Exploiting UML Diagrams for Test Case Generation: A Review

Raj Gaurang Tiwari
Chitkara University Institute of Engineering and Technology
Chitkara University
Punjab, India
rajgaurang@chitkara.edu.in

Arun Pratap Srivastava
Department of Computer Science and Engineering
G. L. Bajaj Institute of Technology & Management
Greater Noida, India
apsvgi@gmail.com

Dr Garima Bhardwaj
Associate Professor
Amity University
Greater Noida
India
gbhardwaj@gn.amity.edu

Vinay Kumar
Department of Computer Science and Engineering
AMITY University
Greater Noida, India
vkumar@gn.amity.edu

2021 2nd International Conference on Intelligent Engineering and Management (ICIEM) | 978-1-6654-1450-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICIEM51511.2021.9445383

Abstract— **Testing is an integral part of life cycle of software development/ It aims to create a highly uncompromising framework. The program tests are conducted to verify the complex actions of the programme in a small range of test cases chosen from the execution domain. The unified modelling language (UML), which attracts considerable attention from the researchers and professionals, has emerged as an important standard for modelling software systems. In this paper, we will look at how to simplify test cases using UML diagrams including actvity, state, and sequence diagram.**

Keywords— Software Testing, UML, Test Case Optimization, Activity Diagram, State Diagram, Sequence Diagrams

I. INTRODUCTION

The testing method identifies the variance between the anticipated and existent outcomes. When the programme is not working according to standards and objectives, a software error is recorded. An error is an unwanted behaviour that is observed during the system execution. To diagnose an error, testing is carried out. A triplet [I,S,O] is usually referred to as a test case, where "I" is the data input, "S" is the system's state, and "O" is the predicted outcome. The set of test cases is known as a test suit and an appropriate test procedure should be used for the optimum test suit, i.e. the highest amount of defects with minimum of test cases should be uncovered.

Testing process is no more confined to a single phase of traditional software development. Testing process is now iterative and incremental. In the context of object-oriented software development, due to incremental development, each version of the software has to be retested to make it bug free. At the same time, post delivery maintenance is also a major concern. Issues like infant mortality, modification to the existing functionality or adding new functionality requires retesting of the software to ensure the quality[1][17].

II. CONCERNS WITH OBJECT-ORIENTED SYSTEMS

The object-oriented model has several benefits such as isolation, encapsulation and maintainability to enhance product consistency and minimise the difficulty of the software development process. But, improper and excessive use of these feature gives rise to several testing problems as well as design and implementation issues. Some of the issues in testing object-oriented systems are given below[2][16]:

1. An object-oriented system is activated through message passing. So it should be ensured that all the inter-object and intra-object message dependencies are taken into account during the testing process.
2. Inheritance ensures reusability by which several repeatable tasks in object- oriented programming can be eliminated. But at the same time deep class hierarchy poses special difficulty during the testing process. This increases the cost of the testing process.
3. Polymorphism is another issue. In polymorphism due to run time binding, the invocation of a method is unknown until run time. So it becomes difficult for creation of integration test plan.
4. Abstract and generic classes provide important means for code reuse. But it is difficult to test abstract classes due to unavailability of instance of a class. Many times in inheritance, classes used in the root are declared as abstract. In such cases any fault in the abstract class can percolate down in the inheritance hierarchy and can result in multiple bugs.
5. Encapsulation results in another hitch in object-oriented testing. It doesn't permit exterior functions like debuggers to access the private attributes of a class.

III. UML MODELS

UML is a basic visual modelling language designed to define, envisage, construct and document object-based virtual systems devices. UML is a standard visual modelling language. UML has a number of schemes to display specific features of software devices[15][18]. Followings are the two categories that include UML diagrams:

- Structure Diagrams: They illustrate the static structure of the system and its representation for diverse abstraction and implementation levels and association to each other.

- Behaviour Diagrams: These diagrams represent the complex action of objects in a system, which can be visualised as a sequence of system shifts over time.

The structure diagram is again classified into ten different diagrams and behaviour diagram is classified into seven different diagrams as shown in figure 1.

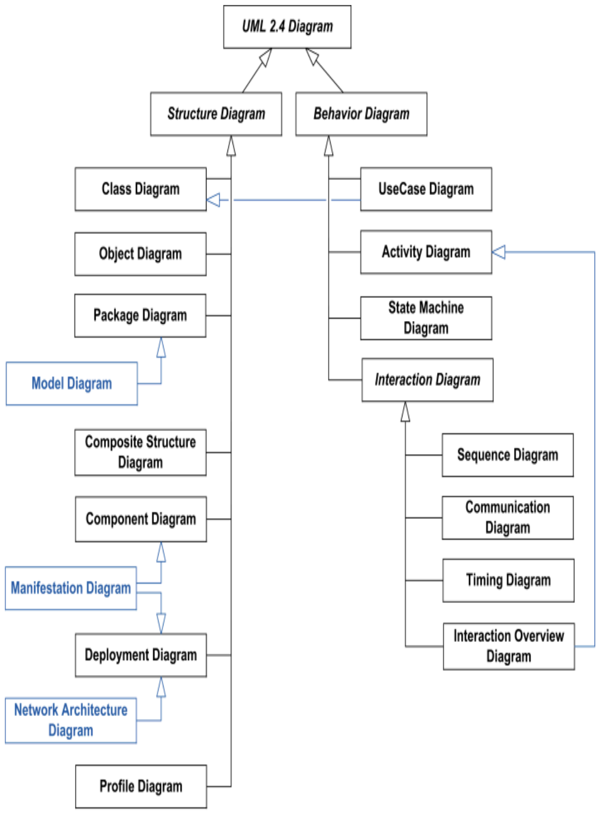

Fig. 1. Classification of UML Diagram

Source: www.UML-diagrams.org

Test case generation from UML specification is more advantageous since it allows the test plan to be carried out in parallel with the development cycle, which results reduced development cost, time and also helps in maintaining software quality and reliability. Different UML models provide diverse views of the system, and different types of faults can be detected using diverse kinds of UML models. As a result, we examine approaches to optimal test case generation using state, activity, and sequence diagrams in this article.

IV. Optimal Test Case Generation

A. Using Activity Diagram

An activity diagram is a variety or special case in a state machine in which states are operations which reflect operational output and transitions result from operations completed. In theory, it can be used to define the method of running a company The operation diagram shows flows and activities within the usage case.

- Bhattacharjee Gargi, Pati Priya. [3] used UML Activity Diagram for prioritization of test cases. They defined the most important pathways when evaluating the software as they are most likely to be carried out. They used a heuristic Meta technique called Tabu quest to stumble on the path with uppermost priority so that it can be tested foremost. The research efficacy is improved by identifying the most important path. Tabu scan can be used for unravelling problems. In the Tabu search process, a possible solution is iteratively passed to an improved neighbourhood solution before a certain stop condition has been met. Initially the authors created the activity diagram. They formulated an activity dependency table(ADT) based on the activity diagram. They created an ADT-based control flow map. The DFS algorithm was then applied to perform potential checks.

- Florestal et al.[4] suggested the EasyTest method in generating UML Activity Diagram Test Cases for the purpose of amalgamating programme modelling, code and test phases and reducing implementation costs and efforts. Even during the modelling stage, this technique recommends an early identification of defects to prevent the implantation of uncovered defects in the coding phase. Authors created an EasyTest method in order to authenticate the solution suggested for the JUnit system.

- The Automatic Test Case Generation Method was suggested by Yupei Yin et al[5], using SysML activity diagram. The SysML activity diagram as input is used and amorphous sections may be limited. Then the input activity diagram was condensed by using the transformation algorithm. It is a repeating mechanism that deals with the loop module, the simultaneous module and the setback of several beginning nodes. This amorphous action diagram was turned into intermediate blackbox model (IBM). IBM consists of a simple module and a black-box map for the initial acts. The basic module only contains black boxes and uncomplicated actions.

- Chang For applications modelled by UML diagrams, Sunet et al.[6] suggested a transformation-based methodology for creating scenario-based test cases. The suggested methodology first transforms a UML activity diagram definition into a mid-term representation by means of a range of transformation rules, from which it generates test scenarios for the specified competition coverage parameters. Finally, the solution draws up a series of test cases for the built-in test scenarios.

- By using the theory of extension The automated system for the development of UML activity diagrams was introduced by Liping Lia et al. [7]. Extension theory is a modern field for the simulation of formal processes and transition discord questions. They develop an algorithm to create the Euler circuit, and automatically produce test sequences via the Euler circuit algorithm, for the discovery of further defects in the software framework using reduced test case. The authors converted the activity diagram into the Euler circuit since the Euler path is a path that traverses each edge just once. The transition coverage of the Euler circuit can be ensured and test cases streamlined.

- Swain et al. [8] proposed a framework for combining and prioritising test cases from UML activity and communication diagrams in order to define test case cluster levels. They made an introduction of representative intermediate tree called the COMMACT tree made of communication and activity diagrams. Then the traversal of a COMMACT tree leads to extract test cases taking into account not just the basic predicates but also the prerequisites. The strategy established a priority measure for the sequence of approaches and relevant properties under safeguards. Their strategy resulted in unredundant test scenarios and prioritised test coverage.

B. *Using State Diagram*

The state machine diagram contains state, state-to-state transitions, events (triggers of a transition), and activities (the response to a transition). Diagrams of the state machine address the system's complex perspective. It demonstrates the states an entity may have in its lifespan as well as the events that cause the state to shift along with its responses. The nodes represent states and the edges represent the transitions between the states.

- Son et. al.[9] utilised state diagram to automatically produce a test case. They proposed two conceptions first mapping the state diagram onto the cause-effect graph and second applying the model transformation technique. Their suggested method consists of three steps of model transformation for test case generation as follows: 1) transforming the state diagram into the cause-effect diagram, 2) transforming the cause-effect diagram into the decision table, and 3) transforming the decision table into the test case in this order. The model transformation is required to design metamodels of target models, and define rules for model transformation. They described the metamodel of a state diagram which is not defined in the previous research, and identified the rule for transforming the state diagram into the cause-effect graph.

- The proposed approach of Preeti Gulia et al.[10] generates and refines test cases from UML State Chart through the use of genetic. The formula for crossover from the genetic algorithm was used to generate the newly-catched test series. Mutation review estimated the efficacy of the test sequences.

- P. R. Srivastava et. al.[12] suggested an algorithm using Ant Colony Optimization (ACO), which offers an automated status-transitional test series. The ant has the information to collect all possible transformations from its current condition in accordance with the algorithm suggested. A tool for testing the viability of the existing state transformations is used. The feasibility collection of transformation described this strategy. This method reduces the number of repeated transformations in the test series and therefore provides maximum coverage.

C. *Using Sequence Diagram*

Sequence Diagram is used to show the interaction of a given set of objects. A sequence diagram (SD) comprises of a sequence of messages "M" and a set of objects "O". Each message m that belongs to M depicts a sequential control flow from one object (o1) of the system to another object (o2) of the system

- D. P. Mohapatra et al.[12] intended to generate robotic UML diagram test cases and to compute the software system Hit Ratio percentage. Firstly, ArgoUML was used to construct the software's sequence diagram. This diagram has been given in order to translate the software to XML text. They subsequently delivered this XML code to the Tcases tool for the blackbox test methodology test case generation.

- Swain et al. [13] developed a technique for building test cases using UML use case as well as sequence diagram. Their proposed technique is used for system and integration testing. They have generated Use case dependency graph (UDG) using use-case diagram and using sequence diagram. They generated concurrent control flow graph (CCFG). By using these two graphs they have generated test cases. Their testing strategy was based upon predicate coverage.

- Bahare Hoseini[14] has used genetic algorithms to propose a model that automatically creates all primary paths and extracts minimum paths for all primary paths with as little as possible duration. The experimental outcomes indicate that the paths produced can easily become the best test paths with the fewest possible test paths.

V. ANALYSIS

Firstly, we analysed the methods employed with activity diagrams to refine and prioritise the test cases. Most of the studies transform activity diagrams in some intermediate state into a graph or tree to produce the test case. Various coverage criteria for creating and maximising test cases have been used in various techniques. Literature research reveals that much of the existing literature has focused heavily on basic diagrams for UML activity and some experiments have addressed dynamic situations including the adaptation of concurrency, loops, joins and forks.

In the next step, we studied the methods used for test case improvement using state diagrams. The majority of approaches exploit the conception of genetic algorithms. Most techniques use DFS to decide related predicates. These conditional predicates are transformed to produce test cases automatically. But the UML state model of a real system is often extremely multifarious and consists of a large number of states as well as transitions. Possibly for this reason, system developers seldom build state models of complete systems.

In the study of the test cases, we find that the use of the sequence diagram as an input object for producing the test path is cost and time efficient as the test loop starts before the implementation step. The participation of the tester is often limited to a minimum in the complexity of the source code. The control flow map generates the test paths, which are taken from the sequence diagrams. Among all the coverage criteria based on graphs, the main path coverage includes diverse coverage criteria based on graphs that lead to complete the path coverage. In comparison, instead of passing all existing roads, the primary pathway coverage focuses on both nodes and edges in the control flow map.

VI. Conclusion

Automating the process of test case creation will lead to considerable time and effort reduction while at the same time contributing to improved software stability through enhanced test coverage.

We discussed different approaches used to perk up the production of test cases in this paper. We addressed the work on the development of test cases for object-oriented systems with UML diagrams. The numerous testing outcomes related to the generation of test cases in several UML diagrams were also discussed. We investigated the model-based optimization of test cases with generic techniques such as taboo search, genetic algorithm, ACO-algorithm, etc.

References

[1] Orso, Alessandro, and Gregg Rothermel. "Software testing: a research travelogue (2000–2014)", Proceedings of the on Future of Software Engineering. ACM, 2014.

[2] P. Sapna and H. Mohanty, “Test generation from UML sequence diagrams”, In Proceedings of 8th International Conference on Quality of Information, and Communications Technology (QUATIC), 2012.

[3] Gargi Bhattacharjee, Priya Pati, “A Novel Approach for Test Path Generation and Prioritization of UML Activity Diagrams using Tabu Search Algorithm”, International Journal of Scientific & Engineering Research, Volume 5, Issue 2, pp1212-1217, 2014

[4] Fernando Augusto Diniz Teixeira, Glaucia Braga e Silva, “EasyTest: An approach for automatic test cases generation from UML Activity Diagrams”, 14th International Conference on Information Technology: New Generations, 2017.

[5] Yufei Yin, Yiqun Xu, Weikai Miao, Yixiang Chen, “An Automated Test Case Generation Approach based on Activity Diagrams of SysML”, International Journal of Performability Engineering, Vol . 13, No. 6, pp. 922-936, 2017

[6] Sun, Chang-ai and Zhao, Yan and Pan, Lin and He, Xiao and Towey, Dave, “A transformation-based approach to testing concurrent programs using UML activity diagrams. Software: Practice and Experience”, Vol 46, Issue 4, pp. 551-576, 2014.

[7] Liping Lia, Xingsen Li, Tao He, Jie Xiong, “Extenics-based Test Case Generation for UML Activity Diagram”, Procedia Computer Science 17, pp. 1186 – 1193, 2013.

[8] Swain, Ranjita Kumari, Vikas Panthi, Durga Prasad Mohapatra, and Prafulla Kumar Behera. "Prioritizing test scenarios from UML communication and activity diagrams", Innovations in Systems and Software Engineering 10, no. 3, pp. 165-180, 2014.

[9] Son, Hyun Seung, et al. "An automatic test case generation from state diagram using cause-effect graph based on model transformation". Journal of Engineering Technology 6.1, pp. 378-393, 2018.

[10] Gulia, Preeti, and R. S. Chillar. "A new approach to generate and optimize test cases for UML state diagram using genetic algorithm, ACM SIGSOFT Software Engineering Notes 37.3 pp. 1-5, 2012.

[11] Srivastava, Praveen Ranjan, and Km Baby. "Automated software testing using metahurestic technique based on an ant colony optimization", Electronic System Design (ISED), 2010 International Symposium on. IEEE, 2010.

[12] Mohapatra, Durga Prasad, Sangharatna Godbley, and Arpita Dutta. "Measuring Hit ratio of Software Systems using UML Sequence Diagram", 58th Annual Convention of Institution of Engineers (India), 2017.

[13] Swain, S.K., Mohapatra, D.P. and Mall, R., “Test case generation based on use case and sequence diagram”, International Journal of Software Engineering, 3(2), pp.21-52, 2010

[14] Hoseini, Bahare, and Saeed Jalili. "Automatic test path generation from sequence diagram using genetic algorithm" Telecommunications (IST), 2014 7th International Symposium on. IEEE, 2014.

[15] Datta, Parul, and Bhisham Sharma. "A survey on IoT architectures, protocols, security and smart city based applications" 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, pp. 1-5, 2017.

[16] Chhabra, Rishu, Seema Verma, and C. Rama Krishna, "A survey on driver behavior detection techniques for intelligent transportation systems," 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, IEEE, pp. 36-41, 2017.

[17] Gupta, Deepali, "The Aspects of Artificial Intelligence in Software Engineering", Journal of Computational and Theoretical Nanoscience 17.9-10, pp. 4635-4642, 2020.

[18] G. Bhardwaj, S. V. Singh and V. Kumar, "An Empirical Study of Artificial Intelligence and its Impact on Human Resource Functions," 2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM), Dubai, United Arab Emirates, 2020, pp. 47-51, doi: 10.1109/ICCAKM46823.2020.9051544.