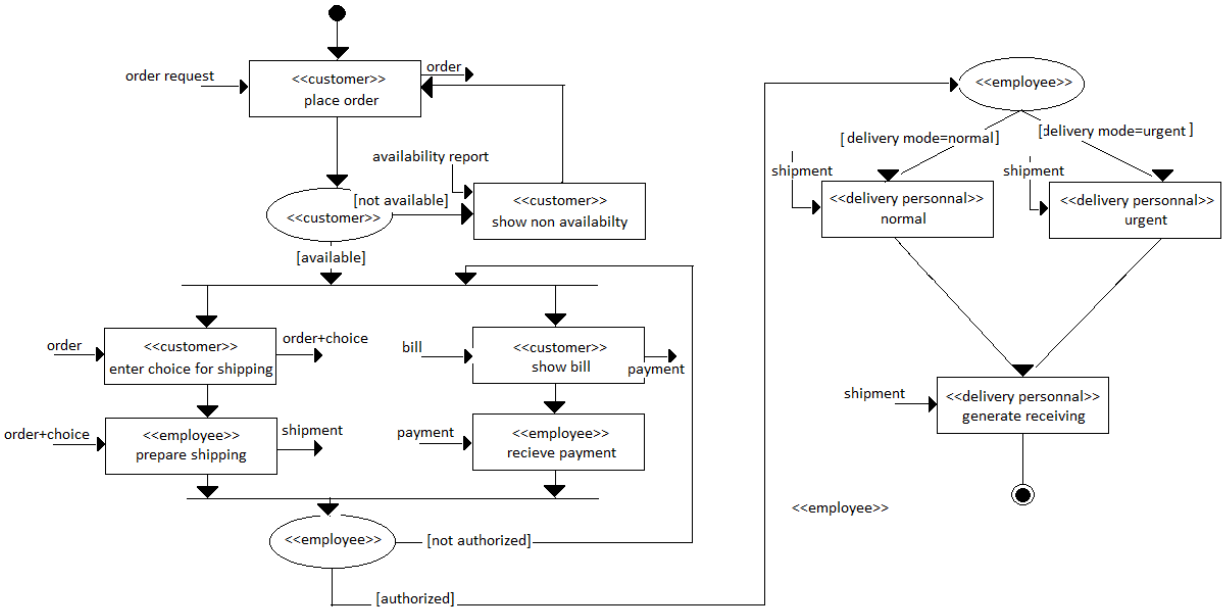


Figure 5. IFD of the given activity diagram in Fig 1.

Applying Technique 5

The activity diagram is converted into less clumsy "Activity Dependency Graph" which is enough for complete path coverage and it provides reduction in test paths with the number of loops it has.

Here the activity diagram has all its components named in the Activity Diagram Table (ADT) Table 1. Then reduction is carried out removing all the components other than activities and Activity Diagram Graph (ADG) is created without and with reduction here only the reduced/enhanced graph is shown.

.

Table 1. ADT of the activity diagram in Fig 1.

Activity name	No reduction	Reduction
Join 1	A	
Place order	B	B
Check availability	C	C
Decision	D	
Show non availability	E	E
Merge 1	F	F
Fork	G	
Enter choice for shipping	H	H
Prepare shipping	I	I
Process billing	J	J
Show bill	K	K
Receive payment	L	L
Authorize payment	M	M
Join 2	N	
Decision 2	O	
Decision 3	P	
Normal delivery	Q	Q
Urgent delivery	R	R
Merge 2	S	
Generate receiving	T	T
Return	U	U

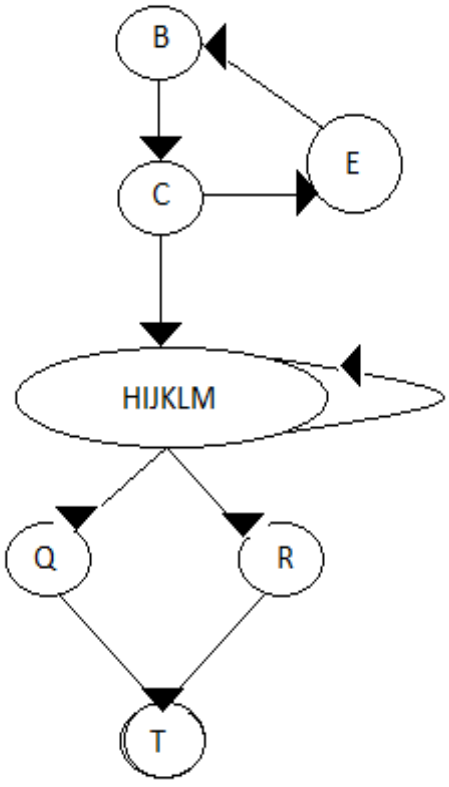


Figure 6. ADG after reduction of the ADT in Table 1.

The given Figure 6 with "all-path" coverage criterion and DFS based algorithm will generate following paths.

TEST PATHS

1.	B-C-B-C
2.	B-C-HIJKLM-HIJKLM
3.	B-C-HIJKLM-Q-T
4.	B-C-HIJKLM-R-T

TEST PATHS AFTER REDUCTION

1.	B-C-B-C-HIJKLM-Q-T
2.	B-C-HIJKLM-HIJKLM-R-T

This technique handles loops to the benefit of testers by reducing the number of test case equally to the number of loops.

6. COMPARISON

From above and other 10 examples on which these five techniques have been applied the following observations are concluded. Individual comparison between techniques is given as follows.

Technique 1 and 2 are only similar with regard to the input i.e. activity diagram, but then they take a different turn where input in Technique 1 is converted in IOAD and Input in Technique 2 is expanded for sub-activities.

Technique 1 and 3 are opposite because in technique 1 concern is the input to, and the output from the system and internal processing of the system is completely ignored whereas technique 3 give immediate importance to internal conditions and branches that will be followed with the given input test data.

Technique 1 and 4 are similar since they deal with activity diagram from user's perspective. Difference occurs in the detail the conversions in two, provide us. It is also observed that technique 1 focuses on concurrent situations only. Manual effort involved in the technique 4 for making tests of acceptance level reduces its benefits to some extent.

Technique 1 focuses on concurrent events whereas technique 5 focuses on simplification, coverage and reduction of test data.

Technique 2 expands activity diagram using sub-activity diagrams and technique 3 focuses on condition classification trees with valid sequences of execution of conditions as they are encountered. These two follow a different way but if a collaborated diagram from technique 2 is given as input to technique 3 they will generate same test cases as output.

Technique 2 and 4 follow a completely different view and two differently skilled testers are required to work upon the input activity diagrams for output i.e. test cases.

Technique 2 and 5 can be collaborated to provide an excellent test data generation technique since technique 2 provides you with maximal detail of the system and technique 5 can be helpful in reducing complexities of the detailed activity diagram and can produce lesser number of test cases.

Test cases generated using Technique 3 comes into action during coding phase while test cases designed using technique 4 are useful in acceptance phase. It is evident that these two follow a different perspective but with the same input can generate same test cases.

Technique 3 and 5 can generate same test cases but technique 5 has an edge over 3 because it reduces the test cases with the number of loops the system has. Technique 3 also holds an edge because of its simplicity.

Technique 4 and 5 are both detailed and well explained techniques for implementing automatic test data generation but they have a different focus. They take their inputs and convert them in very different form but may generate same set of test cases if there are no loops in the system.

7. CONCLUSION

1. Both technique 1 and 4 are develop test cases with user's perspective giving complete focus to what user expects from the system.
2. Technique 2 is beneficial when detailed test data is to be extracted wholly from activity diagrams for a system. Else if a nested activity diagram is used in all other techniques, the number of test cases generated equals test cases generated from techniques 1 and 3.
3. Technique 3 is a simplistic form of test data generation with less of effort required though basic execution style of the system and inter-relation of the subsequent conditions must be known to the tester for best application of the technique.
4. Technique 5 requires some time resource because of the tables and conversions it involves but complete automation can solve this problem and with clear observation it is the most efficient technique.

With various constraints, one or the other technique is suitable depending on the requirement and focus of the tester. All five techniques work well with different perspective but the fifth one introduces a special high by reducing the number of test cases in a loop condition as compared to other techniques.

Future work for the given study can be taking more and more of real life examples and deducing the most effective and suitable technique, out of these five, in different domains.

8. REFERENCES

[1] Hyungchoul Kim, Sungwon Kang, Jongmoon Baik, Inyoung Ko , "Test Cases Generation from UML Activity Diagrams", Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007, VOL 3 pp. 556-561.

[2] Xin Fan, Jian Shu, LinLan Liu, QiJun Liang, "Test Case Generation from UML Subactivity and Activity Diagram", 2009 Second International Symposium on Electronic Commerce and Security(ISECS '09) , VOL 2, pp. 244-288.

[3] Supaporn Kansomkeat, Phachayanee Thiket Jeff Offutt, "Generating Test Cases from UML Activity Diagrams using the Condition-Classification Tree Method", 2010 2nd International Conference on Software Technology and Engineering(ICSTE), pp. V1-62 – V1-66.

[4] Andreas Heinecke, Tobias Br¨uckmann, Tobias Griebe, Volker Gruhn, "Generating Test Plans for Acceptance Tests from UML Activity Diagrams", 2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems(ECBS), pp. 57-66.

[5] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed A. Hashim, Mohamed F. Tolba , 'An Enhanced Test Case Generation Technique Based on Activity Diagrams", 2011 International Conference on Computer Engineering & Systems (ICCES), pp. 289-294.

[6] J. Edvardsson, "A survey on automatic test data generation", Proceedings of the Second Conference on Computer Science and Engineering in Linkoping, pp.21-28, ECSEL, Oct 1999.

[7] Cle´mentine Nebut, Franck Fleurey, Yves Le Traon, Member, IEEE, and Jean-Marc Je´ze´ quel, Member, IEEE, "Automatic Test Generation: A Use Case Driven Approach", IEEE Transactions on Software Engineering, VOL. 32, NO. 3, March 2006, pp. 140-155.

[8] Monalisa Sarma, Debasish Kundu, Rajib Mall "Automatic Test Case Generation from UML Sequence Diagrams", 15th International Conference on Advanced Computing and Communications, 2007(ADCOM '07) pp. 60-67.

[9] Dariusz Dymek, Leszek Kotulski, "Using UML(VR) for supporting the automated test data Generation", Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008, pp. 3-8.

[10] P. Samuel R. Mall A.K. Bothra, "Automatic test case generation using unified modeling language (UML) state diagrams", IET Software, VOL. 2, NO. 2, pp.79-93, April 2008.

[11] Aritra Bandyopadhyay, Sudipto Ghosh ,"Using UML Sequence Diagrams and State Machines for Test Input Generation", 19th International Symposium on Software Reliability Engineering, Nov 2008 , pp. 309-310.

[12] Lizhe Chen, Qiang Li, "Automated Test Case Generation from Use Case: A Model Based Approach", 2010 3rd International Conference on Computer Science and Information Technology (ICCSIT), VOL 1, pp. 372-377.

[13] Shoichiro Fujiwara , Kazuki Munakata , Yoshiharu Maeda , Asako Katayama , Tadahiro Uehara ," Test data generation for web application using a UML class diagram with OCL constraints", Innovations in Sytems and Software Engineering, VOL. 7, No. 4, Dec 2011, Springer-Verlag London Limited.

[14] Bogden Korel, "Automated Test Data Generation for Programs with Procedures", Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis, ACM SIGSOFT Software engineering Notes Vol. 21 Issue 3, May 1996, NY, USA.

[15] Lori A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs" IEEE Transactions on Software Engineering, September 1976, VOL. SE-2, No 3, pp. 215-222.

[16] Lixin Wang, "A Program Segmentation Method For Testing Data Generating Based On Path Coverage", 2010 IEEE International Conference on Software Engineering and Service Sciences(ICSESS), pp. 565-568.