Test Case Generation Model for UML Diagrams

Yasir Dawood Salman, Nor Laily Hashim
College of Arts and Sciences, Universiti Utara Malaysia, Sintok, Kedah, 06010, Malaysia.
laily@uum.edu.my

***Abstract*—The complexity and size of software have been sequentially increasing, and the scope of testing is expanding. To insure deadline delivery and decrease development test cost, the efficiency of software testing needs to be improved. Several approaches for automated test case generation have been proposed over the last few years. However, models for automated test case generation for unified modeling language (UML) diagrams are still in the early stage of development. UML is the most widely used language to describe software analysis and design. Given that test cases can be efficiently derived from UML models, the generation of test cases from UML diagrams has attracted increasing research attention. However, no model currently exists for mapping the generation procedure. This paper proposes a model for automatic test case generation from UML diagrams.**

***Index Terms*—Software Testing; Unified Modeling Language; Test Cases.**

I. INTRODUCTION

Currently, high-quality systems and applications with minimum faults and errors are in high demand. Furthermore, decreasing time and expenses is a major concern [1]. Thus, testing techniques from requirement specifications and design documents need to be structured and automated. Automated testing techniques accelerate the delivery of services or products [2]. If the objective is to decrease expenses and improve existing technology, the use of testing automation is mandatory.

Despite the importance of testing, this process is still considered costly and time consuming [3]. Software testing cost is often calculated as more than 50% of total development cost [4]. One of the software testing methods is test case generation, which can define a test case as a classification of variables or conditions that fulfill specific test coverage criteria. By executing test cases, software testers can discover whether a software system is executed according to the system requirements or according to the sequence of its executions [5].

The unified modeling language (UML) is a widely accepted standard for modeling software systems [6]. It consists of a set of modeling concepts to support an object-oriented approach to software development. UML consists of a set of diagrams that model both the static and dynamic behavior of a system. Various aspects of the system are elaborated at different levels of abstraction by using diagrams such as use case diagram, class diagram, activity diagram, sequence diagram, and state diagram [7].

Test cases help the user to set the entire coverage to the application and test all possible combinations in the application. These cases also provide the user with easily specified steps that were processed to uncover a defect detected during a test. Test coverage also provides the testing progress and the areas in which the application has no errors [8]. Furthermore, test cases can be generated from the system requirement specification and design document [9]. Therefore, test cases can be generated in the early stage of development. The input values inside the test case reflect the system specifications and design; thus, the test case will be used to test the selected operations in detail. Improving the effectiveness and reducing the cost of software testing is beneficial, and these processes can be achieved by automating the test case generation [10] from requirement specifications and design documents.

For software development, the automatic generation of test cases will improve the efficiency and effectiveness of software testing [11]. Enhancing the necessary tools and increasing the automation of software testing will help decrease the expenses of software development and improve software reliability [10], thus decreasing the negative economic issue of defective software. Although many parts of the testing process can be automated, only professional software developers can currently write test cases [12].

However, few studies on automated test case generation revealed the proposed algorithms or testing processes used while conducting the testing, e.g., Refs. [13-17]. This scenario will lead to difficulties in updating or improving their work. Furthermore, the findings in these studies will be difficult to implement in a large scale or will be difficult to use in generating test cases in a fully automated manner.

Given the abovementioned problem, the processes involved in automating test case generation should be identified. From these processes, important components in the test case generation model can be proposed. Therefore, the current study will focus on the problem by proposing a test case generation model by using a state chart as inputs.

The rest of the paper is organized as follows. Section 2 presents the related works in the area of test case generation. Section 3 illustrates the test case generation processes. Section 4 shows the proposed test case generation model. Section 5 provides the conclusions and the discussions on future research directions.

II. RELATED WORKS

Many studies on test case generation have been proposed and implemented [18]. Some of the few important works is cited in this section. Researchers have also proposed many approaches with different methods and algorithms. Nevertheless, these methods share similarities in the use of intermediate models, path extraction, and coverage criteria.

UML is a widely accepted standard for modeling object-oriented software systems. As a semi-formal language, UML is widely used to specify requirements and depict software design. UML provides diagrams to represent the static and dynamic behavior of a system. Class, component, and deployment diagrams are used to represent the static behavior

of a system, whereas activity, sequence, and state diagrams are used to represent the dynamic behavior [7].

Santiago, et al. [16] proposed a method to automate the test case generation from UML state chart diagrams by using a software specification model. This method converts UML state chart diagrams model into an XML-based language table. Furthermore, they generated an intermediate model as a finite-state machine (FSM) based on control flow by using the perform charts tool. They aimed to determine the possibility of representing complex software with clarity and rich detail by using a high-level technique, such as UML state chart diagrams. UML state chart diagrams can be used to model a complex system realistically and provide hierarchy and parallelism to this model. Although these conditions are insufficient for guaranteeing the success of a test case generation approach, these conditions still show an improvement compared with the FSM specification use of Condado as an unconnected tool. Furthermore, Condado implements the switch cover method for the control part. A switch is a transition-to-transition pair, and their method generates test cases to cover all pairs of transitions in the model of the coverage criteria.

Boghdady, et al. [19] proposed a new enhanced methodology to generate test cases automatically from UML activity diagrams by using XML. The XML for each UML activity diagram in any system is transferred to a table called an activity dependency table, which covers all functionalities in the UML activity diagram in a reduced form. A directed graph called activity dependency graph will be automatically generated by using the activity dependency table. This graph is then used in combination with the table to generate all possible test case paths. To achieve minimization, they reduced the test case paths before generating the final efficient set of test cases. To accomplish their validation, they implemented the cyclomatic complexity technique to the generated test case paths and calculated the lower bound for the generated test case paths. Hence, the general performance of the testing process was optimized in terms of time and convenience.

Santiago, et al. [17] presented an environment-name-automated generated test case based on state chart (GTSC), which allows a test designer to generate test cases on the basis of state chart test criteria and FSM methods. This interesting characteristic allows test sequences to be generated from both the state chart and FSM. However, other comparisons need to be made on the all-paths-k-C0-configuration of the state chart coverage criteria family (SCCF), the round-trip route testing offered by Binder [20], and all-paths-k-configurations. Similarly, comparisons can be made between the latest FSM-based methods, such as state counting, and some SCCF criteria. Such an analysis will be enabled with the help of mutation testing by GTSC in applying these test criteria methods.

Shirole, et al. [21] worked on the automatic generation of test cases by using UML state chart diagrams. They used the genetic algorithm (GA) as a medium for their tool by combining information from state chart diagrams. They proposed a search-based approach to handle infeasible paths and test data generation. They used the following steps to generate the test cases. First, they transformed the UML specifications into extend FSM (EFSM). Second, they transformed the EFSM into an extended control flow graph. Third, they generated test sequences by using GA and depth-first search (DFS). Finally, they selected the test cases by using data-flow techniques. In the coverage criteria, they focused on state coverage, transition coverage, all-definition coverage, and all du-path coverage.

Hashim and Salman [22] proposed a test case generation algorithm from an UML activity diagram. They generated the test case by converting the UML activity diagram to an activity graph that stores all activity information. The graph will be used to automatically generate an activity path that contains all possible test case paths. From all the stored information and paths, the test case will be generated automatically. Furthermore, a prototype was created to implement and test the algorithm.

Kundu and Samanta [23] proposed an approach for generating test cases by using UML activity diagrams. In their approach, they translated UML activity diagrams into an activity graph to generate test cases. From the result of the activity graph, they used DFS and the breadth first search algorithm to generate test cases. These generated test cases are based on an activity path coverage criterion and are used to cover loop faults and organization. To achieve UML activity diagram coverage, they considered a coverage criterion called activity path coverage criterion.

Swain, et al. [24] proposed an approach to generate test cases from UML state chart diagrams. They named their approach automatically generating test cases from state chart diagram. First, they constructed a state chart diagram for a given object. Second, they traversed the state chart diagram, selected the conditional predicates, and transformed these conditional predicates into source code. Finally, the test cases are generated and stored by using function minimization technique. From the state chart diagram, they performed a DFS to select the associated predicates. After selecting the predicates, they guessed an initial dataset. They then generated test predicate conditions from a state chart diagram to generate test cases. Their technique accomplishes little coverage in test cases such as transition pair coverage, state coverage, action coverage, and transition coverage. It also achieves fully predicate coverage by generating a test data for each conditional clause. Furthermore, their approach can handle transitions with guards and achieves transition path coverage.

Swain, et al. [25] proposed an approach for test case generation called test generation and minimization for O-O software with state charts. This method analyzes the system that will be tested and accepted by a user and then builds the state chart diagram. They then converted the given UML state chart diagram into an intermediate model and named it a state transition graph. DFS is used to form test sequences and generate all possible paths. Thereafter, they obtained all the valid sequences of the application until a final edge is reached. Finally, they minimized a set of test cases by calculating the node coverage for each test sequence. In the same year, Swain, et al. [26] performed a similar experiment to generate test cases from UML state chart diagrams and named their method generation and minimization of test cases from state charts. First, they built a state chart diagram model for SUT. Second, they conjugated a state transition graph from the state chart diagram. Third, by using the graph, they extracted all of the required information. Fourth, by applying the algorithm of Wang [27] they generated the test cases. Finally, they minimized the set of test cases by calculating the node coverage for each test case to help them determine which test case are covered by other test cases.

III. TEST CASE GENERATION PROCESSES

This section present studies that explained the processes and important components used in generating test cases.

The components of a test case will differ from system to system. However, in its simplest form, the components will be a series of events that lead to a certain execution path with certain conditions. The values for attributes and parameters can be generated on the basis of any constraints and then supplied to the program for test execution [28].

Test case generation has a strong influence on the effectiveness and efficiency of the complete testing process is one of the most critical knowledge demanding tasks [29, 30]. Test cases are typically generated from manual or automatic inputs. Manual generation depends on the expertise of the software testers. However, existing methods for the automatic generation of test cases still need to be enhanced and improved [31].

Various test case generation methodologies using UML diagrams have been proposed by a number of researchers, software developers, and software testers by using many algorithm types and methods. From the UML diagrams, test case generation starts by storing the UML diagram information in a database; and the diagram is then transformed into a graph model [32]. Thereafter, the test paths are generated from the graph model; these paths will help identify all possible routes that the software will follow and form them into test case [33].

The test case generator contains three main phases, which are essential in generating a test case diagram. These phases are shown in Figure 1. The first phase analyzes the developed components of the system and delivers the data to the second phase. The second phase investigates the data to determine the appropriate paths; these paths may represent the high coverage criteria. The third phase tests these paths as arguments. The third phase may provide feedback to the second phase regarding any impracticable paths [34].

According to Verma and Dutta [35], Shanthi and Kumar [36], Boghdady, et al. [37], a reduced form of the stored database (i.e., a dependency table) is needed. This table is generated from the database created for each UML diagram in any system and will be called a chart dependency table (CDT), which covers all the functionalities in the UML diagram. The CDT is then used to automatically generate a directed graph called chart dependency graph (ADG), which is used in conjunction with the CDT as an intermediate model. The ADG will be used later to generate all possible test paths. Furthermore, the reliability of the intermediate model will increase because of consistency checking constraints and automatic information entering.

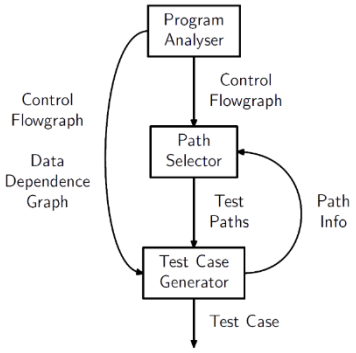

Figure 1: Adopted Architecture of a Test Case Generator System [34]

While converting a UML diagram specification into a CDT, a blind ADG product of the basic states is not generated within each parallel component. The generated machine is the possible combination of configurations based on simulated events.

Moreover, one or more arcs can be pruned to avoid generating a larger machine [16]. On one hand, class graph pruning has a significant drawback of not testing the entire machine. On the other hand, applications of test case generating methods are feasible on complex systems.

A graph describes the logic structure of a software module: the nodes represent computational statements or expressions, the edges represent transfer of control between nodes, and each possible execution path of the module has a corresponding path from the entry to the exit node of the graph [21].

Therefore, after formulating all the necessary information, an algorithm is needed to generate all possible paths [22, 23] on the basis of several possible coverage criteria. From the generated paths, a test case generation algorithm will generate the test case [25, 38].

IV. PROPOSED TEST CASE GENERATION MODEL

On the basis of reviews in Sections 2 and 3, the common processes and components implement by these studies are extracted to produce a test case generation model where a UML diagram is used an input and generate test cases as an output. Furthermore, a database that contains the information from the UML diagram should be used to generate the test cases by extracting the correlating information [8].

The proposed model is composed of six components for test case generation: CDT, chart dependency graph, consistency checking, class graph pruning, test path generation, and test case generation. The proposed model is shown in Figure 2.

On the basis of the proposed model in Figure 2, the development targets will be achieved by using the following processes:

1. Use a UML diagram to define and represent the software development specifications.
2. Construct the CDT automatically as follows: (a) for each pair of distinct classes, fulfill the hierarchical relationships on the basis of the influences entered from the UML diagram; (b) the automatic checking and storing for existence symmetric parent–child or ancestor descendent hierarchical relations for any pair of states; (c) avoid the inconsistency problem by the automatic detect for classes relationships based on set of rules; (d) automatic deduction of new hierarchical relations (if possible).
3. Create the chart dependency graph automatically from the CDT by using the relation stored in the table.
4. Remove the duplication sub-tree from the chart dependency graph to avoid illegitimate test cases and generate the test paths by using class graph pruning.
5. Generate all possible paths by using test path generation from the pruning chart dependency graph.
6. Generate test cases automatically from test path generation.

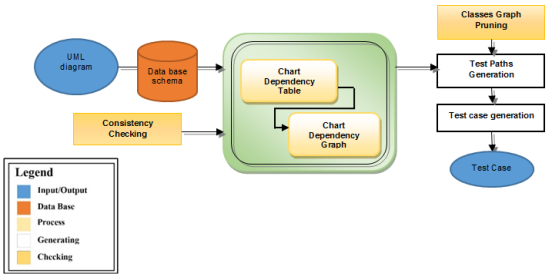

Figure 2: Proposed Automated Test Case Generation Model

Test cases will be generated with the help of stored strings in the database in the form of tables. In the database, the table of the UML diagram has three columns: pre-condition, test chart, and post-condition. The pre- and post-conditions in the UML diagram are given in the documentation of each particular message of the UML diagram [8].

When the class name is found, it has been entered into the database with the related class attributes, along with its attributes, operations, inheritance classes, dependency, and cardinality. Furthermore, every string is stored.

V. CONCLUSIONS

This study discusses the concept of test case generation by using UML diagrams to investigate the current models and processes related to test case generation. A model for performing automated test case generation is proposed to help researchers map the generation of test cases in their work by using UML diagrams. This model will help in settling the requirement for future algorithms and methods. Furthermore, an intermediate graph should be constructed and the steps should be checked continually.

This model can be implemented in many types of UML diagrams, such as state chart, activity diagram, and sequence diagram. This paper also focuses on proposing general processes of test case generation. In the future, an algorithm will be developed on the basis of this model to achieve high coverage in the process.

REFERENCES

[1] Kull A., 2009.Model-Based Testing of Reactive Systems: *TUT Press.*

[2] Dustin E., Garrett T., and Gauf B., 2009. Implementing Automated Software Testing: How To Save Time And Lower Costs While Raising Quality: *Pearson Education.*

[3] A. Kaur and S. S. Harwinder, 2013. Automatic Test Case Generation with SilK Testing. *International Journal of Computer Applications*. 79:32-34.

[4] Anand S., Burke E. K., Chen T. Y., Clark J., Cohen M. B., Grieskamp W., et al., 2013. An Orchestrated Survey Of Methodologies For Automated Software Test Case Generation. *Journal of Systems and Software.* 86:1978– 2001.

[5] Li L., Li X., He T., and Xiong J., 2013. Extenics-based Test Case Generation for UML Activity Diagram, *Procedia Computer Science*. 17:1186-1193.

[6] Specification O. A., 2007. OMG Unified Modeling Language (OMG UML), Superstructure, V2. 1.2. *Object Management Group*.

[7] Sapna P. and Balakrishnan A., 2015. An Approach for Generating Minimal Test Cases for Regression Testing,*Procedia Computer Science*. 47:188-196.

[8] Karambir and Kuldeep K., 2013. Survey of Software Test Case Generation Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*. 937-942.

[9] Hooda I. and Chhillar R., 2014. A Review: Study of Test Case Generation Techniques. *International Journal of Computer Applications*. 107.

[10] Rafi D. M., Moses K. R. K., Petersen K., and Mäntylä M. V., 2012. Benefits And Limitations Of Automated Software Testing: Systematic Literature Review And Practitioner Survey. in *Proceedings of the 7th International Workshop on Automation of Software Test*. 36-42.

[11] Kumaran U. S., Kumar S. A., and Kumar K. V., 2011. An Approach to Automatic Generation of Test Cases Based on Use Cases in the Requirements Phase. *International Journal on Computer Science and Engineering*. 3: 102-113.

[12] Hierons R. M., Merayo M. G., and Nunez M., 2011. Scenarios-Based Testing Of Systems With Distributed Ports, Software: Practice and Experience. 41:999-1026.

[13] Hartmann J., Imoberdorf C., and Meisinger M., 2000. UML-based integration testing. in *ACM SIGSOFT Software Engineering Notes*. 60-70.

[14] Kansomkeat S. and Rivepiboon W., 2003. Automated Generating Test Case Using UML Statechart Diagrams, in *Proceedings Of The 2003 Annual Research Conference Of The South African Institute Of Computer Scientists And Information Technologists On Enablement Through Technology*.296-300.

[15] Kosindrdecha N. and Daengdej J., 2010. A Test Generation Method Based On State Diagram. *JATIT*. 28-44.

[16] Santiago V., do Amaral A. S. M., Vijaykumar N., Mattiello-Francisco M. F., Martins E., and Lopes O. C., 2006. A Practical Approach for Automated Test Case Generation using Statecharts. in *Computer Software and Applications Conference, COMPSAC'06. 30th Annual International.*183-188.

[17] Santiago V., Vijaykumar N. L., Guimarães D., Amaral A. S., and Ferreira É., 2008. An Environment for Automated Test Case Generation from Statechart-based and Finite State Machine-based Behavioral Models. in Software Testing Verification and Validation Workshop. *ICSTW'08. IEEE International Conference on.* 63-72.

[18] Indumathi C. andSelvamani K., 2015. Test Cases Prioritization Using Open Dependency Structure Algorithm. *Procedia Computer Science*. 48:250-255.

[19] Boghdady P. N., Badr N. L., Hashim M. A., and Tolba M. F., 2011. An Enhanced Test Case Generation Technique Based On Activity Diagrams, *in Computer Engineering & Systems (ICCES), 2011 International Conference on*. 289-294.

[20] Binder R. V., 2000. Testing Object-Oriented Systems: Models, Patterns, and Tools. : *Addison-Wesley Professional.*

[21] Shirole M., Suthar A., and Kumar R., 2011. Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm, in *Proceedings of the 4th India Software Engineering Conference.*125-134.

[22] Hashim N. L. and Salman Y. D., 2011. An Improved Algorithm in Test Case Generation from UML Activity Diagram Using Activity Path. Proceedings of the 3rd *International Conference on Computing and Informatics, ICOCI.*

[23] Kundu D. and Samanta D., 2009. A Novel Approach to Generate Test Cases from UML Activity Diagrams, *Journal of Object Technology*. 8:65-83.

[24] Swain R. K., Panthi V., Behera P., and Mohapatra D., 2012. Automatic Test case Generation From UML State Chart Diagram. *International Journal of Computer Applications*. 26-36,

[25] Swain R. K., Behera P. K., and Mohapatra, D. P. 2012. Minimal TestCase Generation for Object-Oriented Software with State Charts. arXiv preprint arXiv:1208.2265.

[26] Swain R. K., Behera P. K., and Mohapatra D. P., 2012. Generation and Optimization of Test cases for Object-Oriented Software Using State Chart Diagram. arXiv preprint arXiv:1206.0373.

[27] Linzhang W., Jiesong Y., Xiaofeng Y., Jun H., Xuandong L., and Guoliang Z., 2004. Generating test cases from UML activity diagram based on Gray-box method. presented at *the Software Engineering Conference* 2004. *11th Asia-Pacific.*

[28] Rapos E., 2012. Understanding The Effects Of Model Evolution Through Incremental Test Case Generation For UML-RT Models.

[29] Zhu H., Hall P. A., and May J. H., 1997. Software unit test coverage and adequacy. Acm computing surveys (csur). 29:366-427.

[30] Bertolino A., 2007. Software testing research: Achievements, challenges, dreams. in 2007 Future of Software Engineering. 85-103.

[31] Koong C.-S., Shih C., Hsiung P.-A., Lai H.-J., Chang C.-H., Chu W. C., et al., 2012. Automatic testing environment for multi-core embedded software—ATEMES, *Journal of Systems and Software*. 85:43-60.

[32] Priya S. S. and Sheba P., 2013. Test Case Generation from UML models-A survey. in *Proc. International Conference on Information Systems and Computing (ICISC-2013)*, INDIA.

[33] Werner E. and Grabowski J., 2012. Mining Test Cases: Optimization Possibilities. *International Journal On Advances in Software*. 5:200-211.

[34] Edvardsson J., 1999. A survey on automatic test data generation. in *Proceedings of the 2nd Conference on Computer Science and Engineering*. 21-28.

[35] Verma A. and Dutta M., 2014. Automated Test case generation using UML diagrams based on behavior. I*nternational Journal of Innovations in Engineering and Technology (IJIET)*. 4

[36] Shanthi A. and Kumar G. M., 2012. Automated Test Cases Generation from UML Sequence Diagram. *International Proceedings of Computer Science & Information Technology*. 41.

[37] Boghdady P. N., Badr N. L., Hashem M., and Tolba M. F., 2011. A proposed test case generation technique based on activity diagrams. *International Journal of Engineering & Technology IJET-IJENS*. 11.

[38] Swain S. K., Mohapatra D. P., and Mall R., 2010. *Test case generation based on use case and sequence diagram*. *International Journal of Software Engineering*. 3:21-52.