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 will 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 criterion [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].

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 coverage 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].

If a test case fulfills 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].

\[
E_c = \left( \frac{E_{cS}}{E_{cUML}} \right) \times 100
\]

\( E_c \): Elements coverage
\( E_{cS} \): Number of elements exercised in the test cases
\( E_{cUML} \): 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.
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.

**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 \( V_t = \{V_1, V_2, V_3, \ldots \} \) and given that the total number of vertex \( |V_t| \) is equal to 5 without the “Start State” and “End State” in the example Figure 4, every \( V_t \) 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 \( V_t \) on the total \( V_t \); the total coverage is achieved as follows:

\[
C_{AS} = \left( \frac{|V_t|}{V_t} \times 100 \right)
\]  

(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 \( C_{AP} \). Given that \( E = 11 \) in the example, in Figure 4, the following \( E \) should be covered at least once to accomplish full coverage:

\[
\begin{align*}
E_1(V_0 \rightarrow V_1) &\quad E_5(V_2 \rightarrow V_5) &\quad E_6(V_4 \rightarrow V_6) \\
E_2(V_1 \rightarrow V_2) &\quad E_6(V_2 \rightarrow V_5) &\quad E_{10}(V_4 \rightarrow V_3) \\
E_3(V_1 \rightarrow V_3) &\quad E_7(V_3 \rightarrow V_4) &\quad E_{11}(V_5 \rightarrow V_1) \\
E_4(V_2 \rightarrow V_3) &\quad E_8(V_3 \rightarrow V_5) \\
E_9(V_4 \rightarrow V_2) &
\end{align*}
\]

Each visited \( E \) has Boolean flag (0) and (1), and the total of its covered edges is \( E_{\text{covered}} \); the total coverage is achieved as follows:

\[
C_{AT} = \left( \frac{E_{\text{covered}}}{E} \times 100 \right)
\]  

(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 \( C_{AP} \in E \) and total decision verities \( V_{\text{decision}} \). Given that \( V_{\text{decision}} = 4 \) in the example, in Figure 3(b), the following \( V_{\text{decision}} \) should be covered at least once:

\[
\begin{align*}
V_{d1} &\quad \{V_1 \rightarrow V_2, (V_1 \rightarrow V_5)\} \\
V_{d2} &\quad \{V_2 \rightarrow V_3, (V_2 \rightarrow V_5), (V_2 \rightarrow V_6)\} \\
V_{d3} &\quad \{V_3 \rightarrow V_4, (V_3 \rightarrow V_5)\} \\
V_{d4} &\quad \{V_4 \rightarrow V_3, (V_4 \rightarrow V_5)\}
\end{align*}
\]

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

\[
C_{AP} = \left( \frac{V_{\text{decision}}}{V_{\text{decision}}} \times 100 \right)
\]  

(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.

\[
E_{\text{AOLP}} = \left( \frac{LT}{TP} \times 100 \right)
\]  

(5)

where \( E_{\text{AOLP}} \) 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, \( LT = loop \times (\text{included decision} + 1) = 1(1 + 1) = 2 \).

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


