Coverage Criteria for UML State Chart Diagram in Model-based Testing

Yasir Dawood Salman, Nor Laily Hashim, Mawarny Md Rejab, Rohaida Romli, Haslina Mohd
Human-Centered Computing Research Lab, Universiti Utara Malaysia, Kedah, Malaysia
yasir.dawod@gmail.com

***Abstract*—Software testing is a necessary and essential part of the software quality process and plays a major role in detecting errors in systems. To improve the effectiveness of test case generation during software testing, and with the growing adoption of UML by software developers and researchers, many studies have focused on the automation of test case generation from UML diagrams. One of these diagrams is the UML state chart diagram. These test cases are generally generated to achieve certain coverage criteria. However, combinations of multiple criteria are required to achieve better coverage. Different studies use various number and type of coverage criteria in their methods and approaches. This paper reviews previous studies to present the most practical coverage criteria combinations for UML state chart diagram, including all-states, all-transitions, all-transition-pairs and all-loop-free-paths coverage. A special calculation is necessary to determine the coverage percentage of the proposed coverage criteria. This paper presents a calculation method to achieve this goal with an example is applied to a UML state chart diagram. This finding would be beneficial in the area of automatic test case generating for model-based testing and especially in the UML state chart diagram.**

***Index Terms*—Coverage Criteria; Test Case Generation; UML State Chart Diagram.**

I. INTRODUCTION

Testing is an important stage of software development, and it provides a method to establish confidence in software reliability. Testing is a challenging task for the analysis of unified modelling language (UML) models, given that information regarding a system is distributed across several model views [1].

UML diagrams aimed to assist in reducing the complexity of a problem with the increase in product sizes and complexities [2]. Still, UML diagrams are large and complex, involving thousands of interactions across hundreds of objects. Owing to the model's complexity, generating test models (e.g., control flow graph from source code) is cumbersome. This situation is especially true in large programs [1].

Model-based testing which uses UML design specifications for testing overcomes the deficiencies that are very difficult to identify in the system state information, either from the code or from the requirement specifications, therefore it has been developed as a promising testing method [3].

The test cases could be generated from requirements specification and design documents, where the UML state chart diagram is one of the diagrams used in the system design early life cycle. The using of UML state chart diagram will generate test cases for the software development, what will make the software testing much more efficient and effective [4]. Enhancing the necessary tools and increasing the automation of software testing would help to decrease the expenses of software development and improve software reliability [5], what would lower the negative economic issue of defective software.

For the past decade, a great amount of research work has been conducted over automatic test case generation from UML state chart diagram [2, 6-11]. The purpose of generating test case using UML state chart diagram is to verify the relations between the behaviour, state transition, state, action, and event. This technique is used to determine if one can fulfil the system specifications through the state-based motion of the system [12].

Test data generation is one of the most time-consuming tasks during software testing, especially for manual testing. With the rapid development of software, many researchers have worked on solving the problem of automatic test data generation [13]. These test cases can be generated according to structural coverage criteria [14]. Coverage criteria are adequacy measures to qualify if a test objective is satisfied when executing test cases on a system under test [15]. Coverage criteria are established to estimate the quality of test cases, and criteria combinations are considered in software testing [16].

Test coverage specifies the degree of the testing been standard such as basis path testing or path testing is achieved. The whole performance from the beginning to the end is represented by a path. Path testing is a testing technique that from the domain of all possible paths through the program [17].

A series of statements, instructions, or high-level design is called a path of software. This path begins with a decision, junction, or entry and comes to end at the same or different decision, exit, or junction. Moreover, the path may experience many decisions, processes and junctions once, twice, or more [18]. The way to divide the program input domain into a path is by use of a suitable test coverage criterion [17].

This paper focuses on determining the factual combination of coverage criteria for test case generation from the UML state chart diagram, given that this area has attracted several researchers in the previous years. However, no practical coverage criteria combinations are available to support this testing, thus far. The objective of this paper is to review the current test coverage criteria for UML state chart diagram and proposed a suitable coverage criteria combination to achieve the highest coverage, also a calculation method for this coverage criteria.

The remainder of this of this paper is organized as follows: the next section discusses coverage criteria testing using the

UML state chart diagram. Calculation of the coverage criteria is discussed next. Finally, the conclusion of the study is presented.

II. BACKGROUND

Coverage criteria on software systems can be defined as the set of conditions and rules imposing a set of test requirements on a software test [19]. A number of coverage criteria are available for testing, and most of them are based on the information of control and data flows [20]. Test coverage criteria enhance the generation of comprehensive test cases based on the number of elements to cover or visit within a diagram.

A test coverage criterion is crucial in validating and analysing the test adequacy of test cases [21]. They can also be used to direct and stop the test case generation processes.

When applying model-based coverage criteria to some model, it can be compared by subsuming them. This subsuming coverage criterion will be considered stronger than the individually subsumed coverage criterion. For example, in satisfying the coverage, all transitions coverage is considered as the minimum coverage criterion. Most of the commercial test generators tools are only able to satisfy slightly weak coverage criteria. For example, the SmarTesting LTD tool is only able to cover all-Transitions coverage criteria [22].

Each test generation method targets certain specific features of the system to be tested. Using test coverage analysis, the extent to which the targeted features are tested can be determined using test coverage analysis. The important coverage analysis based on a model can be the following: all model parts coverage is achieved when at least once the test reaches every part in the model [3].

This section introduces the eight most common transition-based coverage criteria used in test case generation, namely, all-states coverage, all-configurations coverage, all-transitions coverage, all-transition-pairs coverage, all-loop-free-paths coverage, all-one-loop-paths coverage, all-round-trips coverage, and all-paths coverage [23]. Figure 1 shows these criteria.

Notably, the all-loop-free-paths, all-one-loop-paths, and all-round-trips coverage criteria can be relatively inadequate by themselves because they do not guarantee that all states (let alone all transactions) are covered [23].

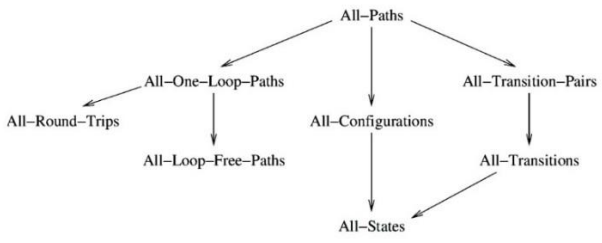

Figure 1: Hierarchy of transition-based criteria [23]

Using an extreme example, a UML state chart diagram primarily loops around a self-transition a few times until a counter reaches a particular value, which then enables the transition leading to the rest of the UML state chart. For this example, the all-loop-free-paths criterion can be satisfied with an empty test case; the all-round-trips criterion can be satisfied with only a single test (one loop around the self-transition); and Binder's algorithm for generating an all-round-trips test case generate tests containing unsatisfiable guards, thereby disabling execution [23]. This finding shows that these coverage criteria should be combined with other criteria, such as all-states or all-transitions, to ensure that the entire UML state chart is covered. Utting and Legeard [23] recommend that all test cases generated from transition-based models satisfy all-transitions coverage as a minimum measure of quality. The following are the proposed coverage criteria for the UML state chart diagram:

All-States Coverage is required to visit every model state at least once by a test case within [23, 24]. This criterion covers all states in every state chart diagram for basic test generation. State coverage is a test adequacy criterion requiring tests to check the output variables of a program. All variables defined when executing a test scope (even those that are invisible, such as private fields of objects) are considered by state coverage [25].

However, the all-states coverage criterion is considered the weakest structural coverage criterion [15]; still, few studies adapted this coverage criterion [7, 10, 24-30].

All-Transitions Coverage specifies that each transition must be fired at least once in some test cases [15, 23]. To test a transition, the test case requires that the object under test be in the accepting state of the transition. The technique does not place any constraints on how to reach the accepting state [31]. This coverage criterion is proposed by several authors on generating test cases from state chart diagrams [6-10, 25-28, 30, 32-36]. Therefore, this coverage criterion is one of the most commonly used.

All-Transition-Pairs Coverage considers adjacent transitions successively entering and leaving a given state. This coverage specifies that for each state, each couple of exiting transition has to be fired at least once [15]. Thus, the transition-pair coverage subsumes the all-transitions coverage. The transition-pair coverage criterion generates more test cases than the transition coverage criterion [37]. Given that all-transition-pairs coverage is not widely used by researchers; Santiago, et al. [9], Offutt, et al. [34], Briand, et al. [38] used all-transition-pairs coverage in their studies. For transition coverage, pairs that are executable by at least one product are considered in the ratio that covers the parallel path [15].

All-Configurations Coverage is required to visit every configuration of the UML state chart diagram at least once. This coverage criterion is the same as all-states coverage for systems with no parallelism [23].

All-One-Loop-Paths Coverage returns all paths containing one cycle at most; thus, each generated path contains one and only one repeated state at most [39]. In other words, this condition requires visiting all the loop-free paths through the model, including all the paths that loop once [40]. Muniz, et al. [39] covered all-one-loop-paths for model-based testing but not for UML state chart diagram in their work.

All-Loop-Free-Paths Coverage must traverse every loop path at least once. A path that does not contain any type of repeating is called loop-free [23]. Notably, this coverage does not frequently cover all transitions. Similarly, this coverage does not constantly cover all states. However, all-one-loop-paths test cases include all paths of the all-loop-free-paths coverage criterion. Therefore, using all-one-loop-paths is sufficient.

All-Round-Trips Coverage is similar to the all-one-loop-paths criterion because it requires a test for each loop in the

model; furthermore, that test only has to perform one iteration around the loop. Nevertheless, this coverage is weaker than all-one-loop-paths because all the paths preceding or following a loop does not require testing [23]. However, Briand, et al. [38] used all-round-trips in their work.

All-Paths Coverage specifies that each executable path should be followed at least once when executing the abstract test case on it [15]. The all-paths criterion corresponds to the exhaustive testing of the state chart diagram model [23]. Few studies consider this coverage in their coverage criteria [27, 28, 35, 41] because it is generally impractical, given that such models typically contain an infinite number of paths due to loops [23].

From the above review, all-state coverage is the weakest coverage, but it still awaits acknowledgement for its importance and comprehensive use. All-transitions coverage and all-transitions-pair coverage are impotent in parallel paths; furthermore, they cover all decision and guard states. These coverage criteria are used by most of the reviewed papers. In all-loop-free-paths, all-one-loop-paths, and all-round-trips coverage, the use of all-loop-free-paths is efficient by itself, given that the test from it covers both all-one-loop-paths and all-round-trips coverage. Conversely, all-path coverage is impractical because in loop cases, this coverage requires an infinite number of paths.

III. Proposed Coverage Criteria Calculation

In this section, an overview of the model to generate test sequence from UML state chart diagram is discussed and then, the selected test coverage will be calculated. However, this paper focuses only on the suitable coverage criteria for the UML state chart diagram. The schematic representation of the model is shown in Figure 2. The proposed methodology involves the following steps:

1. UML state chart diagram construction.
2. Convert the entered UML state chart diagram into a table named here State Relationship Table (SRT).
3. Convert the SRT into an intermediate graph. This intermediate graph named as State Relationship Graph (SRG).
4. Generate all the possible paths using the Generating test case paths algorithm from SRG.
5. Generate a set of test cases by using generating test case paths as an input, which achieves the proposed coverage criteria.

The ATM withdraws UML state chart diagram is selected as a case study. The UML state chart diagram is taken from [42] with some modifications as shown in Figure 3. This example is used to illustrate the transection from the UML state chart diagram to SRG as shown in Figure 4. Then applied the SRG as an example to calculate the proposed coverage criteria.

A coverage criterion can be a measured on any program during software development, such as source code, requirements, or design models. Coverage is usually counted as the percentage of test requirement satisfaction. The coverage attainments of the model assess the quality and completeness of the test case. Coverage criteria are derived from popular heuristics to measure the fault detection capability of test cases [21].

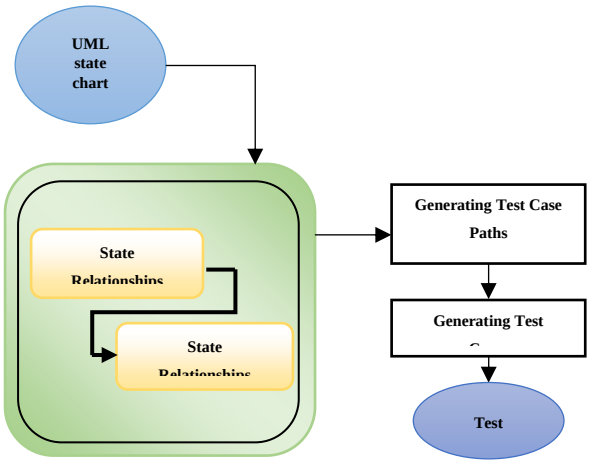

Figure 2: Test case generation model.

If a test case fulfils a set of test requirements in terms of structural elements, then, a coverage criterion is satisfied. Clearly specifying the coverage criteria is important because they are frequently used to measure the effectiveness of test case generation [43].

This section presents the methods of calculating the proposed coverage criteria prestige. These methods use the element coverage equation as the base. The percentage of criteria coverage is used to evaluate the accuracy or quality of test case generation approaches. The calculation formula for the percentage of coverage criteria is depicted in Equation 1. The formula indicates the number of elements contained in the UML diagram, which is exercised in the generated test cases [44].

$$E_c = \left(\frac{E_{tcs}}{E_{tcUML}} \times 100\right) \quad (1)$$

E_c : Elements coverage
E_{tcs} : Number of elements exercised in the test cases
E_{tcUML} : Number of elements in the UML diagram

As seen in Figure 3, State 1 represents the ATM card reading. If the card read guard condition is Yes, it will read the PIN code. However, if the card read guard condition is No, it will eject the card. A similar result is expected in reading the PIN; if the PIN guard condition is Yes, it will be processed to the selection of a transaction; the card will be ejected if the PIN guard condition is No; however, the card will be retained and aborted if an invalid PIN is entered. The user can choose the transaction; then, the transaction will be performed or cancelled; and finally, the card will be ejected. In performing a transaction, the customer can choose between conducting another transaction that results in a loop; then, the customer finishes the transaction and ejects the card.

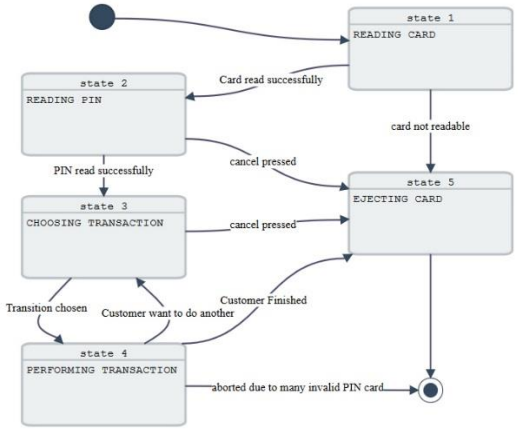

Figure 3: UML State Chart Diagram of an ATM Machine

Each state in the UML state chart is considered as vertex *V* in the state graph, and each transaction is presented as edge *E*. The following subsections discuss the calculation of the proposed coverage criteria.

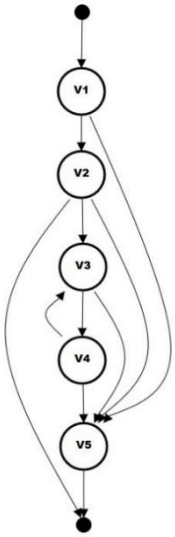

Figure 4: State Relationship Graph for the ATM Machine

All-State Coverage: by applying all-state coverage to the test model, full coverage can be achieved when every state of the UML state chart diagram is visited at least once. Through the sets $V_i = (V_1, V_2, V_3, \dots)$ and given that the total number of vertex (V_t) is equal to 5 without the "Start State" and "End State" in the example in Figure 4, every V_i should be covered at least once to accomplish full coverage. The all-state coverage percentage (C_{AS}) can be calculated by devising the visited vertex V_v on the total V_t ; the total coverage is achieved as follows:

$$C_{AS} = \left(\frac{V_v}{V_t} \times 100\right) \quad (2)$$

All-transition coverage: by applying all-transitions coverage to the test model, full coverage is achieved when the test cases visit every transition of the UML state chart diagram at least once. Each transition has a pre-vertex and a post- vertex [45]. Assume all-transitions *(AT)* so that $AT \in E$, and all-transitions coverage presents (C_{AT}). Given that *E = 11* in the example, in Figure 4, the following E should be covered at least once to accomplish full coverage:

$$E_1(V_0 \to V_1) \quad E_5(V_2 \to V_5) \quad E_9(V_4 \to V_5)$$
$$E_2(V_1 \to V_2) \quad E_6(V_2 \to V_d) \quad E_{10}(V_4 \to V_3)$$
$$E_3(V_1 \to V_5) \quad E_7(V_3 \to V_4) \quad E_{11}(V_5 \to V_d)$$
$$E_4(V_2 \to V_3) \quad E_8(V_3 \to V_5)$$

Each visited *E* has Boolean flag (0) and (1), and the total of its covered edges is E_d; the total coverage is achieved as follows:

$$C_{AT} = \left(\frac{E_d}{AT} \times 100\right) \quad (3)$$

All-transition-pair coverage: to obtain full all-transition-pairs coverage for the test model, visiting each pair of exiting transition of the UML state chart diagram at least once is necessary for the test cases. Assume all-transition-pairs coverage (C_{AP}) so that $C_{\mathrm{AP}} \in E$ and total decision verities $(V_{decision})$. Given that $V_{decision} = 4$ in the example, in Figure 3 (b), the following $V_{decision}$ should be covered at least once:

$$V_{\mathrm{d1}}[(V_1 \to V_2), (V_1 \to V_5)]$$
$$V_{\mathrm{d2}}[(V_2 \to V_3), (V_2 \to V_5), (V_2 \to V_{\mathrm{d}})]$$
$$V_{\mathrm{d3}}[(V_3 \to V_4), (V_3 \to V_5)]$$
$$V_{\mathrm{d4}}[(V_4 \to V_3), (V_4 \to V_5)]$$

Each visited $V_{decision}$ has Boolean flag (0) and (1) and its total is V_{dt}; the total coverage is as follows:

$$C_{\mathrm{AP}} = \left(\frac{V_{dt}}{V_{decision}} \times 100\right) \quad (4)$$

All-one-loop-path coverage: by applying all-one-loop-paths coverage to the test model, full coverage can be achieved when the generated test paths from the UML state chart diagram are visited in every loop, including all the paths that looped once.

$$E_{AOLP} = \left(\frac{LT}{TP} \times 100\right) \quad (5)$$

where E_{AOLP} refers to all-one-loop-paths coverage, and LT to the total number of generated loop test cases. Given that all the paths preceding or following a loop require testing, $LT = loop \times (included\ decision + 1) = 1(1 + 1) = 2$.

For the example in Figure 4, to accomplish all-one-loop-paths full coverage, the two paths in the generated loop test cases should be included in the final testing.

IV. CONCLUSION

This paper established the preliminary practical coverage criteria combinations to support test case generation from the UML state chart diagram. Coverage criteria are popular heuristic means to measure the fault detection capability of test cases. The selected coverage is constructed according to their concept and the previous works, which are all-states coverage, all-transitions coverage, all-transition-pairs coverage, and all-loop-free-paths coverage. Furthermore, this paper provides calculation methods for coverage criteria percentage. For future work, coverage criteria for different UML diagram can be defined and calculated, including the combination of two or more diagrams.

REFERENCES

[1] V. Panthi and D. P. Mohapatra, "Automatic test case generation using sequence diagram," in *Proceedings of International Conference on Advances in Computing*, 2012, pp. 277-284.

[2] Y. D. Salman and N. L. Hashim, "An Improved Method Of Obtaining Basic Path Testing For Test Case Based On UML State Chart," *Science International*, vol. 26, 2014.

[3] N. Pahwa and K. Solanki, "UML based Test Case Generation Methods: A Review," *International Journal of Computer Applications*, vol. 95, pp. 1-6, 2014.

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

[5] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mäntylä, "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*, 2012, pp. 36-42.

[6] V. Chimisliu and F. Wotawa, "Improving test case generation from UML statecharts by using control, data and communication dependencies," in *Quality Software (QSIC), 2013 13th International Conference on*, 2013, pp. 125-134.

[7] L. Li, T. He, and J. Wu, "Automatic Test Generation from UML Statechart Diagram Based on Euler circuit," *International Journal of Digital Content Technology & its Applications*, vol. 6, 2012.

[8] V. Santiago, N. L. Vijaykumar, D. Guimarães, A. S. Amaral, and É. Ferreira, "An environment for automated test case generation from statechart-based and finite state machine-based behavioral models," in *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on*, 2008, pp. 63-72.

[9] V. Santiago, A. S. M. do Amaral, N. Vijaykumar, M. F. Mattiello-Francisco, E. Martins, and O. C. Lopes, "A practical approach for automated test case generation using statecharts," in *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*, 2006, pp. 183-188.

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

[11] D. Patnaik, A. A. Acharya, and D. P. Mohapatra, "Generating testcases for concurrent systems using UML state chart diagram," in *Information Technology and Mobile Communication*, ed: Springer, 2011, pp. 100-105.

[12] Y. D. Salman and N. L. Hashim, "Automatic Test Case Generation from UML State Chart Diagram: A Survey," in *Advanced Computer and Communication Engineering Technology*, ed: Springer, 2016, pp. 123-134.

[13] X. Fan, F. Yang, W. Zheng, and Q. Liang, "Test Data Generation with A Hybrid Genetic Tabu Search Algorithm for Decision Coverage Criteria," 2015.

[14] E. Jee, D. Shin, S. Cha, J. S. Lee, and D. H. Bae, "Automated test case generation for FBD programs implementing reactor protection system software," *Software Testing, Verification and Reliability*, vol. 24, pp. 608-628, 2014.

[15] X. Devroey, G. Perrouin, A. Legay, M. Cordy, P.-Y. Schobbens, and P. Heymans, "Coverage criteria for behavioural testing of software product lines," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2014, pp. 336-350.

[16] J. M. Rojas, J. Campos, M. Vivanti, G. Fraser, and A. Arcuri, "Combining multiple coverage criteria in search-based unit test generation," in *International Symposium on Search Based Software Engineering*, 2015, pp. 93-108.

[17] A. Goodubaigari, "A Software Test Data Generation Tool for Unit Testing Of C++ Programs Using Control Flow Graph," *IJECS*, pp. 2388-2392, 2013.

[18] R. Mall, *Fundamentals of software engineering*. New delhi: PHI Learning Pvt. Ltd, 2009.

[19] A. A. Saifan and W. B. Mustafa, "Using Formal Methods for Test Case Generation According to Transition-Based Coverage Criteria," *Jordanian Journal of Computers and Information Technology*, vol. 1, pp. 15-30, 2015.

[20] H. S. Hong and H. Ural, "Using model checking for reducing the cost of test generation," in *International Workshop on Formal Approaches to Software Testing*, 2004, pp. 110-124.

[21] M. Shirole and R. Kumar, "UML Behavioral Model Based Test Case Generation: A Survey," *ACM SIGSOFT Software Engineering Notes*, vol. 38, pp. 1-13, 2013.

[22] S. Weißleder and D. Sokenou, "ParTeG-A Model-Based Testing Tool," *Softwaretechnik-Trends*, vol. 30, 2010.

[23] M. Utting and B. Legeard, *Practical model-based testing: a tools approach*: Morgan Kaufmann, 2010.

[24] H. Li and C. P. Lam, "An ant colony optimization approach to test sequence generation for state-based software testing," in *Quality Software, 2005.(QSIC 2005). Fifth International Conference* 2005, pp. 255-262.

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

[26] V. Chimisliu and F. Wotawa, "Model based test case generation for distributed embedded systems," in *Industrial Technology (ICIT), 2012 IEEE International Conference on*, 2012, pp. 656-661.

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

[28] M. Shirole, A. Suthar, and R. Kumar, "Generation of improved test cases from UML state diagram using genetic algorithm," in *Proceedings of the 4th India Software Engineering Conference*, 2011, pp. 125-134.

[29] N. Kosindrdecha and J. Daengdej, "A test generation method based on state diagram," *JATIT*, pp. 28-44, 2010.

[30] S. Kansomkeat and W. Rivepiboon, "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*, 2003, pp. 296-300.

[31] J. Al Dallal and P. Sorenson, "Generating class based test cases for interface classes of object-oriented black box frameworks," *Transactions on Engineering, Computing and Technology*, vol. 16, pp. 90-95, 2006.

[32] V. Chimisliu and F. Wotawa, "Using dependency relations to improve test case generation from UML statecharts," in *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, 2013, pp. 71-76.

[33] S. K. Swain, D. P. Mohapatra, and R. Mall, "Test Case Generation Based on State and Activity Models," *Journal of Object Technology*, vol. 9, pp. 1-27, 2010.

[34] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, "Generating test data from state- based specifications," *Software Testing, Verification and Reliability*, vol. 13, pp. 25-53, 2003.

[35] S. Ali, L. C. Briand, M. J.-u. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem, "A state-based approach to integration testing based on UML models," *Information and Software Technology*, vol. 49, pp. 1087–1106, 2007.

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

[37] R. Blanco, J. Fanjul, and J. Tuya, "Test case generation for transition-pair coverage using Scatter Search," *International Journal of Software Engineering and Its Applications*, vol. 4, pp. 37-56, 2010.

[38] L. C. Briand, Y. Labiche, and J. Cui, "Automated support for deriving test requirements from UML statecharts," *Software & Systems Modeling*, vol. 4, pp. 399–423, 2005.

[39] L. L. Muniz, U. S. Netto, and P. H. M. Maia, "TCG-a model-based testing tool for functional and statistical testing," in *ICEIS (2)*, 2015, pp. 404-411.

[40] M. Utting and B. Legeard, *Practical model-based testing: a tools approach*. san francisco: Morgan Kaufmann, 2007.

[41] P. Murthy, P. Anitha, M. Mahesh, and R. Subramanyan, "Test ready UML statechart models," in *Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools*, 2006, pp. 75-81.

[42] M. A. Ali, K. Shaik, and S. Kumar, "Test case generation using UML state diagram and OCL expression," *International Journal of Computer Applications*, vol. 95, 2014.

[43] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, pp. 742-762, 2010.

[44] O. Oluwagbemi and H. Asmuni, "Automatic Generation of Test Cases from Activity Diagrams for UML Based Testing (UBT)," *Jurnal Teknologi*, vol. 77, 2015.

[45] A. Paul and O. Jeff, *Introduction to Software Testing*. New York, NY, USA: Cambridge University Press, 2008.