International Journal of Electrical and Computer Engineering (IJECE)
Vol. 11, No. 1, February 2021, pp. 575~588
ISSN: 2088-8708, DOI: 10.11591/ijece.v11i1.pp575-588

Target-based test path prioritization for UML activity diagram using weight assignment methods

Walaiporn Sornkliang[1], Thimaporn Phetkaew[2]
[1]Management of Information Technology Program, School of Informatics, Walailak University, Thailand
[2]School of Informatics, Walailak University, Thailand

Article Info

Article history:

Received Apr 9, 2020
Revised Jun 11, 2020
Accepted Jun 28, 2020

Keywords:

Edge weight assignment
Node weight assignment
Regression testing
Target-based prioritization
Test path prioritization
UML acitivity diagram

ABSTRACT

The benefit of exploratory testing and ad hoc testing by tester's experience is that crucial bugs are found quickly. Regression testing and test case prioritization are important processes of software testing when software functions have been changed. We propose a test path prioritization method to generate a sequence of test paths that would match the testers' interests and focuses on the target area of interest or on the changed area. We generate test paths form the activity diagrams and survey the test path prioritization from testers. We define node and edge weight to the symbols of activity diagrams by applying Time management, Pareto, Buffett, Binary, and Bipolar method. Then we propose a test path score equation to prioritize test paths. We also propose evaluation methods i.e., the difference and the similarity of test path prioritization to testers' interests. Our proposed method had the least average of the difference and the most average of the similarity compare with the tester's prioritization of test paths. The Bipolar method was the most suitable for assigning weights to match test path rank by the tester. Our proposed method also has given the affected path by changing area higher priority than the other test path.

Corresponding Author:

Thimaporn Phetkaew,
School of Informatics,
Walailak University,
222 Thaiburi, Thasala District, NakhonSiThammarat, 80160, Thailand.
Email: pthimapo@wu.ac.th, thimaporn.p@gmail.com

1. INTRODUCTION

Test case prioritization by importance is intended to rearrange the test cases to suit the needs of software testing. Those test cases that are the most important will be tested prior to the less important cases. Test case prioritization is very useful when faced with limited resources or limited testing time. Test case prioritization could rank test cases based on important objectives, i.e. facilitate target area or modification-aware testing. Several studies broadly categorized the test case prioritization into coverage-based, requirements-based, risk-based, search-based, fault-based, history-based, and others e.g., cost-aware based [1-3]. Coverage-based test case prioritization is a testing method that examines the code directly. The primary measures i.e., code, statement, branch, and function coverage were the most widely used criteria [1, 3-5]. Requirements-based test case prioritization uses the requirements information to classify serious test cases [1]. It is to check whether the delivered software meets the needs of the customer or not [6]. Risk-based test case prioritization prioritizes the test cases that concern the major risks that affected the software [1]. Search-based test case prioritization finds the optimal ranking of the test cases by searching from the global space to fit the objectives, such as greedy, genetic algorithm, ant colony

optimization [1, 2, 7]. Some studies used a concept of hybridizing A* algorithm and ant colony optimization to generate and optimize the test paths [8]. Fault-based test case prioritization sequences the test case to detect targeted faults [1, 9]. History-based test case prioritization uses the history data, for example, the count of executions which revealed a fault to prioritize the test case [1].

Object-Oriented technology is currently widely used in developing software. The concept of Object-Oriented programming such as data dependence, control dependence, and dependency due to object relations are used for test case selection and test case prioritization [10]. The unified modeling language (UML) is a standard model used to design object-oriented programming [4]. Several studies have employed a UML model to generate and prioritize test paths by converting activity diagram into a tree structure [11], or a graph structure [5, 12-17] and then generating the test paths. Some studies have converted a sequence diagram into a graph structure [18, 19] and then generated test paths. Some have converted a state machine diagram [20, 21] into a graph structure and then generated test paths. Further, UML models with several diagrams have been converted and integrated into the same control flow graph to create and prioritize test paths. For example, by using a sequence diagram and interaction overview diagrams [22], using a sequence diagram and a state chart diagram [23], and using a use-case diagram, a sequence diagram, and an activity diagram [24]. In test case prioritization, several studies have assigned weights for activity diagrams or symbols of control flow graph to calculate the scores for test paths, and the scores were then used to prioritize the test paths. There are two groups of approaches to assign weights to the symbols of activity diagrams or control flow graphs. In the first group, the weights are assigned to nodes or symbols of the graph. Each symbol has a different weight. All the symbol weights affect the test path score [15, 16]. The weight calculation of a control flow graph node derives from the numbers of incoming edges to the node multiplied by the count of outgoing edges from the node [13]. In the second group, the weights are assigned to the edges of the graph. Some studies have assigned the weights to the outgoing edges of decision nodes [12]. All the weights are calculated to find the test path score and the scores are used to prioritize the test paths by its importance.

None of the above studies prioritized by the importance of the target area or of the changed area. Besides, they did not put an emphasis on the sequence according to the testers for their test paths of interest. Although special value testing or ad hoc testing [25] can be used to test the target area or changed area, these types of testing are only done by professionals in software testing. These professionals put a lot of effort into delivering appropriate test results that suit the customer's requirements. So, this study proposes a test path prioritization method to generate a sequence of test paths that would match the testers' interests and focuses on the target area of interest or on the changed area.

The rest of this paper is divided as follows. Section 2 explains the research method used in this study. Experimental results and discussion are presented in section 3. Enhancement of the algorithms are presented in section 4. Finally, section 5 concludes the paper.

Related work. Test case prioritization is a reordering of the test cases in a test suite for test executions. Test case prioritization aims to cover any given part of the program by: detecting faults earlier, finding high priority errors, focusing on testing of the last modified part, and reducing the time and cost of testing. Several studies have examined methods to prioritize test paths to increase the efficiency and effectiveness of testing [1, 26].

UML activity diagram is essentially a flowchart that chronologically organizes a set of activities that take place over time. The diagram symbols consist of initial, activity, transition, decision, merge, fork, join, swimlane, and final. A UML activity diagram is transformed into a graph structure or a tree structure to generate and prioritize test paths. In test case prioritization, weights are numeric values assigned to the symbols of a UML activity diagram or to the symbols of a control flow graph. Test case prioritization is then done according to the weights to determine which paths should be tested first [16]. The weights are set according to the symbols in the UML activity diagram. The UML activity diagram will be converted into a directed graph: $G=\{A, E, in, F\}$, where A represents the nodes consisting of action nodes, object nodes, and control nodes i.e., decision, merge, fork, and join nodes, E represents the edges, $E=\{(x, y)|x, y \in A\}$, in is an initial node, and F is a final node [4]. There are three approaches to assigning weights to the symbols of activity diagrams or to symbols of control flow graph. All weights are calculated to find out the test path scores. The scores were used to prioritize the test paths by importance. First, the weights are assigned to nodes of a graph [5, 13-18, 23, 24]. Second, the weights are assigned to edges of a graph [12]. Third, the weights are assigned to both nodes and edges of a graph [11, 22]. According to some reports, the symbols in UML activity diagram or in control flow graph were assigned weights as in Table 1, and the calculation methods of test path scores are shown in Table 2.

Table 1. Comparison of weight assignment methods

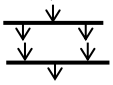
Name	Symbol	Assigned weights
Initial node		1) weight=0 [15]
Activity node		1) weight=1 [5, 11, 15, 16] 2) weight=4, if the node was inter-depended activity [16] 3) weight=6, if the node was control node [16]
Normal edge		1) weight=1 [12] 2) weight=2 [22]
Edge passed the decision node		1) weight=total weight of all outgoing edges must be 1 [12] 2) based on 80/20 rule [22] edge weight=4 for the true edge of the decision node edge weight=1 for the false edge of the decision node
Decision node		1) weight=2 [11, 16] 2) weight=4 [5, 15]
Merge node		1) weight=2 [11] 2) weight=3 [5, 15],
Fork-join node		1) fork weight=2, join weight=2 [5, 15] 2) fork weight=3, join weight=3 [11] 3) fork weight=5, join weight=3 [16]
Final node		1) weight=0 [15]

Table 2. Comparison of calculation methods for test path scores

Weight score	Calculation methods for test path score
Node	1) sum of all node weights, where the weights of the nodes were assigned according to the type of symbols (see Table 1) [5, 15, 16] 2) sum of all node weights, where the weight of the first node equals 1 and the weight of the next node will have 1 added until the last node [18, 23, 24] 3) sum of all node weights, where the weights of the nodes were assigned using Stack-based weight approach and the Basic IF Model [14, 17] 4) the number of nodes+sum of node weights+the number of predicate nodes+the number of logical condition nodes [13]
Edge	1) sum of all edge weights, where the weights of the edges were assigned according to the type of edges (see Table 1) [12]
Node and edge	1) sum of all node weights+sum of all edge weights, where the weights of the nodes were assigned according to the type of symbols (see Table 1) and the weights of the edges were assigned by the number of incoming dependencies of predecessor node multiplied by the number of outgoing dependencies of the successor node [11] 2) sum of all node weights+sum of all edge weights, where the weights of the nodes were assigned by using backward slices approach and the weights of the edges were assigned according to the type of edges (see Table 1) [22]

Kaur *et al*. [13] proposed the prioritization of test paths descended from UML activity diagram by using complexity of the path as follows. First, the activity diagram was transformed into a control flow graph. Second, the test paths were generated from the control flow graph using the depth first search. When it found fork nodes, it would select one representative path by using breadth first search to get unique independent paths. Third, assigning a weight to each node *IF(N)* was by the number of incoming edges to the node *FANIN(N)* multiplied by the number of outgoing edges from that node *FANOUT(N)*, using (1). Each test path score W_p was found by summing node weights W_i in the set T_p and n was the number of nodes, using (2). Fourth, the complexity C of test path was calculated using (3) by gathering the values of the number of nodes (N_p), the score of test path (W_p), the number of predicate nodes (P_p), and the number of logical conditions (C_p). And finally, the test paths would be prioritized from the highest path complexity to the lowest path complexity.

$$IF(N) = FANIN(N)xFANOUT(N) \quad (1)$$

$$W_p = \sum_{i=1}^{n} W_i \forall P \in T_p \quad (2)$$

$$C = N_p + W_p + P_p + C_p \quad (3)$$

Wang *et al*. [15] proposed test path prioritization derived from UML activity diagram as follows. First, the activity diagram was transformed into a control flow graph. Second, the test paths were generated by the depth first search. Third, the weight of each node was assigned based on the type of symbols in UML

activity diagram. If the node was an initial node or an end node, it was assigned 0. If the node was an activity node, it was assigned 4. If the node was a fork node or a join node, it was assigned 2. If the node was a merge node, it was assigned 3. Next, each test path score *f(x)* was the sum of node weights W_i on that test path and *n* was the number of nodes in the chromosome *C* in the population *P*, using (4). Finally, this research prioritized the test paths from the highest score. If two test paths had the same score, it would randomly select the test path priority order between these.

$$f(x) = \sum_{i=1}^{n} W_i C \in P \tag{4}$$

Mahali and Acharya [16] proposed the prioritization of test paths using UML activity diagram and evolutionary algorithm as follows. First, the activity diagram was transformed into an activity graph and all test paths were created. Second, each node was assigned a weight based on the symbols of UML activity diagram. If the node was a normal activity node, it was assigned 1. If the node was a decision node, it was assigned 2. If the node was an activity of one thread depends on each other, it was assigned 3. If the all activity node was within the decision symbol or the fork-join symbol, it was assigned 4. If the node was a fork node, it was assigned 5. If the node was a control node, it was assigned 6. Third, the test path score *F(x)* was the sum of node weights C_{ri} in that test path, using (5). This research prioritized the importance of test paths from the highest to the lowest path score.

$$F(x) = \sum_{i=1}^{n} C_{ri} \tag{5}$$

Jena *et al*. [22] proposed test path generation and prioritization as follows. First, the interaction overview diagram was converted into an interaction graph. The sequence diagram was transformed into message sequence dependency graph. And then, the two graphs were combined into a sequence interaction graph. Second, all possible test paths were created from the graph by using depth first search. Third, the weights were set to each node of the graph according to the number of nodes affected by the current node, determined with backward slicing. The weight of normal edges was 2. The edge from a decision node was assigned by using 80/20 rule and weight 4 was assigned to the true valued edge (80%) while weight 1 was assigned to the false valued edge (20%). Fourth, the score of a path was the sum of node weights plus the sum of edge weights along the test path, using (6). Finally, this study prioritized the test paths from the highest score to the lowest score.

$$Score(path) = \sum_{i=1}^{n} weight(node_i) + \sum_{j=1}^{n} weight(edge_i) \tag{6}$$

2. RESEARCH METHOD

This study aimed to prioritize the importance of the test paths according to the target area of interest, the changed area, and the test paths prioritized by the testers. The steps are as follows.

2.1. Define the criteria to select the UML activity diagram

The UML activity diagrams were selected according to the following criteria: the UML activity diagrams that were frequently used in research studies on software testing, the UML activity diagrams that were used in daily lives, the UML activity diagrams that were easy to understand, and the UML activity diagrams that had various control flows affecting the test path selection. The selection of UML activity diagrams depended on the control structure. Each UML activity diagram selected for this study had different control structures. As a result, the selected four UML activity diagrams for the experiment were as follows i.e., ATM Withdraw activity diagram [27], Shipping Order System activity diagram [13], Buy Beverage from Vending Machine activity diagram [11], and Mileage Purchase Web Portal activity diagram [28].

The ATM Withdraw activity diagram, applied from Boghdady *et al*. [27], comprises many types of control structures such as selection control structures consisting of one-way selection, two-way selection, and nested selection; and a one-way selection control structure within fork-join structures as shown in Figure 1. The Shipping Order System activity diagram, applied from Kaur *et al*. [13], comprises a fork-join structure with no control structure inside, selection control structure consisting of two-way selection, but there is no iteration control structure as shown in Figure 2. The Vending Machine activity diagram, applied from Sapna and Mohanty [11], which comprises two iteration control structures with pre-test loop. This diagram does not

have the fork-join structure as shown in Figure 3. The Mileage Purchase activity diagram, applied from Paiboonkasemsut and Limpiyakorn [28], comprises selection control structures consisting of two-way selection and nested selection, and there is a fork-join structure. This diagram does not have an iteration control structure as shown in Figure 4.

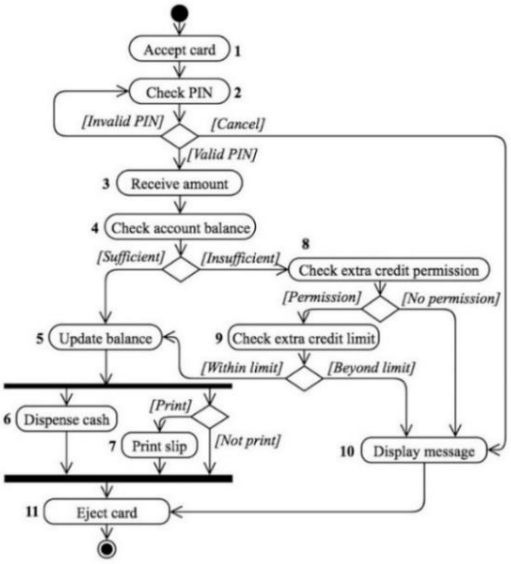


Figure 1. ATM withdraw activity diagram

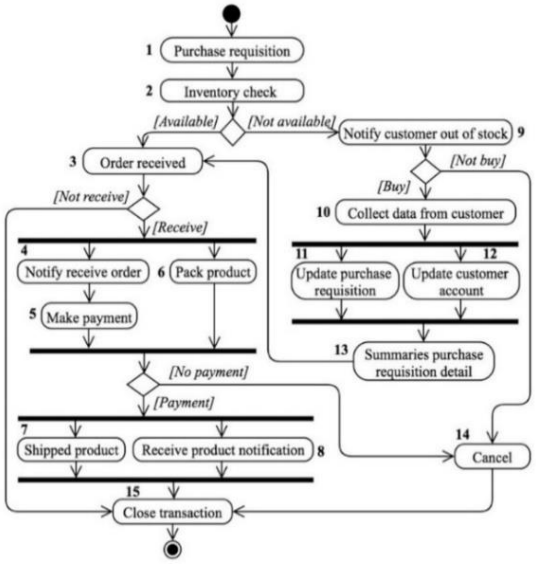


Figure 2. Shipping order system activity diagram

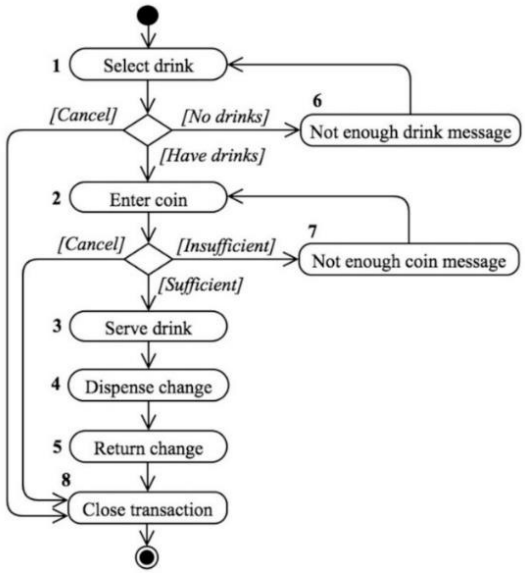


Figure 3. Vending machine activity diagram

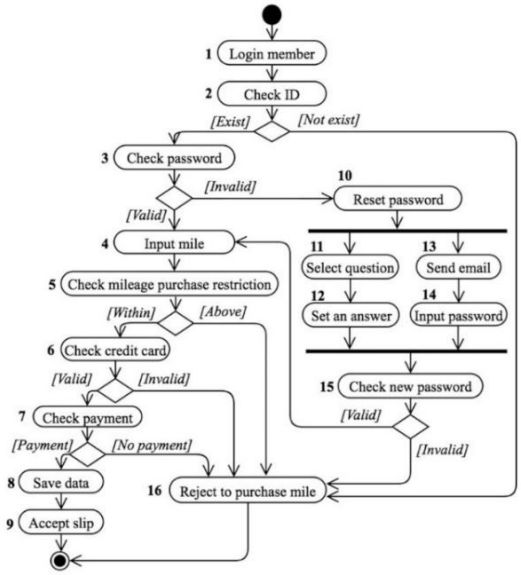


Figure 4. Mileage purchase activity diagram

2.2. Transform the UML activity diagram to an activity flow graph

The four activity diagrams were transformed to an activity flow graph. For the concurrence region, a questionnaire-based survey for test path by testers' interest, 65 professional software testers show that the path must be traversed from left-to-right. Each thread is listed from the activity of top-level to activity of low level according to the depth first search method.

The ATM Withdraw activity diagram was transformed into an activity flow graph, which is a directed graph. Each node represents the symbols of activity diagram and each edge represents the flow in the activity diagram. The activity flow graph of ATM Withdraw activity diagram is depicted in Figure 5,

where *I* is the initial node, *F* is the final node, *di* is a decision node, *fi* is a fork node, and *ji* is a join node. The Shipping Order System activity diagram, the Vending Machine activity diagram, and the Mileage Purchase activity diagram were transformed into an activity flow graph, depicted in Figures 6-8 respectively.

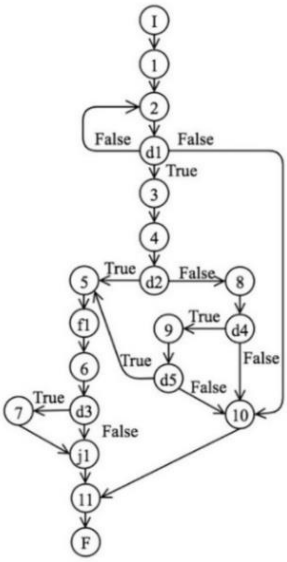


Figure 5. ATM withdraw activity flow graph

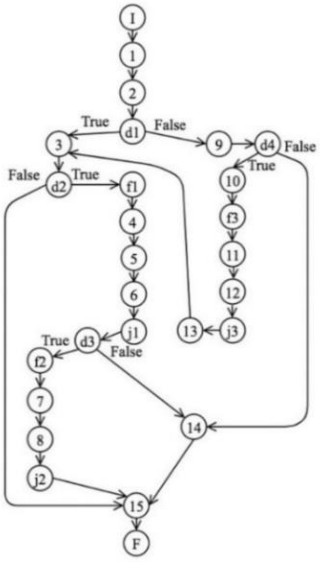


Figure 6. Shipping order system activity flow graph

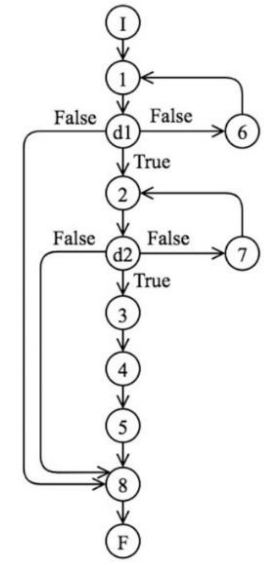


Figure 7. Vending machine activity flow graph

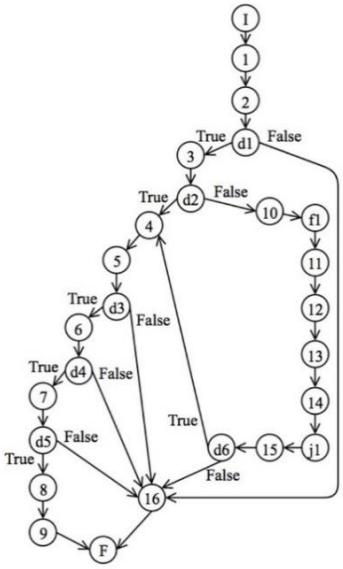


Figure 8. Mileage purchase activity flow graph

2.3. Questionnaire-based survey for test path prioritization by testers' interest

The four activity diagrams in section 2.1 were used to generate test paths in each activity diagram. Then, the test paths were given to 65 professional software testers to prioritize them by importance according to their interests to create a baseline for comparisons. The data obtained were analyzed for prioritization frequencies of the test paths. The survey results on the prioritization of the test paths in ATM withdraw, Shipping order system, Vending machine, and Mileage purchase activity diagram according to the testers' interests are summarized in Tables 3-6 respectively. In Table 3, the ATM withdraw activity diagram has generated 14 test paths (P1 to P14). All 65 testers will assign a path (row) to rank the importance of the test according to their interest by arranging them in order 1-14 (columns). The highest value depicted in

bold-faced in each column shows that the testers have the most agreement in that order. For example, most testers agree that P5 should test in the third rank of the test paths.

Table 3. The results of the survey on prioritization of the test paths in ATM withdraw activity diagram according to the testers' interests

Test path no.	Test path	Rank of test path	The frequency distribution of ranking													
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
P1	I-1-2-d1-3-4-d2-5-f1-6-d3-7-j1-11-F	1	**32**	9	4	1	6	1	1	3	0	3	0	1	1	3
P2	I-1-2-d1-2-d1-3-4-d2-5-f1-6-d3-7-j1-11-F	7	6	4	7	8	5	2	**18**	8	2	2	2	1	0	0
P3	I-1-2-d1-3-4-d2-5-f1-6-d3-j1-11-F	2	7	**19**	7	3	2	7	4	4	1	2	6	0	3	0
P4	I-1-2-d1-2-d1-3-4-d2-5-f1-6-d3-j1-11-F	8	0	3	5	8	2	5	9	**16**	5	4	2	3	2	1
P5	I-1-2-d1-3-4-d2-8-d4-9-d5-5-f1-6-d3-7-j1-11-F	3	4	3	**18**	9	8	3	5	2	6	3	1	2	1	0
P6	I-1-2-d1-2-d1-3-4-d2-8-d4-9-d5-5-f1-6-d3-7-j1-11-F	9	6	0	5	3	2	7	3	5	**16**	11	2	3	1	1
P7	I-1-2-d1-3-4-d2-8-d4-9-d5-5-f1-6-d3-j1-11-F	4	0	2	6	**18**	3	5	9	8	3	2	4	1	3	1
P8	I-1-2-d1-2-d1-3-4-d2-8-d4-9-d5-5-f1-6-d3-j1-11-F	10	1	6	1	1	4	5	4	9	6	**16**	6	2	0	4
P9	I-1-2-d1-3-4-d2-8-d4-9-d5-10-11-F	5	0	1	2	0	**19**	7	6	3	9	5	5	6	1	1
P10	I-1-2-d1-2-d1-3-4-d2-8-d4-9-d5-10-11-F	11	1	1	3	3	3	1	2	3	7	8	**21**	6	6	0
P11	I-1-2-d1-3-4-d2-8-d4-10-11-F	6	0	0	1	7	3	**16**	1	4	6	5	7	10	4	1
P12	I-1-2-d1-2-d1-3-4-d2-8-d4-10-11-F	12	1	0	6	2	4	0	3	0	1	2	8	**26**	5	7
P13	I-1-2-d1-10-11-F	13	8	6	2	0	3	1	3	0	0	1	1	1	**30**	9
P14	I-1-2-d1-2-d1-10-11-F	14	4	6	3	1	3	2	0	0	1	0	3	0	6	**36**

Table 4. The results of the survey on prioritization of the test paths in shipping order system activity diagram according to the testers' interests

Test path no.	Test path	Rank of test path	The frequency distribution of ranking						
			1	2	3	4	5	6	7
P1	I-1-2-d1-3-d2-15-F	6	8	10	7	3	1	**23**	13
P2	I-1-2-d1-3-d2-f1-4-5-6-j1-d3-f2-7-8-j2-15-F	1	**30**	7	12	6	4	3	3
P3	I-1-2-d1-3-d2-f1-4-5-6-j1-d3-14-15-F	3	1	10	**27**	10	10	5	2
P4	I-1-2-d1-9-d4-10-f3-11-12-j3-13-3-d2-15-F	5	5	3	11	7	**28**	10	1
P5	I-1-2-d1-9-d4-10-f3-11-12-j3-13-3-d2-f1-4-5-6-j1-d3-f2-7-8-j2-15-F	2	11	**24**	2	9	12	4	3
P6	I-1-2-d1-9-d4-10-f3-11-12-j3-13-3-d2-f1-4-5-6-j1-d3-14-15-F	4	1	7	11	**23**	7	7	9
P7	I-1-2-d1-9-d4-14-15-F	7	12	4	2	1	2	11	**33**

Table 5. The results of the survey on prioritization of the test paths in vending machine activity diagram according to the testers' interests

Test path no.	Test path	Rank of test path	The frequency distribution of ranking									
			1	2	3	4	5	6	7	8	9	10
P1	I-1-d1-8-F	10	10	1	3	1	2	1	2	0	2	**43**
P2	I-1-d1-2-d2-8-F	9	1	12	1	6	3	4	2	4	**29**	3
P3	I-1-d1-2-d2-3-4-5-8-F	**1**	**34**	5	6	3	4	5	3	2	1	2
P4	I-1-d1-6-1-d1-8-F	8	1	5	5	2	8	4	1	**29**	9	1
P5	I-1-d1-6-1-d1-2-d2-8-F	7	0	1	2	6	3	10	**28**	10	4	1
P6	I-1-d1-2-d2-7-2-d2-8-F	5	0	5	10	7	**20**	7	7	3	6	0
P7	I-1-d1-6-1-d1-2-d2-7-2-d2-8-F	6	0	2	1	9	9	**21**	10	6	4	3
P8	I-1-d1-6-1-d1-2-d2-3-4-5-8-F	3	2	5	**24**	7	7	5	7	7	1	0
P9	I-1-d1-2-d2-7-2-d2-3-4-5-8-F	2	9	**25**	10	3	5	1	1	1	6	4
P10	I-1-d1-6-1-d1-2-d2-7-2-d2-3-4-5-8-F	4	9	6	2	**20**	4	9	3	2	2	8

Table 6. The results of the survey on prioritization of the test paths in mileage purchase activity diagram according to the testers' interests

Test path no.	Test path	Rank of test path	The frequency distribution of ranking									
			1	2	3	4	5	6	7	8	9	10
P1	I-1-2-d1-16-F	10	13	2	1	1	1	1	0	1	1	**44**
P2	I-1-2-d1-3-d2-10-f1-11-12-13-14-j1-15-d6-4-5-d3-6-d4-7-d5-8-9-F	5	12	4	5	3	**28**	3	1	0	6	3
P3	I-1-2-d1-3-d2-10-f1-11-12-13-14-j1-15-d6-4-5-d3-6-d4-7-d5-16-F	6	3	8	4	5	6	**28**	4	2	2	3
P4	I-1-2-d1-3-d2-10-f1-11-12-13-14-j1-15-d6-4-5-d3-6-d4-16-F	7	0	2	9	8	1	8	**28**	6	3	0
P5	I-1-2-d1-3-d2-10-f1-11-12-13-14-j1-15-d6-4-5-d3-16-F	8	0	3	7	8	4	3	6	**32**	2	0
P6	I-1-2-d1-3-d2-10-f1-11-12-13-14-j1-15-d6-16-F	9	1	6	2	2	4	5	4	2	**35**	4
P7	I-1-2-d1-3-d2-4-5-d3-6-d4-7-d5-8-9-F	1	**39**	4	1	1	5	3	2	3	1	6
P8	I-1-2-d1-3-d2-4-5-d3-6-d4-7-d5-16-F	2	1	**28**	7	6	3	9	4	3	3	1
P9	I-1-2-d1-3-d2-4-5-d3-6-d4-16-F	3	1	0	**28**	5	8	4	9	7	3	0
P10	I-1-2-d1-3-d2-4-5-d3-16-F	4	1	7	1	**25**	3	4	5	10	7	2

2.4. Propose approaches to test path prioritization

2.4. Propose approaches to test path prioritization

The weight assignments in the activity flow graph transformed from the UML activity diagram are the topic of this section and then the scores of the test paths are calculated. The activity flow graph consists of nodes and edges as shown in Figures 5, 6, 7, and 8. The nodes and edges have assigned weights as follows.

2.4.1. Define weights of the nodes in the activity flow graph

Initial node and final node presented the flow start and the final step are assigned 0. Merge, fork, and join node which are the control node that deputed flow of the program are assigned 0. Decision node is assigned 0, we use several methods to consider the edge weights for the decision node instead. Activity node contained the program statement is assigned 1.

2.4.2. Define weights of the edges in the activity flow graph

Normal edge is assigned 1. Edge of decision node is based on the work breakdown structure. It can have more than 2 edges but the total weight of True value and false value is 100%. The True edge is assigned more weight than the false edge. In this research, the weight of an edge is based on either time management method, Pareto method, Buffett method, Binary method, or Bipolar method as shown in Table 7. The total weight of edges of a decision node is 5. The Time management, a 70/30 rule is applied to assign weights to edges from decision node [29]. For 70% the True edge weight assigned is 3.5, and 30% the false edge weight assigned is 1.5. The Pareto, an 80/20 rule is applied to assign weights to edges from the decision node [22, 29]. For 80% the True edge weight assigned is 4 and 20% the false edge weight assigned is 1. The Buffett, a 90/10 rule is applied to assign weights to edges from a decision node [29]. For 90% the True edge weight assigned is 4.5, and 10% the false edge weight assigned is 0.5. The Binary, a 100/0 rule is applied to assign weights to edges from the decision node. For 100% the True edge weight assigned is 5 and 0% the false edge weight assigned is 0. The Bipolar, a 100/-100 rule is applied to assign weights to edges from the decision node. For 100% the True edge weight assigned is 5 and -100% the false edge weight assigned is -5.

Table 7. Define weights of the edges from a decision node

Edge	Weight assignment methods				
	Time management (70/30 rule)	Pareto (80/20 rule)	Buffett (90/10 rule)	Binary (100/0 rule)	Bipolar (100/-100 rule)
True	3.5	4.0	4.5	5.0	5.0
False	1.5	1.0	0.5	0.0	-5.0

2.4.3. Calculating scores and prioritizing for test paths

The test paths from each activity diagram were given scores $Score_p$ to prioritize their importance. W_e is an edge weight in the test path and E is the number of edges in the test path. W_n is the weight of the test path (sum of node weights in test path) and N is the number of nodes in the test path. The score of a test path $Score_p$ is calculated by (7), and the test paths are prioritized according to their scores in descending order.

$$Score_p = \left(\frac{\sum_{e=1}^{E} W_e}{\sum_{n=1}^{N} W_n} \right) \quad (7)$$

2.5. Comparing computed results on prioritization of test paths with the survey results on the testers' prioritization of test paths

The objective of the proposed approach is to rank the test paths the same as the testers' interests. In this section, the difference and the similarity between test path prioritization by the proposed methods and the subjective test path prioritization according to the testers' interests will be evaluated.

2.5.1. The calculation of the difference to subjective prioritization of the test paths

The difference *Diff(T,R)* between test path rank by the proposed methods *(R)* and the subjective test path prioritization according to the testers' interests *(T)* is quantified in this section. The difference between test path prioritization can be calculated by (8) where P is the number of test paths in activity flow graph. The value of difference *Diff(T,R)* is between 0 and 1, the ideal value for *Diff(T,R)* is 0. D_p is the difference of a T-R pair of the test path. T_p is the test path rank according to the testers' interests, and R_p is test path rank

by a proposed algorithmic method. If R_p is equal to T_p, this means that the proposed method can match the test path rank to the testers' interests, the difference of the test path is 0. If R_p is more than T_p, this means that rank by the proposed method is too late for testing. The difference of the test path is punished as the double of the difference of their ranks. If R_p is less than T_p, the difference of the test path calculated as the difference of their ranks. *MaxDiff* is the maximum sum of the difference of T-R pair of all paths.

$$Diff(T,R) = \left(\sum_{p=1}^{P} D_p \right) / MaxDiff \,, \tag{8}$$

where

$$D_p = \begin{cases} 2|T_p - R_p|, & R_p > T_p \\ 0, & R_p = T_p \\ |T_p - R_p|, & R_p < T_p \end{cases}, \quad MaxDiff = \begin{cases} 3\left(\frac{P}{2}\right)^2, & P \quad is \quad EvenNumber \\ 3\left(\frac{P^2-1}{4}\right), & P \quad is \quad OddNumber \end{cases}.$$

2.5.2. The calculation of the similarity to subjective prioritization of the test paths

The similarity *Sim(T,R)* between test path rank by the proposed methods *(R)* and the subjective test path prioritization according to the testers' interests *(T)* is quantified in this section. The similarity between test path prioritization can be calculated by (9), where P is the number of test paths in activity flow graph. The value of similarity *Sim(T,R)* is between 0 and 1, the ideal value for *Sim(T,R)* is 1. S_p is the similarity of a T-R pair of the test path. T_p is the test path rank according to the testers' interests, and R_p is test path rank by a proposed algorithmic method. If R_p is less than T_p, this means that rank by the proposed method is not too late for testing, the similarity of the test path is 1. If R_p is equal to T_p, this means that the proposed method can match the test path rank to the testers' interests, the similarity of the test path is doubly rewarded. If R_p is more than T_p, this means that rank by the proposed method is too late for testing, the similarity of the test path is 0. *MaxSim* is the maximum sum of the similarity of T-R pair of all paths.

$$Sim(T,R) = \left(\sum_{p=1}^{P} S_p \right) / MaxSim, \tag{9}$$

where

$$S_p = \begin{cases} 1, & T_p > R_p \\ 2, & T_p = R_p \\ 0, & T_p < R_p \end{cases}, \quad MaxSim = 2P \,.$$

3. RESULTS AND DISCUSSIONS

The study assessed the proposed weight assignment methods i.e. node weights and edge weights applying with Time management method, Pareto method, Buffett method, Binary method, and Bipolar method. In the experiment, the test paths from four UML activity diagrams (as described in section 2.3) were prioritized and compared with prior researches.

3.1. The results of test path prioritization

The prioritizations of test paths from the activity flow graphs were obtained by assigning weights using five proposed methods: calculating the scores for the test paths by (7), calculating the difference and the similarity to subjective prioritization of the test paths according to the testers' interests by (8) and (9), respectively. If two test paths had the same score, it would assign the same rank value and the next number(s) will be skipped. The results of test path prioritization for ATM Withdraw activity flow graph are shown in Table 8, for Shipping Order System activity flow graph are shown in Table 9, for Vending Machine activity flow graph are shown in Table 10, and for Mileage Purchase activity flow graph are shown in Table 11. In all four tables, "Early" means the number of test paths that rank by the proposed method is earlier than rank by the tester. "Late" means the number of test paths that rank by the proposed method is later than rank by the tester. "Equal" means the proposed method can match the test path rank to the testers' interests.

Table 8. The results of test path prioritization for ATM withdraw activity flow graph

Test path no.	Test path rank by proposed weight assignment methods					Test path rank by tester
	Time management	Pareto	Buffett	Binary	Bipolar	
P1	5	4	2	2	1	1
P2	6	6	5	4	3	7
P3	7	7	7	6	5	2
P4	8	8	8	8	7	8
P5	1	1	1	1	2	3
P6	3	2	3	3	4	9
P7	2	3	4	4	6	4
P8	4	5	6	7	8	10
P9	10	9	9	9	9	5
P10	9	10	10	10	10	11
P11	12	11	11	11	11	6
P12	11	11	12	12	13	12
P13	14	14	14	13	12	13
P14	13	13	13	14	14	14
Early	8	8	6	5	7	
Equal	1	1	3	5	2	
Late	5	5	5	4	5	
Difference	0.43	0.37	0.33	0.29	0.31	
Similarity	0.36	0.36	0.43	0.54	0.39	

Table 9. The results of test path prioritization for shipping order system activity flow graph

Test path no.	Test path rank by proposed weight assignment methods					Test path rank by tester
	Time management	Pareto	Buffett	Binary	Bipolar	
P1	2	2	3	4	5	6
P2	1	1	1	1	1	1
P3	3	2	2	2	3	3
P4	6	6	6	6	6	5
P5	4	4	3	3	2	2
P6	5	5	5	5	4	4
P7	7	7	7	7	7	7
Early	1	2	2	2	1	
Equal	3	2	2	2	5	
Late	3	3	3	3	1	
Difference	0.33	0.36	0.28	0.25	0.08	
Similarity	0.50	0.43	0.43	0.43	0.79	

Table 10. The results of test path prioritization for vending machine activity flow graph

Test path no.	Test path rank by proposed weight assignment methods					Test path rank by tester
	Time management	Pareto	Buffett	Binary	Bipolar	
P1	6	9	9	9	9	10
P2	1	1	1	1	4	9
P3	4	2	2	2	1	1
P4	10	10	10	10	10	8
P5	2	3	5	5	6	7
P6	2	3	5	5	6	5
P7	5	7	8	8	8	6
P8	7	5	3	3	2	3
P9	7	5	3	3	2	2
P10	9	8	7	7	5	4
Early	5	4	3	3	4	
Equal	0	0	2	2	2	
Late	5	6	5	5	4	
Difference	0.79	0.55	0.39	0.39	0.27	
Similarity	0.25	0.20	0.35	0.35	0.40	

Table 11. The results of test path prioritization for mileage purchase activity flow graph

Test path no.	Test path rank by proposed weight assignment methods					Test path rank by tester
	Time management	Pareto	Buffett	Binary	Bipolar	
P1	10	10	10	10	10	10
P2	5	5	5	4	4	5
P3	6	6	6	6	6	6
P4	7	7	7	7	7	7
P5	8	8	8	8	8	8
P6	9	9	9	9	9	9
P7	2	1	1	1	1	1
P8	1	2	2	2	2	2
P9	3	3	3	3	3	3
P10	4	4	4	5	5	4
Early	1	0	0	1	1	
Equal	8	10	10	8	8	
Late	1	0	0	1	1	
Difference	0.04	0.00	0.00	0.04	0.04	
Similarity	0.85	1.00	1.00	0.85	0.85	

3.2. Comparison of test path prioritization by the proposed algorithmic method and testers' interests

On the evaluation of our proposed methods, the average of the difference and similarity values of all proposed methods were compared with prior published studies i.e., Kaur *et al*. [13], Wang *et al*. [15], Mahali and Acharya [16], and Jena *et al*. [22] with weight assignment as described in Table 1 and test path score calculation as described in Table 2.

3.2.1. Comparison of the difference to subjective prioritization of the test paths

On comparing five proposed algorithmic method with prior published studies, the difference values of all proposed methods were less (better) than those in the prior published studies for ATM Withdraw activity diagram, Shipping Order System activity diagram, and Mileage Purchase activity diagram as shown in Table 12. The Bipolar method had the least average of the difference value. Moreover, in comparing every average difference score of the proposed methods with the results from prior published studies, they performed better.

Table 12. The difference values in test path prioritization

Activity diagram	Difference values					Prior published			
	Time management	Pareto	Buffett	Binary	Bipolar	[13]	[15]	[16]	[22]
1. ATM Withdraw	0.43	0.37	0.33	0.29	0.31	0.59	0.56	0.54	0.61
2. Shipping Order	0.33	0.36	0.28	0.25	0.08	0.42	0.42	0.42	0.50
3. Vending Machine	0.79	0.55	0.39	0.39	0.27	0.35	0.31	0.25	0.40
4. Mileage Purchase	0.04	0.00	0.00	0.04	0.04	0.80	0.72	0.80	0.80
Average difference	0.40	0.32	0.25	0.24	**0.18**	0.54	0.50	0.50	0.58

3.2.2. Comparison of the similarity to subjective prioritization of the test paths

The experimental results show the similarity values of all proposed methods were more (better) than those in the prior published studies for ATM Withdraw activity diagram, shipping order system activity diagram, and Mileage Purchase activity diagram as shown in Table 13. The Bipolar method had the most average of the similarity value. Moreover, in comparing every average similarity score of the proposed methods with the results from prior published studies, they performed better.

Table 13. The similarity values in test path prioritization

Activity diagram	Similarity values					Prior published			
	Time management	Pareto	Buffett	Binary	Bipolar	[13]	[15]	[16]	[22]
1. ATM Withdraw	0.36	0.36	0.43	0.54	0.39	0.39	0.39	0.39	0.32
2. Shipping Order	0.50	0.43	0.43	0.43	0.79	0.29	0.29	0.29	0.29
3. Vending Machine	0.25	0.20	0.35	0.35	0.40	0.50	0.65	0.60	0.50
4. Mileage Purchase	0.85	1.00	1.00	0.85	0.85	0.35	0.35	0.35	0.35
Average similarity	0.49	0.50	0.55	0.54	**0.61**	0.38	0.42	0.41	0.37

4. TARGET-BASED TEST PATH PRIORITIZATION

The enhancement of our proposed methods was the test path prioritization with the target area or changed area. To prioritize by the importance of the target area or of the changed area, the test paths for which the testers focused on the target area were prioritized by adding weight to those paths. As shown in (10), *WT* is the target weight. In case that the path composes of the target areas specified by the tester, *WT* is the maximum length of all test paths otherwise the *WT* value is 0. For example, now consider the test paths in the ATM Withdraw activity flow graph as shown in Figure 5, the testers were interested in the target area of node 9, so the test paths that passed through node 9 (P5, P6, P7, P8, P9, P10) were prioritized higher than the other test paths, as shown in Table 14.

$$Score_p = \left(\frac{\sum_{e=1}^{E} W_e}{\sum_{n=1}^{N} W_n} \right) + WT \quad (10)$$

Table 14. Prioritization of test paths by the importance of target area in ATM withdraw activity flow graph

Test path no.	Test path rank by proposed weight assignment methods									
	Time management		Pareto		Buffett		Binary		Bipolar	
	w/o target	w/ target	w/o target	w/ target	w/o target	w/ target	w/o target	w/ target	w/o target	w/ target
P1	5	7	4	7	2	7	2	7	1	7
P2	6	8	6	8	5	8	4	8	3	8
P3	7	9	7	9	7	9	6	9	5	9
P4	8	10	8	10	8	10	8	10	7	10
P5	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**1**
P6	**3**	**3**	**2**	**2**	**3**	**2**	**3**	**2**	**4**	**2**
P7	**2**	**2**	**3**	**3**	**4**	**3**	**4**	**3**	**6**	**3**
P8	**4**	**4**	**5**	**4**	**6**	**4**	**7**	**4**	**8**	**4**
P9	**10**	**6**	**9**	**5**	**9**	**5**	**9**	**5**	**9**	**5**
P10	**9**	**5**	**10**	**6**	**10**	**6**	**10**	**6**	**10**	**6**
P11	12	12	11	11	11	11	11	11	11	11
P12	11	11	11	11	12	12	12	12	13	13
P13	14	14	14	14	14	14	13	13	12	12
P14	13	13	13	13	13	13	14	14	14	14

Note: w/o stands for "without" and w/ stands for "with".

5. CONCLUSION

This paper proposed algorithmic test path prioritizations demonstrated with four UML activity diagrams. The activity diagram is selected according to criteria which are used in daily lives, easy to understand, had various control flows, and frequently used in prior published studies on software testing. The selection of activity diagrams depended on the control structure. Each activity diagram selected for this study had three different control structures which are selection control structures, iteration control structures, and fork-join structures. The four activity diagrams are converted to an activity flow graph. Then, an activity flow graph is used to generate test paths. 65 professional software testers prioritize test paths by importance according to their interests to create a baseline for comparisons. In this research, we define the weights of the node and edge in the activity flow graph. Initial node, final node, merge node, fork node, and join node are assigned 0. Activity node is assigned 1. Decision node is assigned 0, we define the edge weights for the decision node instead. Time management method, Pareto method, Buffett method, Binary method, and Bipolar method were applied to assign weights to edges of the decision node. Normal edge is assigned 1. We proposed the test path prioritization method to generate a sequence of test paths that would match the testers' interests.

The experimental results show that to rank all possible paths, the difference and the similarity to the baseline prioritization were compared. It was found that test path prioritization with our node weight assignment and edge weight assignment applied from Bipolar method gave the least average difference and the most average similarity from the baseline. The result of enhancement of test path prioritization with the target area or changed area shows that if the testers were interested in the target area of any activity in the activity diagram then the test paths that passed through those activities were prioritized higher than the other test paths. In the future, we will apply optimization techniques i.e., artificial bee colony algorithm with our proposed node weight assignment and edge weight assignment applied from Bipolar method to rank test paths based on important objectives, especially target area.

REFERENCES

[1] M. Khatibsyarbini, et al., "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology*, vol. 93, pp. 74-93, 2018.

[2] A. Bajaj and O. P. Sangwan, "A Systematic Literature Review of Test Case Prioritization Using Genetic Algorithms," *IEEE Access*, vol. 7, pp. 126355-126375, 2019.

[3] R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization," *Journal of King Saud University-Computer and Information Sciences*, 2018.

[4] R. K. Swain, et al., "Generation of Test Case Using Activity Diagram," *International Journal of Computer Science and Informatics*, vol. 3, no. 2, pp. 1-10, 2013.

[5] A. K. Jena, et al., "A Novel Approach for Test Case Generation from UML Activity Diagram," *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pp. 621-629, 2014.

[6] P. Achimugu, et al., "A systematic literature review of software requirements prioritization research," *Information and Software Technology*, vol. 56, no. 6, pp. 568-585, 2014.

[7] Palak and P. Gulia, "Hybrid swarm and GA based approach for software test case selection," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 6, pp. 4898-4903, 2019.

[8] S. Alani, et al., "A hybrid technique for single-source shortest path-based on A* algorithm and ant colony optimization," *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 9, no. 2, pp. 256-263, 2020.

[9] J. Chen, et al., "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering," *Journal of Systems and Software*, vol. 135, pp. 107-125, 2018.

[10] U. Farooq, et al., "A review of object-oriented approach for test case prioritization," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 16, no. 1, pp. 429-434, 2019.

[11] P. G. Sapna and H. Mohanty, "Prioritization of Scenarios Based on UML Activity Diagrams," *2009 First International Conference Computational Intelligence, Communication Systems and Networks*, pp. 271-276, 2009.

[12] A. Gantait, "Test Case Generation and Prioritization from UML Models," *2011 Second International Conference on Emerging Applications of Information Technology*, pp. 345-350, 2011.

[13] P. Kaur, R. Sibal, and P. Bansal, "Prioritization of Test Scenarios Derived from UML Activity Diagram Using Path Complexity," *CUBE'12: Proceedings of the CUBE International Information Technology Conference*, pp. 355-359, 2012.

[14] S. Dalal and R. S. Chhillar, "A Novel Technique for Generation of Test Cases Based on Bee Colony Optimization and Modified Genetic Algorithm (BCO-mGA)," *International Journal of Computer Applications*, vol. 68, no. 19, pp. 0975-8887, 2013.

[15] X. Wang, X. Jiang and H. Shi, "Prioritization of Test Scenarios using Hybrid Genetic Algorithm Based on UML Activity Diagram," *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 854-857, 2015.

[16] P. Mahali and A. A. Acharya, "Model Based Test Case Prioritization Using UML Activity Diagram and Evolutioinary Algorithm," *International Journal of Computer Science and Informatics*, vol. 3, no. 2, pp. 42-47, 2013.

[17] F. M. Nejad, R. Akbari and M. M. Dejam, "Using Memetic Algorithms for Test Case Prioritization in Model Based Software Testing," *2016 1st Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, pp. 142-147, 2016.

[18] V. M. Sumalatha and G. S. V. P. Raju, "Object Oriented Test Case Generation Technique using Genetic Algorithms," *International Journal of Computer Applications*, vol. 61, no. 20, pp. 20-26, 2013.

[19] D. K. Yadav and S. Dutta, "Regression test case selection and prioritization for object oriented software," *Microsystem Technologies*, pp. 1-15, 2019.

[20] N. Panda, et al., "Test scenario prioritization for object-oriented systems using UML diagram," *International Journal of System Assurance Engineering and Management*, vol. 10, no. 3, pp. 316-325, 2019.

[21] I. Hooda and R. S. Chhillar, "Test Case Optimization and Redundancy Reduction Using GA and Neural Networks," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 6, pp. 5449-5456, 2018.

[22] A. K. Jena, et al., "Test Case Generation and Prioritization Based on UML Behavioral Models," *Journal of Theoretical and Applied Information Technology*, vol. 78, no. 3, pp. 336-352, 2015.

[23] N. Khurana and R. S. Chillar, "Test Case Generation and Optimization using UML Models and Genetic Algorithm," *Procedia Computer Science*, vol. 57, pp. 996-1004, 2015.

[24] N. Khurana, et al., "A Novel Technique for Generation and Optimization of Test Cases Using Use Case, Sequence, Activity Diagram and Genetic Algorithm," *Journal of Software*, vol. 11, no. 3, pp. 242-250, 2016.

[25] P. C. Jorgensen, "Software Testing a Craftsman's Approach," *4th ed. Boca Raton, FL, CRC Press*, 2014.

[26] M. Sahak, et al., "Evaluation of Software Product Line Test Case Prioritization Techniques," *International Journal on Advanced Science Engineering Information Technology*, vol. 7, no. 4-2, pp. 1601-1608, 2017.

[27] P. N. Boghdady, et al., "An Enhanced Test Case Generation Technique Based on Activity Diagrams," *The 2011 International Conference on Computer Engineering and Systems*, pp. 289-294, 2011.

[28] P. Paiboonkasemsut and Y. Limpiyakorn, "Reliability Tests for Process Flow with Fault Tree Analysis," *2015 2nd International Conference on Information Science and Security (ICISS)*, pp. 1-4, 2015.

[29] R. S. Pressman, "Software Engineering: A Practitioner's Approach," *7th ed. NY, McGraw-Hill*, 2010.

BIOGRAPHIES OF AUTHORS

Walaiporn Sornkliang received the B.Ba. degree in Computer Business from Sripatum University, Bangkok, Thailand in 1994. She also received the M.Sc. degree in Computer Science from Mahidol University, Bangkok, Thailand in 2002. She is currently pursuing the Ph.D. degree in Management of Information Technology at School of Informatics, Walailak University, Nakhon Si Thammarat, Thailand. Her current research interests include software engineering and software testing.

Thimaporn Phetkaew received her B.Sc. degree in Applied Mathematics and she also received her M.Sc. degree in Computer Science from Prince of Songkla University, Thailand, in 1997 and 2000, respectively. In 2004, she received her Ph.D. degree in Computer Engineering from Chulalongkorn University, Thailand. She is currently working as Assistant Professor in School of Informatics, Walailak University. She is a member of Center of Excellence in Informatics Innovation at Walailak University. She is also the reviewer of many International Conferences and International Journal. Her research interests include Software Engineering, Software Testing, Data Mining, and Machine Learning.