ISSN 2229-5518

A Fault Based Object Oriented Testing using UML

Ravindra Kr. Gupta, Hari Ji, Gajendar Singh Chandel

Abstract— we propose a testing technique for object-oriented programs. Based on the state and collaboration models of a system, we construct an intermediate representation, which we have named state collaboration diagram (SCOTEM). We generate test cases to achieve state-activity coverage of SCOTEM .We have empirically evaluated the effectiveness of our approach. The results show that the proposed technique could detect seeded integration testing faults which could not be detected by the related approaches.

The previous work of this topic is only show the state and activity model. But we can consider the event if any message deliver to an object that behaves according to message so we can say that event based .the programming approach with the help of UML (unified modeling language) to generate the text file for collaboration diagram and the prototype model is used for the testing of path generated by that prototype model. My testing work is based on path based, path is generated with the help of UML diagram, and it shows the message sequence number it's also provide the source to target path, object, transition state. Transition shows the message imitate from source to destination. And the message passing according to the sequence number each, sequence number identifies the separate massage.

Index Terms— UML based testing, Automatic test case generation, state diagram, collaboration diagram, Mutation testing.

1 Introduction

The object oriented paradigm provides a lot of benefits like encapsulation, abstraction, inheritance and reusability to improve the quality of software because the reusability reduces the code. The reduction of code is very important things in the programming because if we are uses the many times of the code its only show the wastage of memory and increases the code. Hence the object oriented technology or concept is very beneficial for the reeducation of code. The object oriented features are use to detect the defect in the class testing .because the different class are integrated to each other so the faults may be occur. The UML has used to notation and graphical representation for the object and message also use to capturing the Message source to destination generally UMLs model is used to design the different types of diagram before the development of any software.UML models are used are used to source information in software testing [17, 11]. Many UML design artifacts have been used in different ways to perform different kinds of testing. For instance, UML state charts have been used to perform unit testing, and interaction diagrams (collaboration and sequence diagrams) have been used to test class interactions. Modularity aims at encapsulating related functionalities in classes. However, complete system-level functionality (use case) is usually implemented through the interaction of objects. Typically, the complexity of an OO system lies in its object interactions, not within class methods which tend to be small and simple.

2 Related Works

Traditional testing strategies for procedural programs, such as data flow analysis and control flow analysis cannot be directly applied to OO programs [35]. Extensions of these techniques for OO programs have been proposed by Buy et al. [26] and Martena et al. [4]. A structural test case generation strategy by Buy et al. [26] generates test cases through symbolic execution and automates deduction for the data flow analysis of a class. Kung et al. [36] proposed an idea to extract state models from the source code, whereas others suggest test generations from pre-existing state-models [11 and 40]. In the sections below, we will discuss more specific UML-based testing techniques.

Automatic test case generation from UML diagrams has received considerable attention from researchers [22, 7, 28]. There have been attempts to generate test cases from UML activity diagrams [16, 25]. Others have worked on UML state chart diagrams [4]. UML activity diagram-based test case generation has been investigated in [25] by Lizhang et al. They have generated test cases using a gray box method. In their approach, test scenarios are directly derived from the activity diagrams modeling an operation. This method deals with the logical coverage criteria of white box method and finds all the possible paths from the design model which describes the expected behavior of an operation. Subsequently, all the information for test case generation (i.e. input/output sequence parameters, the constraint conditions and expected object method sequences) is extracted from each test scenarios. Finally, they generate the possible values of all the input/output parameters by applying category-partition method [17]. It generates test cases which can achieve the path coverage. But this method

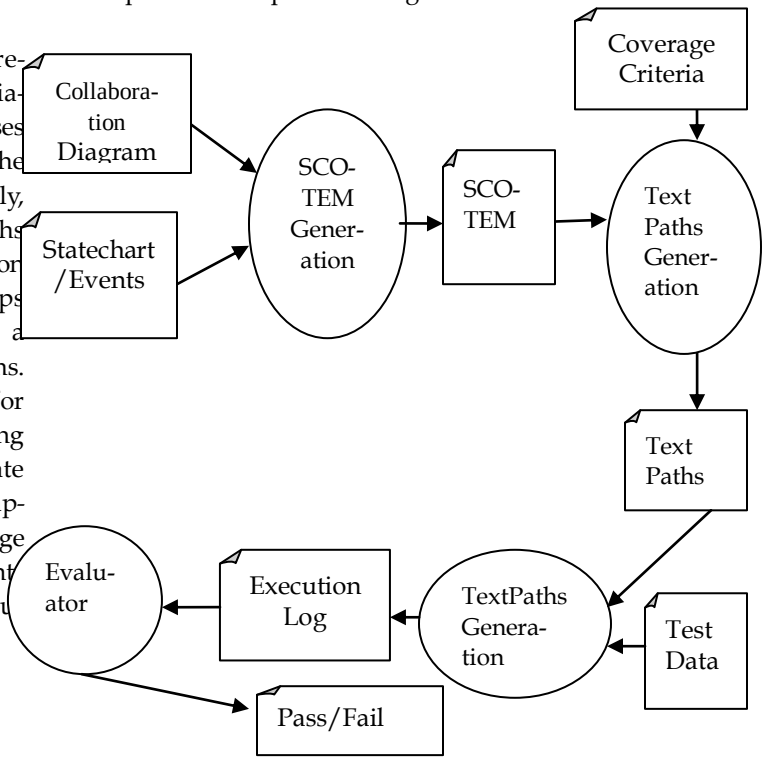
ignores information about the state of the objects within the system at any time of execution.

That is, the system takes input data, performs some computations, and outputs the result. They proposed a novel algorithm to generate thin threads from activity diagrams, which included preprocessing of the system level activity diagrams, converting them into activity hyper graphs and then deriving all execution paths from the graph. Their method does not contain any state information for the objects of the system.

Chen Mingsong et al. [16] presented an idea to obtain the reduced test suite for an implementation using activity diagrams. They considered the random generation of test cases for Java programs. Running the programs with applying the test cases, they obtained the program execution traces. Finally, a reduced test suite is obtained by comparing the simple paths with program execution traces. Simple path coverage criterion helps to avoid the path explosion due to the presence of loops and concurrency. Offutt and Abdurazik [10, 9] developed a technique for generating test cases from UML state diagrams. They generate test cases automatically from change events for Boolean class attributes. They were successful in developing several useful coverage criteria that are based on UML state charts. Their approach targets class-level testing. Their approach achieves transition coverage, full predicate coverage and transition-pair coverage. They also provide useful insight on including test prefixes that contain inputs necessary to put the software into the appropriate state for the test values.

3. Defining the SCOTEM test Model

The SCOTEM is a specific graph structure: A vertex corresponds to an instance of a class (in a particular state) articipating in the collaboration.A Modal Class can receive a message in more than one state and exhibit distinct behavior for the same message in different states. To capture this characteristic, for modal classes, the SCOTEM contains multiple vertices, where each vertex corresponds to an instance of the class in a distinct abstract state (corresponding to states defined in state charts). On the other hand, a non-modal class only requires a single vertex in the SCOTEM graph.The edges in the SCOTEM test model are of two types: message and transition edges. A message edge represents a call action between two objects, and a transition edge represents a state-transition of an objection receiving a message. Each message edge may also contain a condition or iteration. Each message may cause a state transition to occur. A transition edge connects two vertices of the same class. State charts may have multiple transitions to distinct states for the same operation. Hence, there may be multiple transition edges (representing a conditional state transition) for the same message edge in SCOTEM. Each of these transitions is generally controlled by mutually exclusive conditions (to prevent non-determinism). The internal representation of a vertex holds the class name and state of the instance it corresponds to. Message edges are modeled in the SCOTEM by attributes of a message including message sequence number, associated operation, receiver object, and the sender object. The transition edges are modeled by the attributes of a transition including sequence number, associated operation, accepting state and sending state. The proposed model graphical representation present in fig1.



Fifg1 proposed model for testing

4. Constructing the SCOTEM

It's based on the path generation the path generation is tuff task if the number of path increases. The single path calculation is easy but if the path is complicated the manual calculation is tuff but his model provides the automated path generation facility.

The example used consists of an implementation of a Question Calculation (QC). In its comprehensive form, user can login the system. The user enter the correct login and password then he get the question for solving he can solve the question and he put the answer correct the system show the result and grade. And next question show for solve user, solve correct or incorrect system show accordingly pass fail or grade.

The implementation of QC that we consider in our example is a restricted form of the assessment mode that deals with the addition operation only. Currently, the application presents questions one by one, one after the other, for user/ students. Students are given unlimited time to solve each problem, but a

counter can count the question and the condition operation can provide the condition and according to condition system show the grade and pass fail status. The system is very simple firstly the student login the system if that is correct then he will login the system. If the user id incorrect the massage you get from the system please enter the correct id. And the password is incorrect the system shows the message insert correct id. After login the system show the question and option to user input the answer from keyboard then the condition operator count the answer of the question if three questions are correct the system show the pass and grade of that user.

Display() is used for login the system if the correct user show the system login is correct.*Login'@'Unautorized.*

Login instants can be also proved I the other form of admin and user. After login the Display_quest() function can be responsible for the displaying the question. The answer if the answer is correct three or more than three the grade will display according to question.

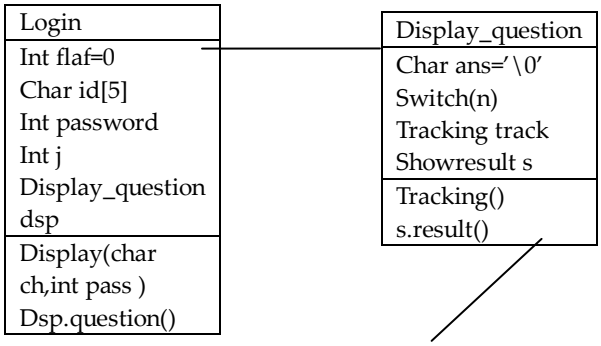
The Tracking () function can responsible for the tracking of the question to display the question accordingly the sequence number.

ShowResult() function responsible for the displaying the result overall the completion of the question and also show the grade of the user who solving the question

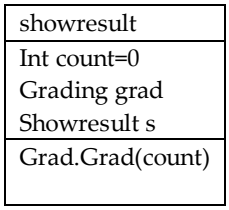
checker first of all the person or admin login the system the class login responsible for login the user or admin. After the login system display the question with the option to select the person for appropriate question. Tracking class track how many number of question solved by user .only five questions are show here user can solve the entire question or switch from one or more questions. The show result function and the grading function display the result of the user if the user can solved below the three questions. The result function can show the user is fail because the condition is applied if less than three question user failed above the three or three shows the pass. Grade function can show the grade a,b,c, according to the solved question if solved question is three the grade is c,solved question is four grade is shown b,if solved question is five grade is shown a.

Table1: Test case for QC system.

S.No.	Sequence	Result
1	Login id & password valid	Valid
2	Login id valid password invalid	Invalid
3	Login id invalid password valid	Invalid



Class diagram



5.0 Case study

A class diagram of QC system created using rational rose tool has been considered for test case automation process. This program user can initiate with the help of login in two of any one mode like user or admin mode.We have validate the proposed approach with the help of an program like question

4	Loginid=abcd, Password=12345	Valid
5	Loginid=abca, password=12345	Invalid
6	Loginid=abcb, password=12345	Invalid
7	Loginid=abca, password=12345	Invalid
8	Loginid=bcab, password=12345	Invalid
9	Loginid=cdba, password=12345	Invalid
10	Loginid=abcd, password=12341	Invalid
11	Loginid=abcd, password=12354	Invalid
12	Logindid=abcd, password=13245	Invalid
13	Loginid=abcd, password=32145	Invalid
14	Option=A,B,C,D,E,for answer	Valid
15	Option=a,b,c,d,for answer	Invalid
16	If condition <=3,for grade A	Invalid
17	If condition <=4,for grade A	Invalid
18	If condition <=5,for grade A	Invalid
19	If condition <=3,for grade B	Invalid
20	If condition <=2,for grade A	Invalid
21	If condition >=5,for grade A	Valid
22	If question solved<=1,fail	Valid
23	If question solved<=2,fail	Valid
24	If question solved >=3,Pass grade C	Valid
25	If question solved >=4,Pass grade B	Valid
26	If question solved >=5,Pass grade A	Valid
27	If question solved option is E to Z	Invalid
28	Question solved option is A,B,C,D	Valid

6. Mutation Testing

The best effectiveness of test cases can be evaluated using the fault is injected in the program. The fault injected technique is called mutation analysis. Mutants are created for the testing its only change the same type of operators or condition. Like the condition <= or >=, data change, operation change. Mutation testing is a process by which faults are injected in the system to verify the efficiency of the test case. Mutation based analysis is a fault based testing strategy that starts with a program to be tested and makes numerous small syntactic changes into the original program. In a program with injected faults is called MUTANTS. The faults are inserted and tested in the following manner .one faulty version of program is created at a time and run against all the test cases one by one until either fault is revealed or all test cases are executed. a fault is considered to be revealed, if the output of faulty version of program is different from the original program on the same input. If a test case set is capable of causing behavioral differences between original program and mutant, mutant is considered as killed by test. The product of mutation analysis is a measure called mutation score, which indicates the percentage of mutants killed by a test set. Mutants are obtained by applying mutation operators that introduce the simple changes to original program (or specification). The faults are kept in separate version of the program to avoid interactions between such as masking.

6.1 Fault Inject

The test cases divided in different part .for the question checker process the following parameter is listed in Table 2 were considered for mutation analysis process. Our test case program the testing is based on the mutation. The mutation testing first of faults inject in the program. The mutants are the similar values injected in the program which we are called seeds in the program. For the QC class diagram we consider 50 mutants that use the mutation operator as show in Table 2. The summary of the mutants are show in Table3.

Table 2: operator and description

S.No.	Operator	Description
1	Function	Replace the name of the function
2	Loop	Changes the value of loop
3	Condition	Change the condition
4	Arguments	Change the function arguments
5	Data value	Replace the name of Data
6	Relation operator	Replace the relational operator
7	Missing statement	Missing the statement

Table3: Summary of mutants for question checker system.

Operator	Faults Inject	Faults Found
Function	4	4
Loop	4	3
Condition	5	3
Arguments	5	5
Data value	24	20
Relation operator	3	2

Missing statement	5	4

Function:-We can change the name of the function no prototype tool provides the facility to trace the function.
Loop:-We can change the value of loop, so loop cannot execute all values.
Condition: - Condition can change some value can get some value and some cannot get.
Arguments: - We can change the arguments of the function.
Data values: - Data values can change to create mutants.
Relation operator:-This operator removes the relation condition of message.
Missing statement: - This operator responsible for missing the values.

6.2 Mutation Score

The product of mutation analysis is a measure called Mutation Score, which indicates the percentage of mutants killed by a test set. Mutation score, which indicates the percentage of mutants killed by a test set .Mutation score, is founded by comparing the faults injected to faults found.

Score= (∑fault found/∑fault injected)

In the QC system application we inject 50 faults and 40 were revealed from the test cases generated. Using the above formula we get 80.0% score for QC collaboration diagram which shows efficiency level of our approach. It is diagrammatically represented in the form of bar chart as shown in figure. 8

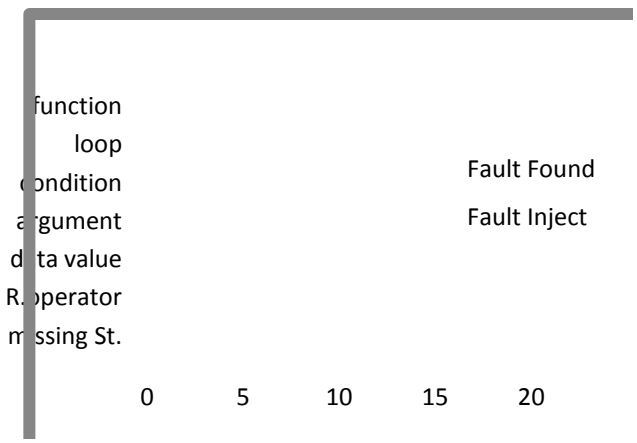


Fig 8.Mutation Operators

We also performed unit level testing and integration level testing and whose results is summarized in Table 4.

Faults	Number of Faults Inserted	Faults Found by M. Prasana Approach[10]	Faults Found by our approach
Unit Faults	30	23(76.6%)	24(80%)
Integration Faults	18	15(83%)	16(88%)

Unit fault by previous approach (%) = 23x100/30=76.6
Unit fault by our approach (%) =24x100/30=80
Integration fault by previous approach (%) =15x100/18=83
Integration fault by our approach (%) =16x100/18=88

Conclusion

Our work is a model based approach is dealing with the object behavior. We have presented a technique to generate test cases automatically from state diagram of a particular use case and statechart diagram of participating object in a use case. Our experimental results shown that it has the capability to revel 80% fault in the unit level and 88% fault in the integration level. So we can say the integration level testing is more powerful than unit level testing. Our approach is meant for cluster level testing where object interactions are tested by considering state-transitions of objects and the corresponding activities taking place in a use case. Our algorithm generates test conditions, scenarios and object-method sequences from SCOTEM using state-activity coverage. Our approach is used to exercise activity synchronization in the context of multiple state combinations in order to detect synchronization of state as well as activity faults within a use case of the system. We have implemented a prototype tool based on our approach and have used it satisfactory on QC example problems.
In the present work, we have assumed that the test data for each test case would be selected manually by the tester. Selecting test data for a large number of test cases would be tedious and time consuming. So we want to take up automatic generation of test data from test specifications as a future work. We are also now investigating how other UML models can be used to achieve higher test coverage. The same method can be uses for the use case diagram and multipath approach.

Future work

In my approach I am discussing that the SCOTEM model is based on the state diagram and the collaboration for the class integration testing on the base of graph. We can detect the state faults during the integration.the proposed algorithm can be applied for other UML diagram like Use-

case,Sequence,Activity diagram for generating test cases as further research in this direction.

References

[1] [1] (i) R.V. Binder, Testing Object-Oriented Systems - Models, Patterns, and Tools, Addison-Wesley, 2000.

[2] (ii) Caleton Technical ReportSCE-05-02.

[3] [2] B. Beizer, Software Testing Techniques, 2nd Edition, Van Nastrand Ranhald, 1990.

[4] [3] B. Bruegge, A.H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, Prentice Hall, Second Edition, 2003.

[5] [4] G.J. Myers, The Art of Software Testing, John Wiley and Sons, 1979.

[6] [5] A. Baldini, A. Benso, P. Prinetto, System-level Functional Testing from UML Specifications in End-of-production Industrial Environments, International Journal of Software Tools Technology and Transfer, Vol. 7 (4), 2004, pp. 326- 340.

[7] [6] L. Briand, Y. Labiche, Y. Wang, An Investigation of Graph-Based Class Integration Test Order Strategies, IEEE Transactions on Software Engineering, Vol. 29 (6), 2003, pp. 594 - 607.

[8] [7] L.C. Briand, M. Di Penta, Y. Labiche, Assessing and Improving State-Based Class Testing: A Series of Experiments, IEEE Transactions on Software Engineering, Vol. 30 (11), 2004, pp. 770-793.

[9] [8] A. Cavalli, S.M. Maag, S. Papagiannaki, G. Verigakis, From UML Models to Automatic Generated Tests for the dotLRN e-learning Platform, Electronic Notes in Theoretical Computer Science, Vol. 116 (1), 2004, pp. 133-144.

[10] [9] M.E. Delamaro, J.C. Maldonado, A.P. Mathur, Interface Mutation: An Approach for Integration Testing, IEEE Transactions on Software Engineering, Vol. 27 (3), March 2001, pp. 228-247.

[11] [10] L. Gallagher, A.J. Offutt, A. Cincotta, Integration Testing of Object-oriented Components using Finite State Machines, Journal of Software Testing, Verification, and Reliability, January 2006.

[12] [11] Y.G. Kim, H.S. Hong, D.H Bae, S.D. Cha, Test cases generation from UML State Diagrams, IEEE Software, Vol. 146(4), 1999, pp.187-192.

[13] [12] Y. Le Traon, T. Jeron, J.M. Jezequel, P. Morel, Efficient Object-oriented Integration and Regression Testing, IEEE Transactions on Reliability, Vol. 49 (1), March 2000, pp. 12-25.

[14] [13] Y. Ma, J. Offutt, Y. Kwon, MuJava: An Automated Class Mutation System, Journal of Software Testing, Verification and Reliability, Vol. 15 (2), June 2005, pp. 97-133.

[15] [14] J. Offutt, S. Liu, A. Abdurazik, P. Ammann, Generating Test Data from State-based Specifications, Journal of Software Testing, Verification and Reliability, Vol. 13 (1) , 2003, pp. 25-53.

[16] [15] A.S.M. Sajeev, B. Wibowo, UML Modeling for Regression Testing of Component Based Systems, Electronic Notes Theoretical Computer Science, Vol. 82(6), 2003, pp.1-9.

[17] [16] S.R.S. de Souza, S.C.P.F. Fabbri, W.L. de Souza, J.C. Maldonado, Mutation Testing Applied to Estelle Specifications, Software Quality Journal, Vol. 8 (4), 1999, pp. 285-301.

[18] [17] A. Abdurazik, J. Offutt, Using UML Collaboration Diagrams for Static Checking and Test Generation, Proceedings of the Third International Conference on the Unified Modeling Language (UML '00), York, UK, October 2000, pp. 383-395.

[19] [18] P. Ammann, J. Offutt, and H. Huang, Coverage Criteria for Logical Expressions, Proceedings of the 14th International Symposium on Software Reliability Engineering, (ISSRE'03), 2003, pp. 99-107.

[20] [19] J.H. Andrews, L.C. Briand and Y. Labiche, Is Mutation an Appropriate Tool for Testing Experiments?, Proceedings of the IEEE 27th International Conference on Software Engineering (ICSE) 2005, St. Louis, Missouri, USA, May 2005, pp. 15-21.

[21] [20] F. Basanieri, A. Bertolino, A Practical Approach to UML-Based Derivation of Integration Tests, Proceedings of the Software Quality Week QWE2000, 2000.

[22] [21] F. Basanieri, A. Bertolino, E. Marchetti, COWTest: Cost Weighted Test Strategy, Proceedings of the ESCOMSCOPE, London, England, 2001, pp. 387-396.

[23] [22] A. Bertolino, E. Marchetti, Introducing a Reasonably Complete and Coherent Approach for Model-based Testing, Proceedings of the International Workshop on Testing and Analysis of Component-based Systems (TACoS 2004), Electronic Notes in Theoretical Computer Science, 2004, pp. 85-97.

[24] [23] L. Briand, Y. Labiche, A UML-Based Approach to System Testing, Proceedings of the Fourth International Conference on the Unified Modeling Language (UML'01), 2001, pp. 194-208.

[25] [24] L. Briand, Y. Labiche, G. Soccar, Automating Impact Analysis and Regression Test Selection Based on UML Designs, Proceedings of the International Conference on Software Maintenance (ICSM'02), Canada, 2002, pp. 252- 261.

[26] [25] L.C. Briand, J. Cui, Y. Labiche, Towards Automated Support for Deriving Test Data from UML Statecharts, Proceedings of the ACM/IEEE International Unified Modeling Language conference (UML 2003), July 2003, pp. 249- 264.

[27] [26] U. Buy, A. Orso, M. Pezze, Automated Testing of Classes, Proceedings of the International Symposium on Software Testing and Analysis, ACM Press, 2000, pp. 39–48.

[28] [27] P. Chevalley, P. Thevenod-Fosse, Automated Generation of Statistical Test Cases from UML State Diagrams, Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01), 2001,pp. 205-214.

[29] [28] F. Fraikin, T. Leonhardt, SeDiTeC - Testing Based on Sequence Diagrams, Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02), 2002, pp. 261-266.

[30] [29] P. Frohlich, J. Link, Automated Test Case Generation from Dynamic Models, Proceedings of the 14th European Conference on Object-Oriented Programming, 2000, pp: 472 – 492.

[31] [30] J. Hartmann, C. Imoberdorf, M. Meisinger, UML-Based Integration Testing, Proceedings of the International Symposium on Software Testing and Analysis (ISTA'00), 2000, pp. 60 - 70.

[32] [31] A. Hartman, K. Nagin, AGEDIS Tools for Model-Based Testing, Proceedings of the International Symposium on Software Testing and Analysis (ISTA'04), pp. 129-132.

[33] [32] S. Gnesi, D. Latella, M. Massink, Formal Test-Case Generation for UML Statecharts, Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer Systems (ICECCS'04), 2004, pp. 75- 84.

[34] [33] S. Kansomkeat, W. Rivepiboon, Automated-Generating Test Case Using Statechart Diagrams Test Case Using UML Statechart Diagrams, Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology, 2003, pp. 296-300.

[35] [34] S. Kim, L. Wildman, R. Duke, A UML Approach to the Generation of Test Sequences for Java-based Concurrent Systems, Proceed-

ings of the 2005 Australian Software Engineering Conference (ASWEC'05), 2005, pp. 100-109.

[36] [35] D.C. Kung, N. Suchak, P. Hsia, Y. Toyoshima, C. Chen, On Object State Testing, Proceedings of the 17th Annual International Computer Software and Applications Conference (COMPSAC'94), 1994.

[37] [36] D.C. Kung, P. Hsia, Y. Toyoshima, C. Chen, J. Gao, Object-Oriented Software Testing- Some Research and Development, Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE'98), November 1998, pp. 158 - 165.