UTM Computing Proceedings
Innovation in Computing Technology and Applications
Volume: 2| Year: 2017 |ISBN: 978-967-0194-95-0

Exhaustive Search for Test Case Generation from UML Sequence Diagram and Statechart Diagram

Nurelisa Binti Mohd Efendi [1], Hishammuddin Asmuni [2]

Department of Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia,
81310 Johor Bahru, Johor, Malaysia

[1]elnurelisa@gmail.com, [2]hishamudin@utm.my

Abstract: *Software testing is a vital phase in the life cycle of software development. This is because it can help the software tester to minimize the cost of software development. There are three types of approaches to perform software testing, which are code-based testing, specification-based testing and model-based testing. The model-based testing using Unified Modelling Language (UML) diagrams can act as a forecast of the expected output of a system under development with the generation of test cases. However, the problem incurred when using model-based testing to generate test cases is the lack of approaches in the automatic generation of test cases. To overcome these problems, method to generate the test cases automatically is proposed. This paper focuses on test case generation by using UML Statechart diagram and Sequence diagram. The generation of test cases automatically using the proposed approach. Lastly, the analysis of the results show that the proposed approach able to help in test case generation. The automated approach developed has been proven generate the test cases with the required testing coverage, which are 100% path coverage for sequence diagram and 100% state coverage for statechart diagram.*

Keywords: Software testing, model-based testing, statechart diagram, sequence diagram, test case generation.

1.0 Introduction

Software industry always focuses on producing high quality of software product for the client. In software development, the most important phase to take part is software testing (Bhateja, 2015). It is capable of ensuring the software quality product (Zeng et al., 2009). However, software testing phase always is the less important phase by software developers because there only less time to deliver the complete software project to the client (Chartchai et al., 2007). Most software organizations always set the goal to complete the software product on time as specified by user or client. Therefore, the software testing phase always misses performing when the focus of software organizations is delivering high-quality product according to user or client budget. This phase becomes important when software organizations received a large scale of a software project (Shanmuga and Sheba, 2008). To reduce the time and cost in software development, there is a lot of approaches proposed by the researchers such as an automated software testing tool that can help software developers to generate the test cases from software design. The automated approach of the test case generation help to make software testing phase more efficient (Sabharwal and Sharma, 2011).

2.0 Related Work

This section explained the various papers related to the techniques used for generating test cases using Unified Modelling Language (UML) Sequence diagram and Statechart diagram. Firstly, the UML sequence diagram is explained. Secondly, the UML statechart diagram and the local exhaustive search are discussed.

Statechart diagrams provide the behaviour of a system under test and can be used to achieve the state coverage (Swain *et al*., 2010). The latter tool is used to generate the test cases from UML statechart diagram and can be able to interface the components based on COM/DCOM and CORBA middleware and use the simple example which is TnT environment (Hartmann *et al*., 2000). Next, the combination of UML statechart diagrams and genetic algorithm are used to generate the test cases by transforming EFSMs into control flow graph and apply a genetic algorithm to traverse the graph (Shirole *et al*., 2011).

Sequence diagram is able to shows the scenario of the sequence of the system executed to cover the path coverage criteria (Samuel and Joseph, 2008). The approach used to generate the test cases is transforming the UML sequence diagram into

sequence diagram graph or called as SDG then the graph is traverse and the graph is traversed by using breadth-first approach (Sarma *et al*., 2007).

A local exhaustive search is one of the dynamic technique use to generate test cases (Parnami *et al.,* 2012).

3.0 Proposed Test Case Generation Tool

The proposed technique used in this research was developed to validate the generation of the test cases using the automated tool named UML Sequence and Statechart Test Case Generation (USSTCG). The proposed technique is depicted in Figure 1 and its explanation for each phase is described in this section.

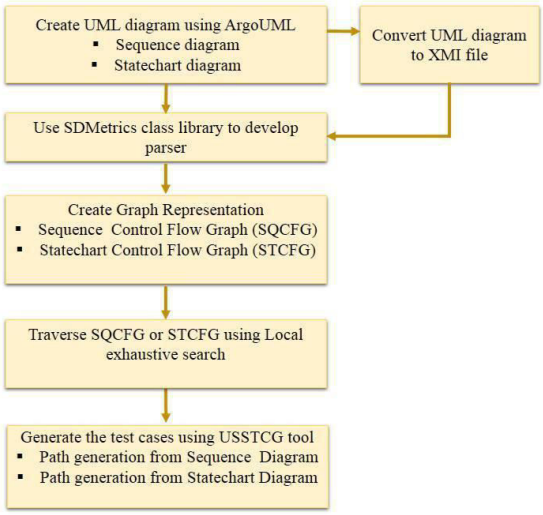

Figure 1. Proposed Test Case Generation Technique

First phase is create an UML diagram for both diagram using ArgoUML software model, which is an open source tool for create the UML Sequence diagram and Statechart diagram. Then, each UML diagram are export into an XMI file format. For each elements of type in an XMI file are identified and extracted using parser.

The parser in this paper is developed using SDMetrics class library. All implementation of tool in this paper are using Java programming language. The parser process to read an XMI file using two main type of class library in SDMetrics which are metamodel.xml and xmiTrans1_0.xml.

The graph representation for both diagram is developed using JGraphX class library. Each element in an XMI file is mapped to create the Sequence Control Flow Graph (SQCFG) and Statechart Control Flow Graph (STCFG). For SQCFG the node mapped with element of types classifier role in an XMI file and the edges mapped with elements of types message in an XMI file. For STCFG the node mapped with element of types state in an XMI file and the edges mapped with elements of types transition in an XMI file.

Each SQCFG or STCFG are traversed using local exhaustive search to generate the test cases. USSTCG tool able to traverse the path in SQFG by started with initialization of the first message from the object or classifier role. Then the generation of neighbour is selected according to the flow of message from the current message as stated in initialization and keep the potential path to be test whether it is accepted. The path generation is acceptable when it has meet the termination criteria. For this graph the local exhaustive search is stopped when reached at the end of the sequence of message. A slight different when applied local exhaustive search for STCFG. The initialization is the state named as **"**initial**"** to be the current solution. Then, the neighbour for the state **"**initial**"** is selected to compute as the potential path generation. The acceptance test is to test whether the move from the state **"**initial**"** to the neighbour of previous state is accepted or not. This approach search exhaustively all neighbour from the

current state until the state named as **"**final**"**. The termination test phase to test whether the search should terminate. The termination criteria for path generation from the statechart control graph is based on the set of limit number of path generation.

4.0 Experimental Result

In order to evaluate the capabilities of the proposed tool, two sets of experiments were conducted on simple type of case study. In summary, this paper shows the bank ATM case study for sequence diagram and Student Information System case study for statechart diagram.

The USSTCG tool is evaluated for the Sequence diagram in Figure 2 of bank ATM created using ArgoUML. Java code is being developed to parse the XMI file of sequence diagram and the graph representation of sequence diagram as shown in Figure 3. The path generated is shown in Figure 4.

The USSTCG tool is evaluated for the Statechart diagram in Figure 5 of Student Information System created using ArgoUML. Java code is being developed to parse the XMI file of statechart diagram and the statcehart control flow graph as depicted in Figure 6. The path generated is shown in Figure 7.

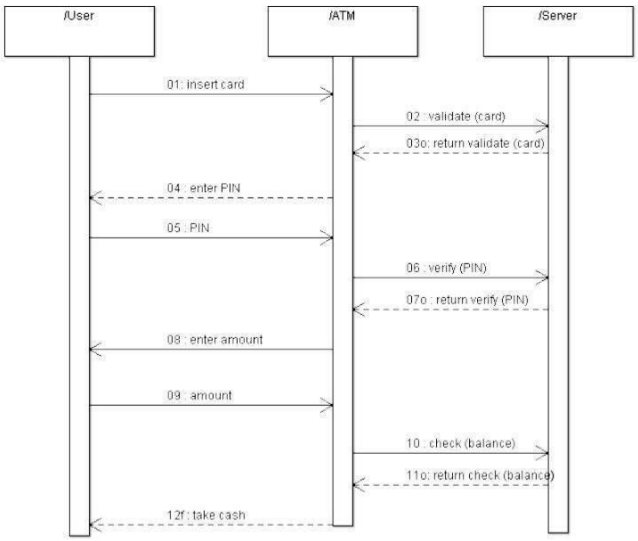

Figure 2. UML Sequence diagram of bank ATM

Path generation

1 - 01: insert card -> 02 : validate (card) -> 03o: return validate (card) -> 04 : enter PIN -> 05 : PIN -> 06 : verify (PIN) -> 07o : return verify (PIN)(Check Failed / Return False)

2 - 01: insert card -> 02 : validate (card) -> 03o: return validate (card)(Check Failed / Return False)

3 - 01: insert card -> 02 : validate (card) -> 03o: return validate (card) -> 04 : enter PIN -> 05 : PIN -> 06 : verify (PIN) -> 07o : return verify (PIN) -> 08 : enter amount -> 09 : amount -> 10 : check (balance) -> 11o: return check (balance) -> 12f : take cash(End of Sequence)

4 - 01: insert card -> 02 : validate (card) -> 03o: return validate (card) -> 04 : enter PIN -> 05 : PIN -> 06 : verify (PIN) -> 07o : return verify (PIN) -> 08 : enter amount -> 09 : amount -> 10 : check (balance) -> 11o: return check (balance)(Check Failed / Return False)

Figure 4. Path generation from SQCFG

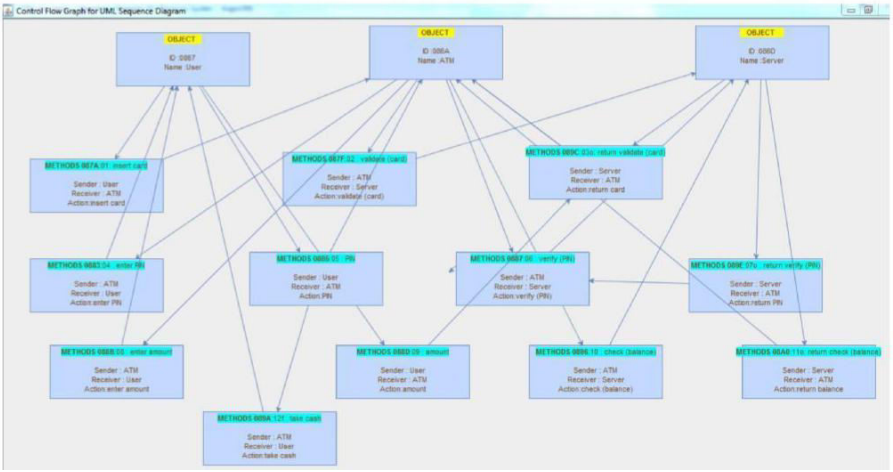

Figure 3. Graph representation of Sequence diagram

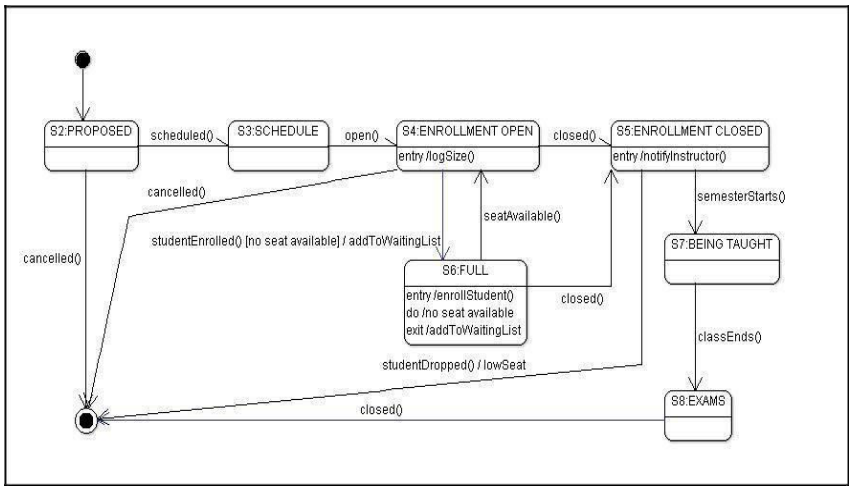

Figure 5. UML Statechart diagram of Student Information System

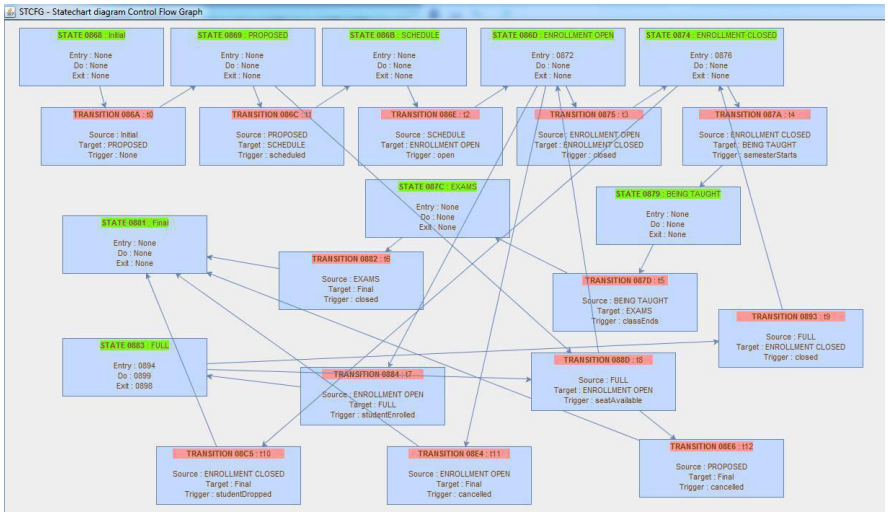

Figure 6. Graph representation of Statechart diagram

Path generation

1 : Initial ->S2:PROPOSED->Final

2 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S6:FULL->S5:ENROLLMENT CLOSED->Final

3 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->Final

4 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S6:FULL->S4:ENROLLMENT OPEN->S5:ENROLLMENT CLOSED->S7:BEING TAUGHT->S8:EXAMS->Final

5 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S5:ENROLLMENT CLOSED->Final

6 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S6:FULL->S4:ENROLLMENT OPEN->S5:ENROLLMENT CLOSED->Final

7 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S6:FULL->S4:ENROLLMENT OPEN->Final

8 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S6:FULL->S4:ENROLLMENT OPEN->S6:FULL->S4:ENROLLMENT OPEN->S5:ENROLLMENT CLOSED->S7:BEING TAUGHT->S8:EXAMS->Final

9 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S5:ENROLLMENT CLOSED->S7:BEING TAUGHT->S8:EXAMS->Final 10 : Initial ->S2:PROPOSED->S3:SCHEDULE->S4:ENROLLMENT OPEN->S6:FULL->S4:ENROLLMENT OPEN->S5:ENROLLMENT CLOSED->S7:BEING TAUGHT->S8:EXAMS->Final

Figure 7. Path generation from STCFG

5.0 Discussion

Based on the result from previous section, the analysis of test case generation from both UML diagram are evaluated. More precisely the generated test cases were analysed in terms of number of possible test cases generated and type of testing coverage. As can be seen in Table 1, all statechart and sequence diagram across the two selected of case studies was utilized for its generation of test cases and type of testing coverage.

Table 1. Analysis for the number of test cases and type of testing coverage

Type of UML diagram	**Student Information System**		Type of UML diagram	**Bank ATM**	
	Number of test case generation	Type of testing coverage		Number of test case generation	Type of testing coverage
Statechart	9	State coverage	Sequence	4	Path coverage

6.0 Conslusion

The results of the proposed approach showed that the automated approach for test case generation able to identify the path in UML statechart diagram and UML sequence diagram. This paper utilized the ArgoUML tool due to the fact that, it is open source and the model was converted to XMI format. After exporting the UML model file format, the STCFG and SQCFG tool converts the artefacts into a graph representation by identifying the information with specific elements as nodes and message or transitions as edges. The USSTCG tool is used to get the path generation from both UML diagram.

References

Bhateja, N. (2015). A Study on Various Software Automation Testing Tools. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5(6), 1250–1252.

Chartchai Doungsa-ard, Keshav Dahal, Alamgir Hossain, and T. S. (2007). An automatic test data generation from UML state diagram using genetic algorithm. *Second International Conference on Software Engineering Advances (ICSEA 2007)*, 47–52.

Hartmann, J., Imoberdorf, C., & Meisinger, M. (2000). UML-Based Integration Testing. *Paper Presented at the ACM SIGSOFT Software Engineering Notes*, 60–70.

Hoseini, B., & Jalili, S. (2014). Automatic Test Path Generation from Sequence Diagram Using Genetic Algorithm. *7th International Symposium on Telecommunications (IST'2014)*, 106–111.

Keyvanpour, M. R., Homayouni, H., & Shirazee, H. (2012). Automatic software test case generation: An analytical classification framework. *International Journal of Software Engineering and Its Applications*, 6(4), 1–16.

Khan, M. E., & Khan, F. (2014). Importance of Software Testing in Software Development Life Cycle. *International Journal of Computer Science*, 11(2), 120–123.

Parnami, S., Sharma, K. S., & Chande, S. V. (2012). A Survey on Generation of Test Cases and Test Data using Artificial Intelligence Techniques. *Proceedings of the International Conference on Advances in Computer Science and Electronics Engineering*.

Sabharwal, S., Sibal, R., & Sharma, C. (2011). Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams. *International Journal of Computer Science Issues*, 8(2), 433–444.

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)*, 60–65.

Samuel, P., & Joseph, A. T. (2008). Test Sequence Generation from UML Sequence Diagrams. *2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*.

Shanmuga Priya, S., & Sheba, P. D. (2008). Test Case Generation from UML Models – A Survey. *International Journal of Emerging Technology and Advanced Engineering Certified Journal*, 3(1), 449–459.

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*, 125–134.

Sumalatha, V. M., & G.S.V.P.Raju. (2012). An Model Based Test Case Generation Technique Using Genetic Algorithms. *The International Journal of Computer Science & Applications (TIJCSA)*, 1(9), 46–57.

Swain, S. K., Mohapatra, D. P., & Mall, R. (2010). Test Case Generation Based on State and Activity Models. *Journal of Object Technology*, 9(5), 1–27.

Zeng, F., Li, L., Li, J., & Wang, X. (2009). Vulnerability testing of software using extended EAI model. 2009 WRI World Congress on Software Engineering, WCSE 2009, 4, 261–265.