Testing from UML Design using Activity Diagram: A Comparison of Techniques

Muhammad Touseef Ikram
Department of CS
Capital University of Science & Technology,
Islamabad, Pakistan

Naveed Anwer Butt
Department of CS
University of Gujrat,
Gujrat, Pakistan

Altaf Hussain
Department of CS
Capital University of Science & Technology,
Islamabad, Pakistan

Aamer Nadeem
Department of CS
Capital University of Science & Technology,
Islamabad, Pakistan

ABSTRACT
UML diagrams present the graphical representation of the system. Model-driven testing not only helps in early identification of faults but also results in reducing the testing effort at the later stages of SDLC. This paper intends to identify and make a critical review of different techniques for test case generation using UML activity diagrams (UAD). System activity diagram is used to depict the different dynamic aspects of the system. UAD not only presents the sequential or concurrent activities but also presents the conditional and parallel activities. For this literature survey different aspects like test case generation, test automation, and test case prioritization & minimization using UAD has been explored. The analysis of the literature portrays that extensive literature exists regarding automation of the testing using various aspects of activity diagrams. Similarly, test cases prioritization has also been explored using the activity diagrams incorporating manual, automated and semi-automated techniques.

Keywords
Test Case Generation, UML Activity Diagram, Software Testing, Test Cases, Test Automation

1. INTRODUCTION
Unified Modeling Language (UML) is a standard language for modeling software and to model business processes and has emerged as a common standard for modeling the object-oriented paradigms [1] [9]. One of the most common reasons of software failure is the presence of errors in the analysis and design phase which makes them more important than code [3]. If the design artifact of software is used for the test case generation; it will not only result in early planning for test cases but also facilitate in eliminating most of the UML design errors and their propagation in later phases [6]. UML sequence diagram, activity diagram, class diagram, and state chart diagram are most commonly used as an artifact for test case generation [1]. This paper review is primarily focused on the analyzing the test case generation process from UML design using only the activity diagram.

However, the extraction of test information from activity diagram is not a simple task due to the following reasons:

- Concepts in the activity diagram are represented at higher level of abstraction as compared to other diagrams &
- The presence of concurrent activities and loop structure in the activity diagrams also results in the path explosion.

A number of studies have explored in which authors have considered the UML activity diagrams and have generated different types of graphs from it considering different coverage criterion like path coverage, basic path coverage or simple path coverage. UML activity diagrams have been utilized by a number of studies covering different aspects of software testing. Some of the studies have used the activity diagrams for the test case generation and prioritization using the gray box, black box, and model-based testing paradigm. Some studies are focused on the test case minimization for the optimal test coverage with least effort. Some have talked about the automation of the testing technique utilizing the activity diagrams. A few of the studies have explored the aspect of test sequence generation which is an important activity in testing.

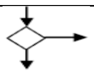
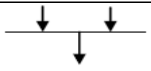
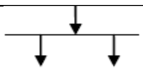
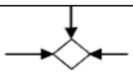
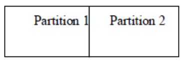
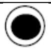
Following are a few of the hypothesis which have been kept in mind while conducting an exploratory survey of existing literature:

1. Are there any techniques (automated or semi-automated) for the test cases generation by using UML activity diagrams?
2. What Coverage criterion has been incorporated for the extraction of test cases from UAD?
3. What types of techniques & approaches used for test case prioritization and reduction?

2. UML ACTIVITY DIAGRAM
UML activity diagrams are behavioral diagrams which depict the internal behavior of different operations of a program with the help of nodes and edges [2]. UML activity diagrams have been used in different domains for work-flow representation [6]. Activity diagrams have extracted their core conception from Petri-nets, flow charts and state transition diagrams with the difference that it supports concurrent activities, loops and event-driven behavior [6][2]. A UAD (UML Activity Diagram) is used to represent different activities, sub-activities, transitions, decision points, guard conditions, concurrent activities, branch, merge, swim lanes, join forks and etc. [11][7]. An activity diagram starts with one start activity and ends at one final activity [5]. The table below shows the description of different artifacts of UAD:

Table 1: Basic symbols used to model Activity Diagram

Name	Symbols	Description
Initial Node	●	Initial node is represented by filled circle.
Activity	▭	Activities are represented by the rounded rectangle.
Edge	→	Transitions are drawn as directed arrows to show the control flow among activities.

Decision Node		A decision node is presented with a diamond usually having an inward stream and two outward streams.
Join		Join is represented with a line also called the synchronization bar receiving multiple inward and one outward flow.
Fork		The fork is also represented with a synchronization bar which receives one inward and multiple outward flows.
Merge		Merge operation is denoted by a diamond shape box with multiple inward flows.
Swim Lanes		Describes the concurrent flow of activities and by whom these are performed.
Final Node		Final node is represented by filled circle with an outline.

3. EVALUATION CRITERIA

Since numerous methods have been suggested and proposed for generation of test cases from UML activity diagram and these techniques have their own benefits and drawbacks. Therefore, it is necessary that some evaluation criteria must be set for the evaluation of these techniques. Following are a few of the parameters which are identified in evaluating the techniques. All the techniques to generate test cases from UADs are evaluated on these parameters:

Intermediate Representation: Usually the information extracted from the UAD is converted into some intermediated transformation so that it can be cleaned and further utilized for the test case generation. This parameter helps in evaluating that what are different representation forms that are being used in generation of test cases from UADs.

Coverage Criteria: Different coverage criterions are used for the evaluation of the testing techniques. This aspect highlights the most commonly used coverage criterions in test case generation.

Automation: Another challenging task in generation of test cases from UADs is to automate the technique for the identification, prioritization and minimization of test cases generated. This evaluation criterion highlights the automation level of the analyzed technique.

Tool Support: In test case generation tool support is regarded as a promising feature for the success of the proposed technique. This aspect in our evaluation criteria depicts the availability of different tools for the automated extraction of test cases from the UAD.

Case Study: This aspect depicts the practicability of the studied techniques in some domain and possible evaluate a technique as “Yes” and “No”.

4. LITERATURE REVIEW

In this literature survey we have classified the literature according to different aspects of testing from UML design using activity diagram:

4.1 Automated Test Case Generation Techniques using Activity Diagrams

In [4] Mingsong et al. presents a technique for test cases generation specifically for the programs written in JAVA and UADs as design specifications. This proposed method at first generates random test cases from Java programs. Then the execution traces of the program are generated by executing the Java program along with generated test cases. Additionally for minimization of generated test set it compares the activity diagram with the program running traces. Three coverage criterions namely transition, simple path coverage, and activity coverage are addressed for activity diagrams. One limitation associated with this approach is that it is restricted only to those UML activity diagrams which don’t contain or consider loops or concurrency artifacts. The proposed technique was implemented as a software prototype called AGTCG.

Chen et al. [5] developed a methodology for test cases selection from randomly generated test cases set considering UML activity diagrams for the coverage criteria for Java programs. To generate abundant random test cases the author has used a Java program. In this paper, activity diagrams are employed for automatic generation of test cases and later on providing a reduced test suite for Java Programs. As there is the random generation of test cases but this randomness cannot assure that the required coverage can be achieved with the selected test cases. The online stock exchange system is used as a case study. Further, the research has been comprehended with a tool support allowing the user in constructing, editing, and analyzing the UADs.

In [8] a technique for automatic generation of a test sequence is presented which is an important activity during testing by using UML activity diagram. The proposed technique first converts the activity diagram into the XML file which is further traversed to find the test sequences. To enrich the study and generate the test sequences Verma & Arora have proposed two algorithms.

First, an algorithm is used to extract the incoming and outgoing edges from the activity diagram for each node. The second algorithm utilizes these edges for test sequences generation and also optimizes the test sequences by rejecting the infeasible paths. Comparisons of the proposed technique with the earlier techniques which involve an intermediary model or graphical representation depict less overhead and less cost involvement. Another benefit of the proposed approach is that it involves lesser infeasible test cases because of the optimization algorithm

An automatic technique incorporating the concurrent activities in activity diagram is described in [10]. Due to the existence of fork-join nodes in the activity diagram there can be a lot of test scenarios which adds difficulty to the test case generation process. The proposed technique also considers the domain dependency between the concurrent activities to limit the exponential growth of the test scenarios. The dependency information between activities inside the fork and join pairs helps in producing only the feasible or possible test scenarios and thus help in dwindling the generation of infeasible test scenarios.

A technique has been proposed by Sun et al. [11] for automatic generation of test cases from activity diagrams exploiting the model driven testing aspect. To generate the test cases automatically from the tool first they preprocessed the data and then extracted its activity set and transition set (intermediate representation) which was further used to generate the test sequence and test scenario. The transformation rules are implemented to generate scenarios considering concurrency coverage criteria specified by the tester. With the adaptation of the TSGen tool testing can be scheduled earlier which helps in better test planning & also save the efforts.

4.2 Test case generation considering various techniques using Activity Diagrams

In [1] Kumar and Bhatia presented a technique to generate test case by extracting and combining test cases from various UML diagrams like class diagram, state diagram, and activity diagram. The crux of the paper is that the author has combined the information from multiple diagrams to generate dynamic test cases. One of the practical and powerful implications of combining information from multiple artifacts is that these test cases cover the broader aspect of dynamic testing. Online Hotel Reservation System is used as a case study to generate dynamic and efficient test cases covering both static and dynamic aspects of the software system.

Wang Linzhang et al. [2] have utilized Gray-Box technique for test cases generation from UADs. Gray-box testing technique generates test cases by representing the expected behavior of the system along with the structure of the software and the high-level design models of the software under testing. Hence, the advantages of black box testing and white box testing techniques are exploited by the gray box testing technique. Basic path coverage criterion is used for the extraction of test cases and scenarios from the UADs and the basic condition in this regard is that the loop must be executed at most once which helps in avoiding path explosion.

Hetal and Shinde [3] proposed the methodology for minimization of generated test cases. It uses the model driven testing technique for the optimal test suite generation. This technique intends to study the changes in the design document and its impact on the generated test cases resulting in identifying common and uncommon test cases. Further a focused study is carried out using uncommon test cases resulting in better resource utilization. Test cases prioritization is performed later on using code coverage, time and test effort required for common and uncommon test cases execution.

Ye et al. [16] have presented an approach based on regression testing of the SUT using the activity diagram. Various versions of the same software were analyzed to study the impact of the different modifications performed in the system by an automated approach considering different revisions of the activity diagrams using the path coverage as a coverage criteria. Execution traces are extracted from the execution of software. From the execution traces, it revises the activity diagram and then constructs the new activity diagram automatically incorporating the revisions in the software. This not only helps in identifying the affected path but also reflects the new paths. The major contribution and a comparison with the existing literature depict the automation of identification process of changed parts and generation of new test cases for testing new behaviors.

This paper [6] describes another way of generating test cases from the UADs by using AC grammar. For the generation of test case first the UAD is transformed into a grammar called the activity convert grammar which is then used for the test cases generation. The AC grammar based technique resulted in all paths coverage and better performance as compared to existing techniques. The conversion of the activity diagram into AC grammar is a three step process. Firstly activity diagram is converted into Activity Dependence (AD) table, and then AC grammar is generated from the table which ultimately results in test case generation. The path coverage criteria and comparison of the results with path coverage have been benchmarked by the authors. The results produced depicted that path coverage technique was unable to detect errors while their proposed technique detected four design errors. One limitation with this study is that they have only tested the results on the simple activity diagram and haven't considered the activity diagrams containing fork, join, and loops.

Patel & Patil in [7] has made a comparison of the two techniques proposed for the automatic test case generation using UADs and presented the results in the form of graphical analysis. Both concurrent and non-concurrent activities have been considered by the test case generation technique. In the proposed solution they have not only covered the loop testing but also considered the concurrent activities by developing an innovative test coverage criteria namely activity path coverage criteria. They have not only utilized both the techniques but also have made a comparison of the techniques in the form of a statistical graph of generated test cases from both the techniques. One technique used in this paper first enriches the UAD with the test information, which further lead to activity graph generation from the UAD and further generates the test cases from the activity graph. Similarly, second technique utilizes certain variables like TDN, RNN, TDNI, RNNI, etc., to extract possible paths. They have implemented both the algorithms to automatically generate test cases. One of the basic limitation with this technique is that it only covers the UAD and considers only one use case at a time and doesn't consider the infinite loop if present in an activity diagram.

Khurana & Saha [18] have made a comparison of five different techniques of test data generation using activity diagrams. The five techniques used in this paper for exploratory study are: (1) test data generation using IOAD (input-output explicit activity diagram) (2) test data generation using sub activity diagrams (3) test data generation using condition classification tree method (4) test data generation for acceptance testing (5) Enhanced test case generation technique. They have made the comparative study of these techniques on shipping company example. In techniques 1 and 4 the major emphasis is on the user's perspective meaning what user expects from the system. When the test data is extracted from the activity diagram then the technique 2 is beneficial whereas when there are complex activity diagrams for a system then the number of test cases generated is almost equivalent to the test case generated through techniques 1 & 3. As a comparative case with other approaches, technique 5 takes a lot of time because of the involvement of tables, graphs and conversions in it. However, the fifth technique results in a reduction of test cases in case of complex activity diagrams involving loops when compared with other techniques.

The technique presented in paper [20] converts the UAD 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. This technique deals with activity diagram from user's perspective. It is also observed that technique focuses on concurrent situations only. This technique concerns about the input to, and the output from the system and internal processing of the system is completely ignored.

This paper [21] presents a technique that analyzes 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.

This technique [22] uses all the conditional branches in finding out which branches of the conditions of UAD are covered by which of the test cases by using a minimal test suite. Conditions (diamonds) in the diagram are used to find out minimal test suite that covers all conditional branches. This technique gives immediate importance to internal conditions and branches that will be followed with the given input test data.

4.3 Articles involving the test case minimization techniques using UML Activity Diagrams

In this section, we present the test case prioritization approaches using UADs considering that testing performed in an optimized manner not only results in cost and time reduction but also facilitates in end user requirement satisfaction in an effective and efficient fashion.

In paper [9], Sabharwal et al. presented a test case generation process based on Genetic Algorithm for the test path prioritization which must be tested first. In this regard, they have used both UML Activity and sequence diagrams for extracting the test paths. The proposed approach makes use of concepts of information flow (IF) model, Genetic Algorithm (GA), and stack based techniques to identify the critical path clusters. The application of genetic algorithms resulted in optimization and improvement in the efficiency of testing process. A stack-based approach is considered to assign weights to different nodes of the activity diagrams & incorporating the modifications in the testing requirements. For this sake, they have made use of FAN-IN and FAN-OUT techniques. Later by using the weights assigned they have calculated the strength and complexity profile of each test sequence generated and ultimately test paths are prioritized with the higher strength and complexity. In the study, they found that the proposed technique proved to be significant in fault location identification during implementation resulting in test effort reduction. One of the limitations of the study is that the proposed technique has not been tested for the complex scenarios.

An approach, for test cases integration and prioritization has been proposed by Swain et al. [12] from UML activity and communication diagram to generate cluster level test cases. They have introduced an intermediate tree representation named COMMACT tree built from the communication and activity diagrams. Then a traversal of COMMACT tree results in extraction of the test cases considering not only simple predicates but also the conditional predicates. The approach produced a prioritization metric considering method activity sequence and associated artifacts under guard conditions. Their approach resulted in non-redundant and prioritized test scenarios along with adequacy test coverage.

In this paper, an attempt has been made to generate optimized (best fitted) and prioritized test cases from both activity & collaboration diagram using the constrained based genetic algorithms. In the proposed system, they have made use of combinatorial optimization technique by making an amalgam of transition coverage with the genetic algorithms. The core theme of the C-GA is that it takes all the possible values as input and then to minimize the input domain constrained genetic algorithm is applied. Further by using fitness function optimal test set is extracted. The fitness function assigns a weight to each and every event in the input domain and more weight is assigned to those events that have more decision points i.e., branches. As a basic principle to minimize the number of errors in the test case generation they have also proposed an error minimization technique.

Jena et al. [14] presented a test cases prioritization technique using UADs by using the genetic algorithms. This technique first converts the UAD into activity flow table which is further converted into an activity flow graph. Up till this point, this approach uses the common techniques because a lot of techniques talk about the transition of activity diagram into a graph. Activity coverage criterion has been used in this article for the traversal of flow graph extracted from the activity diagram for test cases generation using the depth-first search method by involving activity diagrams from multiple domains. Furthermore, generated test cases are optimized using a simple genetic algorithm which reduces the number of test cases generated. A comparison of proposed technique with state-of-the-art describes this technique also take conditions, interactions, concurrent paths and asynchronous activities into consideration. In their future work they want to automate the proposed approach and also want to make an exploratory study by applying this technique on other UML design diagrams.

In [15] a technique has been proposed for test set prioritization based on the structural artifacts on the UML activity diagrams. In this technique, the UAD is first converted into a tree structure. Then a weight is assigned to each branch of the tree on the basis of the probability of defect occurrence and the complexity of the each branch of the extracted tree. This technique not only covers the simple aspect of activity diagrams but also considers the fork and join constructs by assigning them the highest weight and priority.

An approach for the prioritization of test case is proposed by Fernandez and Misra [17] using software risk information. Firstly different risk factors were identified that can affect the software and then the other factors that can affect the risk factors were extracted. Further, the experimental study was conducted involving more than one hundred IT professionals for the validation of the proposed method. The proposed method helped in rearranging the test effort by taking into consideration the risk assumed by the end user considering risk factor. One of the practical implications of the proposed approach is that it can help in improving the efficiency & effectiveness when applied to complex and real projects.

5. EVALUATION OF TEST CASE GENERATION TECHNIQUES

The existing state of the art techniques utilize different additional methods for the test cases generation from the UAD and most commonly used methods are trees, graphs, genetic algorithms (GA), labeled transition systems (LTS) and finite state machines (FSM). From the systematic literature review of the testing techniques, it is established that a few of the techniques focus on resolving the problem of generation of redundant paths during generation of test cases from the UADs. Mostly in the existing techniques behavioral diagrams are used which have an associated drawback of limited test coverage and some result in redundant test cases with less test coverage. The techniques mostly generate a large number of test sequences and test cases for achieving a good result. There is still a gap of the automation of the testing process from the UML activity diagrams, most of the techniques are not automated, some have a few aspects of the process automated and some utilize the information from other UML diagrams to complete the test case generation process. Some approaches have described the theoretical aspects of the process with a few examples and case studies. However, the proposed approaches generate test cases for all the types of the testing (white box, black box & gray box) and ascertaining almost all the levels of testing.

6. CONCLUSION

In this literature review, we have explored the different state of the art techniques utilizing UADs for the generation of test cases. We have categorized the reviewed techniques into automated and semi-automated techniques involving different testing approaches like gray-box, model driven and integration testing. Further, we have also reviewed the techniques used for test case optimization and prioritization using UAD. For most of the test case prioritization techniques, genetic algorithms are used. In most of the studies, activity diagrams are first converted into some intermediate state which is further converted into a graph or tree to generate the test cases. Different coverage criterion has been used by different techniques for the generation of test cases and their optimization. The analysis of the literature depicts that much of the existing work has been highly concentrated on the simple UML activity diagrams and a few of the studies have talked about complex UAD involving the adaptation of concurrency, joins, forks, and loops. To conclude it can rightly be said that there is still a room for automatic test case generation from UAD combining multiple coverage criterions and considering complex activity diagrams.

Table 2: A comparison of different techniques

Technique	Testing Aspect	Testing Technique	Case Study	Test Criteria based on Activity Diagram
Kumar and Bhatia (2012)	Test Case Generation	___	Hotel Reservation System	Pre-Condition, Post Condition, Guard Condition
Linzhang et al (2004)	Test Case Generation	Gray Box Testing Technique (DFS)	ATM	Basic Path Coverage Criterion
Hetal and Shinde (2014)	Test Case Generation and Minimization	Model Based Testing Technique	Hall Booking	___
Mingsong et al. (2007)	Test Case Generation	Program Execution Traces	JAVA Applications	Activity, Transition and Simple Path Coverage
Chen et al. (2007)	Test Case Generation	Black Box	On-line stock exchange system	Yes (all)
Pechtanun & Kansomkeat (2012)	Test Case Generation	AC grammar	NO	Path Coverage
Patel & Patil (2013)	Test Case Generation	Model Based Test case generation Technique	JAVA Implementation	Activity path coverage, Loop coverage and Concurrent Activities
Verma & Arora (2014)	Test Sequence Generation	XML	NO	Path coverage criteria
Sabharwal et al. (2011)	Test Case Prioritization	Genetic Algorithm	Student Enrolment system	Path coverage criteria
Sapna & Mohanty (2008)	Test Scenario Generation	Domain Dependency	NO	Activity, Transition & path adequacy criteria
Biswal et al. (2011)	Test Case Prioritization & Optimization	Constrained Genetic Algorithm	ATM cash withdrawal	Transition Coverage Criteria
Jena et al. (2014)	Test Case Optimization	Genetic Algorithm	ATM Withdrawal system	Activity Coverage Criteria
Ye et al. (2012)	Test Case Generation	Integration Testing	Online stocking exchange system(OSES)	Path Coverage Criteria
Andreas Heinecke (2010)	Test Case Generation	Applied Modified Depth-First-Search Algorithm	Traveler problem	All path coverage criterion
Biswal (2008)	Test Case	Depth first	ATM	Path Coverage Criteria

(2008)	Generation	search		
Philip Samuel (2009)	Test Case Generation	Grey Box	ATM	Path Coverage Criteria
Kim et al. (2007)	Test Case Generation	I/O explicit Activity Diagram (IOAD)	Order Processing System	Path Coverage Criteria
Boghdady et al. (2011)	Test Case Generation	Activity Dependency Graph (ADG)	ATM	All branch, All predicate, All basis path coverage criteria

Table 3: A comparison of different techniques (PROS & CONS)

Technique	Technique	Advantages	Disadvantages
Sun	Transformation-based approach	Considers conditional activities along with parallel activities of system under testing for test case generation.	It ignores the loop conditions which requires that some activities must be repeated satisfying a particular condition.
Kim et al.	I/O explicit Activity Diagram (IOAD)	It controls the redundant test case generation by overwhelming the internal input and output events.	Test cases are generated by using basic path coverage criterion and consider only the single occurrence of each activity. This will result in the execution of the activities in the loop only once. This technique will also remove the redundant edges and nodes while intending to generate the representative paths from the basic paths.
Fan et al.	Integration testing (Bottom-up strategy) & (round-robin strategy), Functional decomposition	Due to Round robin strategy this technique generated the least number of test cases when it is compared with the other techniques.	As this technique transforms the each activity into a sub activity which results in increase in the number of test cases generated as compared to other techniques.
Swain et al.	COMMACT tree	Combines the data from both UML Activity and Sequence Diagram which results in detection of fault from both the diagrams.	Redundant test cases generation
Boghdady et al.	Branch coverage criterion and Depth First Search (DFS) traversal technique	Incorporates the hybrid coverage criterion Covers all the conditions, branches and basic paths however loops are considered only zero or one times	The basic drawback associated with this technique is that it considers the execution of loops at most once.
Wang et al.	Table representation DFS algorithm	Generates the test cases directly from UADs Completely based on UML models Mostly automated	Simple fork-join
Pechtanun & Kansomkeat	AC Grammar	Helps in better identifying the dependency between activities.	Results have been tested on the simple activity diagram Didn't consider activity diagrams containing fork, join and loops.
Boghdady et al.	Activity Dependency Graph (ADG) Activity Dependency Table (ADT) Cyclomatic Complexity	Test coverage criteria covers all the types of nodes like decision, fork, merge, join, activity and object nodes and also covers the conditional threads and loops.	It executes the loop activities only once.

7. REFERENCES

[1] Kumar, R., & Bhatia, R. K. (2012). Interaction Diagram Based Test Case Generation. In Global Trends in Information Systems and Software Applications (pp. 202-211). Springer Berlin Heidelberg.

[2] Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., & Guoliang, Z. (2004, November). Generating test cases from UML activity diagram based on gray-box method. In Software Engineering Conference, 2004. 11th Asia-Pacific (pp. 284-291). IEEE.

[3] Ms. Hetal J. Thanki, Prof. S.M.Shinde. "Test Case Generation and Minimization using UML Activity Diagram in Model Driven Environment", International Journal of Computer & organization Trends (IJCOT), V9 (1):41-44 June 2014.

[4] C. Mingsong, Q. Xiaokang, and L. Xuandong, “Automatic Test Case Generation for UML Activity

Diagrams”, In Proc. of the International Workshop on Automation of software test, pp. 2-8, 2006.

[5] Chen, M., Qiu, X., Xu, W., Wang, L., Zhao, J., & Li, X. (2009). UML activity diagram-based automatic test case generation for Java programs. The Computer Journal, 52(5), 545-556.

[6] Pechtanun, K., & Kansomkeat, S. (2012, June). Generation test case from UML activity diagram based on AC grammar. In Computer & Information Science (ICCIS), 2012 International Conference on (Vol. 2, pp. 895-899). IEEE.

[7] Patel, P. E., & Patil, N. N. (2013, April). Testcases Formation using UML Activity Diagram. In Communication Systems and Network Technologies (CSNT), 2013 International Conference on (pp. 884-889). IEEE.

[8] Verma, V., & Arora, V. (2014, May). A novel approach for automatic test sequence generation for java fork/join from activity diagram. In Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on (pp. 1611-1615). IEEE.

[9] Sangeeta Sabharwal, Ritu Sibal and Chayanika Sharma, “Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams”, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2, May 2011.

[10] Sapna, P. G., & Mohanty, H. (2008, December). Automated Scenario Generation Based on UML Activity Diagrams. In Information Technology, 2008. ICIT'08. International Conference on (pp. 209-214). IEEE.

[11] Sun, C. A., Zhang, B., & Li, J. (2009, August). TSGen: A UML Activity Diagram-Based Test Scenario Generation Tool. In Computational Science and Engineering, 2009. CSE'09. International Conference on (Vol. 2, pp. 853-858). IEEE.

[12] Swain, R. K., Panthi, V., Mohapatra, D. P., & Behera, P. K. (2014). Prioritizing test scenarios from UML communication and activity diagrams. Innovations in Systems and Software Engineering, 10(3), 165-180.

[13] Biswal, B. N., Nanda, P., & Mohapatra, D. P. (2010). A Novel Approach for Optimized Test Case Generation Using Activity and Collaboration Diagram. International Journal of Computer Applications, 1(14), 67-71.

[14] Jena, A. K., Swain, S. K., & Mohapatra, D. P. (2014, February). A novel approach for test case generation from UML activity diagram. In Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on (pp. 621-629). IEEE.

[15] Sapna, P. G., & Mohanty, H. (2009, July). Prioritization of scenarios based on UML Activity Diagrams. In Computational Intelligence, Communication Systems and Networks, 2009. CICSYN'09. First International Conference on (pp. 271-276). IEEE.

[16] Ye, N., Chen, X., Ding, W., Jiang, P., Bu, L., & Li, X. (2012, July). Regression Test Cases Generation Based on Automatic Model Revision. In Theoretical Aspects of Software Engineering (TASE), 2012 Sixth International Symposium on (pp. 127-134). IEEE.

[17] Fernandez-Sanz, L., & Misra, S. (2012). Practical Application of UML Activity Diagrams for the Generation of Test Cases. Proceedings of the Romanian Academy, Series A, 13(3), 251-260.

[18] Khurana, R., & Saha, A. (2012). Empirical Comparison of Test Data Generation Techniques using Activity Diagrams. International Journal of Computer Applications, 51(7), 13-19.

[19] Kansomkeat, S., Thiket, P., & Offutt, J. (2010, October). Generating test cases from UML activity diagrams using the Condition-Classification Tree Method. In Software Technology and Engineering (ICSTE), 2010 2nd International Conference on (Vol. 1, pp. V1-62). IEEE.

[20] 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.

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

[22] 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.

[23] 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.