International Journal of Latest Trends in Engineering and Technology
Vol.(16)Issue(1), pp.038-043
DOI: http://dx.doi.org/10.21172/1.162.06
e-ISSN:2278-621X

A SURVEY ON AUTOMATIC GENERATION OF TEST CASES FROM UML DIAGRAMS

Nikhil Karve[1] , Mayur Aitavdekar[1], Mahima Chandan[1], Prathamesh Wali[1]

Abstract: When building a high-quality software, the main focus is on the testing phase from the software development life cycle. By various ways we can test a system under development but many of these consume a lot of time and effort and so research on approaches that automate this process of testing and that consume fewer human efforts is focused on, in this paper. ratio (4.79) was found highest under T_2. Colour properties*i.e.* L, a and b value of colour were found highest under T_1.

Keywords: Software testing, test cases, UML diagrams, coverage criteria, software engineering.

I. INTRODUCTION

Software systems are getting more complex to build with increasing demand of automation. Systems that are developed are then tested according to the user expectations. Testing is also done on the designs made before execution so as to confirm the end product functionalities. Errors can be encountered while coding or while deployment if user satisfaction is not fulfilled. Identification of errors and error management for a good software is very important. The earlier the faults are detected the better the system turns out to be and fewer is the time wastage. Also, defective software can't be accepted as it may directly hamper variety of factors. For developing a software, automated testing is necessary in terms of software quality. Models are an important requirement for automating a system and to provide the required useful information. Use cases helps the requirements engineering process whereas test cases provide assurance about the system's quality. Generating test cases that take care of as much of the system's functionality as possible make the system robust and an error free software for deployment. In model-based testing approaches checking functionalities and features of software is done before execution, it is actually done while designing the software. While talking about the designing phase, the advantages of UML diagrams is the first thing one thinks about. UML also known as the UML diagrams are the standard way of representing any software. There are diagrams in the UML that show the flow of the system, that showcase the functionalities, that show the dependencies and the diagrams that tell everything about the system. As the UML diagram is the input to the coding phase, flaws in the design make it a very difficult task for the developer to code the software and results in a faulty software. With the increase in input parameters it becomes hard to generate test cases and so research on these methods was done, many tools were made for generating test cases which had many different approaches. Coverage criteria of various algorithms are different which leads to less accuracy in the process of testing.

II. LITERATURE SURVEY

Usually a test case is first generated and then it is selected after which minimization of test case takes place and then test cases is prioritized and finally is evaluated. The very first phase is the most important and requires most of the efforts called the Test Case Generation. This part of our paper speaks about the waterfall model of software development life cycle, the process of software testing, the process of test case generation and all research related to techniques used generating test cases.

1. Techniques used for test case generation-

[1]*Department of Computer Engineering, VIIT, Pune, Maharashtra, India*

It actually is the total count of test cases that affect the money, time invested and the human effort required. Generating Test cases being the most crucial operation in the process of testing. The bigger picture of classification of Test case generation is requirement related methods, techniques dependent on specification, techniques that are related to diagram specially unified modeling language, source code-based techniques and genetic ways. A couple of test cases are determined depending on the basis of previous considerations including fault distribution covering different coverage criteria in the Random test case generation approaches. Covering a particular section, statement or a functionality is the purpose of goal-oriented test case generation approach. Testing the goal is the primary objective here and not the execution path. The domain of software engineering uses UML as a modeling language for a general way of representation. There are two different ways of representing views of a system model: Structural and Behavioral. In structural system model objects, attributes, dependencies and operations are emphasized. The static way of representing view comprises of class diagrams and composite diagrams like structure diagram. Whereas, when the dynamic behavior is considered where collaborations among the objects and alterations in the states of the objects is considered the system model is called Behavioral view.

Basically, the most basic five parts in the waterfall software development life cycle (SDLC), which are: Requirements phase, Designing phase, Implementation phase, Testing phase, Maintenance phase. The motive of testing phase is to find errors. The intent of detecting faults includes checking that the particular software meets the requirements and that it fulfills the capability of performance. It assures the developer and user about its quality by finely seeing the way the output is derived of a software system to test it whether it worked as required and to identify if any errors or problems.

A. Specification-Based Techniques

It is one of the fundamental techniques of test case generation in which requirements gathering and specification play very important role. As we are considering software development life cycle, we must specify some formal requirements and specifications. The formal requirements are very important as they are used to generate test cases in this specification-based technique. With design and implementation of software, we can get required results for test cases. The formal requirement specification document is used as base for checking various outputs which reduces major efforts in testing. Specification and tests have very strong relationship between each other which helps us to find faults and it simplifies the testing process. It also helps to discover the problem with specifications. We can get the benefits from this technique if we do this step at early stage in software development process which will ultimately save the time and resources of the company.

B. Sketch Diagram-Based Techniques

This technique contains variety of methods of generating test cases from UML diagrams [19], [20]. We can shift the testing in the earlier part of the development and also, we can generate the test cases that are not dependent on any particular model. These are few important advantages of sketch-based diagram techniques. This technique has been widely used in traditional and web-based applications. But in recent times, the complexity of web-based applications are constantly increasing day by day which means we need more robust test case generation techniques to test such kind of complex web based systems.

C. Source Code-Based Techniques

Prof. N. k. Sharma and prof. Divya Saxena conducted survey in recent times on such kind of techniques. Control flow graph information is used in such kind of techniques for identification of paths which need to be covered during the traversal of graph and also generation of correct test cases for these identified paths. Generally, one can easily derive the control flow graph from source code.

D. Scenario Based

Scenario based test case generation, model-based test case generation and genetic based test case generation these are the main approaches of test case generation systems. Prof. Baikutha Narayan Biswal presented a paper 'A Novel Approach for Scenario-Based Test Case Generation'. For complex transactions this technique works very well especially when we need to understand how program will work when experienced use uses it. In the recent paper [9], researchers also worked on approach for activity diagram where they focused on coverage criteria. Their approach is very much useful to solve the problem of looping faults. In their second approach, the algorithm can actually detect the location of fault which can ultimately makes the task of testers easy, saves the time of human and efforts too.

E. Model Based Test Case Generation

Model based test case generation is challenging and also many researches involve achieving optimal sets of test cases. In the above stated paper [11] they proposed a model-based approach, which motivates developers to improve their design and quality and also finding mistakes in the implementations at a preliminary stage reducing development time. They also stated that it is possible to develop automatic tools using their approach. Used sequence diagrams to generate test cases. They traversed sequence diagrams and conditional predicates were selected and those conditional predicates are transformed to source code. Then, the test cases are generated from the source code-based technique. From the sequence diagram, they perform a DFS to select the associated predicates in the generated list. They have generated test predicate conditions from UML sequence diagrams, which are later used to generate test cases. Model based techniques identify respective test cases for the software with respect to the UML diagrams such as activity, state machine diagram etc. By symbolic execution, static path testing is done. Goal-oriented techniques identify test cases covering a selected goal such as a statement or branch.

F. Genetic-Based Test Case Generation

Swain, R. Panthi [18] proposed an automatic generation of test cases for state chart diagrams, which they used both model based and Genetic Algorithm in the later part. Initially the UML diagram is converted to EFSM (Extended Finite State Machine) which usually satisfies provided guard conditions. Later this extended FSM was converted to an extended control flow graph. For generating workable test cases from test sequence data, this genetic algorithm is used. Automatic test case generation using Genetic Algorithm approach mostly uses the data mining approach. Authors of paper [12] applied crossover technique on the class diagram and the depth first search (DFS) algorithm is used for traversal. When a tree structure approach is considered together with a genetic algorithm then it shows that it has the ability to show at unit level about 80% faults in which 8% more faults at integration level. Genetic algorithms were integrated with mutation testing which showed nearly 80% of effectiveness.

G. Critical path method

T.Y. Chen et al proposed a critical path method for generation of test cases. Formation of test cases takes place for the critical path with the help of refinement of functional choices which is very helpful to the software tester. The success criterion is highly dependent on the predicted and the real results for a specific test and the gap between them shows us the error which helps to find the accuracy of the software.

H. Code based Test Generation

Straightforwardly generated test cases from code will be code-based test case generation. This principally used strategy is considered for existing dependent tests on adequate test criteria. This technique bolsters testing strategies like regression to diminish the size of test suite or sort tests depending on priority.

I. Graphical User Interface based Test Case Generation Technique

Imran Ali Qureshi and Aamer Nadeem introduced completely an overview of test case generation strategies that are fundamentally helpful for Graphical User Interface based test case generation techniques.

1. The very first method of generating test cases for Graphical User Interface obligations is a method where complete interaction sequence is considered.

2. The second strategy for generation of test cases depends on Graphical User Interface is by limited state based testing and examination of graphical User Interface.

3. The third strategy for generating test cases through Graphical User Interface is by characterizing experiment for Graphical User Interface testing.

4. The fourth strategy/method of test case generation is accomplished with the assistance of a model driven methodology.

5. The fifth strategy for generating test cases is finished by achieving cost-efficient approach for based on model technique for Graphical User Interface testing.

6. Another technique for generating test cases is a model-based methodology for testing Graphical User Interface utilizing various leveled predicate change nets.

7. The next method of generating test cases is from the Graphical User Interface Model itself.

8. The last technique for generating test cases is through the use of Graphical User Interface dynamic state as criticism to generate test cases.

The paper likewise portrays all above test case generation techniques for a wide range of Graphical User Interface applications and presents a improved picture about the usefulness of every method and further presented a table for correlation wherein every method has been contrasted and different systems by utilizing some assessment parameters.

J. Genetic Algorithm

The latest research is on test case generation, reduction and assessment utilizing Genetic Algorithm. While considering test case generation from State chart Diagrams of Unified Modeling Language diagrams most extreme work is finished utilizing Genetic Algorithm. A Genetic Algorithm is system that goes under advancement method which can be applied to different issues. It utilizes natural selection strategy, where the best arrangement endures. The Genetic calculation essentially requires two of the most significant segments that are

(a) An encoding used to present an answer of the arrangement space

(b) A function that determines the fitness and goodness of possible solutions.

III. SURVEY RESULT

Author(s)	**Input model**	**Method**
Supaporn Kansomkeat and Wanchai.R	State chart model	Parsing using TFG and mutation on analysis
J. Offutt	State chart model	Specification test
S. Gnesi	State chart	Input/output systems random test selection
Lionel Briand	State chart model	Normalization operation and transition of guard conditon
Li and Lam	State chart model	Ant colony optimization
Santiago	State chart model	Condado
Murthy	State chart model	Extended state chart model
Ali	Collaboration diagrams	Collaboration test model
Santiago	State Charts	unique IO methods
Kosindrdecha and Daengdej	State chart	Sketch Diagram based
Swain	State chart and activity chart	Mutation analysis and System Design
Shirole	State chart model	Genetic algorithm

Li	State chart model	Euler circuit algorithm
Swain	State chart model	Depth first search, Model J unit
Swain	State chart model	Test generation and Minimization for O-O software with state charts
Swain	State chart model	Generation and minimization of test cases from State Charts
Chimisliu and Wotawa	State chart model	
V Panthi, Durga P Mohopatra	Sequence Diagram	DFS and Selection of Predicate from fuction
P Satish, Arinjita P, K Rangarajan	Sequence Diagram	CTDM Parsing
A.V.K. Shanthi1 and G. Mohan Kumar	Sequence Diagram	Genetic Algorithm
E Cartaxo, F Neto and P Machado	Sequence Diagram and LTS	TFG
Abinash Tripathi and Anirabn Mitra	Sequence and Activity Diagram	Dfs and System Graph
Khandai, M., Acharya, A. A., & Mohapatra, D. P	Sequence Diagram	MDG and Selection of predicates
P Samuel, Sahoo and R Mall	Sequence Diagram	MDG and Selection of predicates .
Mahesh S, Suthar. A & Kumar	Sequence Diagram	Genetic Algorithm
Fan, X., Shu, J., Liu, L., & Liang, Q.	Activity Diagram	Functional Decomposition.
Mingsong, C., Xiaokang, Q. & Xuandong, L.	Activity Diagram	Random generation of test cases using Java.
Thomas, A., & Kimball, J.	Activity Diagram	Interface behaviour descriptions and edge value analysis
Xu, D., Zhu, G., Lan, Z. & Li, J.	Activity Diagram	Model based test case generation
Oluwagbemi, O., & Asmuni, H.	Activity Diagram	Activity flow tree
Samuel, P., Mall, R., & Bothra, A.	Activity Diagram	Data flow logic, functional minimisation technique
Hettab, A., Kerkouche, E., & Chaoui, A.	Activity Diagram	

Table 1: Research work completed by researchers on different UML Diagrams and getting automatic test cases.

IV. CONCLUSION

Our paper discusses different ways or research done in the field of automatic test case generation from UML diagrams. Various algorithms and methods that are used in generating a test case for testing are Search Based Software Test case generation, Finite state Machine, Model Based Testing are read and literature survey for UML

diagrams which is necessary for virtualization of the system. Also, according to our paper the main focus is primarily on UML related testing techniques. Other diagrams could be studied in detail for future work. Coverage criteria can yet be maximized and make testing more efficient.

REFERENCES

[1] Fan, X., Shu, J., Liu, L., & Liang, Q. (2009). Test Case Generation from UML Sub activity and Activity Diagram. 2009 Second International Symposium on Electronic Commerce and Security. doi:10.1109/isecs.2009.160

[2] Mingsong, C., Xiaokang, Q., & Xuandong, L. (2006). Automatic test case generation for UML activity diagrams. Proceedings of the 2006 international workshop on Automation of software test - AST '06. doi:10.1145/1138929.1138931

[3] Thomas, A., & Kimball, J. (2017). A prototype tool for generating and executing test cases from UNIFIED MODELING LANGUAGE-based interface behavior descriptions. 2017 IEEE 28th Annual Software Technology Conference (STC). doi:10.1109/stc.2017.8234458

[4] Xu, D., Zhu, G., Lan, Z., & Li, J. (2014). A novel systematic approach to automatic test scenario generation from UML activity diagrams. Advanced Computer Control. doi:10.2495/icacc130831

[5] Oluwagbemi, O., & Asmuni, H. (2015). AUTOMATIC GENERATION OF TEST CASES FROM ACTIVITY DIAGRAMS FOR UML BASED TESTING (UBT). Jurnal Teknologi, 77(13). doi:10.11113/jt.v77.6358

[6] Samuel, P., Mall, R., & Bothra, A. (2008). Automatic test case generation using UML (UNIFIED MODELING LANGUAGE) state diagrams. IET Software, 2(2), 79. doi:10.1049/iet-sen:20060061

[7] Hettab, A., Kerkouche, E., & Chaoui, A. (2008). A Graph Transformation Approach for Automatic Test Cases Generation from UML Activity Diagrams. Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering - C3S2E '15. doi:10.1145/2790798.2790801

[8] Satish, P., Paul, A., & Rangarajan, K. (2014). Extracting the Combinatorial Test Parameters and Values from UML Sequence Diagrams. 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops. doi:10.1109/icstw.2014.11

[9] Sarma, M., Kundu, D., & Mall, R. (2007). Automatic Test Case Generation from UML Sequence Diagram. 15th International Conference on Advanced Computing and Communications (ADCOM 2007). doi:10.1109/adcom.2007.68

[10] Khandai, M., Acharya, A. A., & Mohapatra, D. P. (2011). A novel approach of test case generation for concurrent systems using UML Sequence Diagram. 2011 3rd International Conference on Electronics Computer Technology. doi:10.1109/icectech.2011.5941581

[11] Dahlweid, M., Brauer, J., & Peleska, J. (2015). Model-Based Testing: Automatic Generation of Test Cases, Test Data and Test Procedures from SysML Models. SAE Technical Paper Series. doi:10.4271/2015-01-2553

[12] Carballa, D., & Castro, L. (2016). Automatic generation of UML sequence diagrams from test counterexamples. Proceedings of the 15th International Workshop on Erlang - Erlang 2016. doi:10.1145/2975969.2975977

[13] Nayak, A., & Samanta, D. (2012). Synthesis of Test Scenarios Using UML Sequence Diagrams. ISRN Software Engineering, 2012, 1-22. doi:10.5402/2012/324054

[14] Tran, H. (2001). Test Generation using Model Checking. Paper presented at Proceeding Conference on Automated Verification.

[15] Shirole, M., Suthar, A., & Kumar, R. (2011). Generation of improved test cases from UML state diagram using genetic algorithm. Proceedings of the 4th India Software Engineering Conference on - ISEC '11. doi:10.1145/1953355.195337

[16] Chimisliu, V., & Wotawa, F. (2013). Using Dependency Relations to Improve Test Case Generation from UML Statecharts. 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops. doi:10.1109/compsacw.2013.2

[17] Gnesi, S., Latella, D., & Massink, M. (n.d.). Formal test-case generation for UML state charts. Proceedings. Ninth IEEE International Conference on Engineering of Complex Computer Systems. doi:10.1109/iceccs.2004.1310906

[18] Modeling and Verification Using UML Statecharts. (2006). doi:10.1016/b978-0-7506-7949-7.x5000-4

[19] Prasanna, M., & Chandran, K. R. (2011). Automated Test Case Generation for Object Oriented Systems Using UML Object Diagrams. High Performance Architecture and Grid Computing, 417-423. doi:10.1007/978-3-642-22577-2_56

[20] Chevalley, P., & Thevenod-Fosse, P. (n.d.). Automated generation of statistical test cases from UML state diagrams. 25th Annual International Computer Software and Applications Conference. COMPSAC 2001. doi:10.1109/cmpsac.2001.960618

[21] N. K. Sharma, Divya Saxena, 2013, Study Of Approaches For Generating Automated Test Cases By UML Diagrams, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 02, Issue 06 (June 2013)

[22] Goodenough, J. B., & Gerhart, S. L. (1975). Toward a theory of test data selection. Proceedings of the international conference on Reliable software -. doi:10.1145/800027.808473

[23] McMinn, P. (2004). Search-based software test data generation: a survey. Software Testing, Verification and Reliability, 14(2), 105-156. doi:10.1002/stvr.294

[24] Swain, R. K. (2012). Minimal Testcase Generation for Object-Oriented Software with State Charts. International Journal of Software Engineering & Applications, 3(4), 39-59. doi:10.5121/ijsea.2012.3404

[25] Korel, B., Tahat, L., & Harman, M. (2005). Test prioritization using system models. 21st IEEE International Conference on Software Maintenance (ICSM'05). doi:10.1109/icsm.2005.87

[26] Dr. V. Chandra Prakash, S. Tatale, V. Kondhalkar, L. Bewoor, "A Critical Review on Automated Test Case Generation for Conducting Combinatorial Testing Using Particle Swarm Optimization", International Journal of Engineering & Technology, June 2018