JSCDM

http://penerbit.uthm.edu.my/ojs/index.php/jscdm

e-ISSN : 2716-621X

Journal of Soft Computing and Data Mining

Test Case Prioritization Using Swarm Intelligence Algorithm to Improve Fault Detection and Time for Web Application

Kohani K. Mohan[1], Nurezayana Zainal[1*], Mohd Zanes Sahid[1]

[1]Faculty of Computer Science and Information Technology,
Universiti Tun Hussein Onn Malaysia, Parit Raja, 86400 Batu Pahat Johor, MALAYISA

*Corresponding Author

DOI: https://doi.org/10.30880/jscdm.2023.04.02.006
Received 07 May 2023; Accepted 13 July 2023; Available online 04 October 2023

Abstract: Prioritizing test cases based on several parameters where important ones are executed first is known as test case prioritization (TCP). Code coverage, functionality, and features are all possible factors of TCP for detecting bugs in software as early as possible. This research was carried out to test and compare the effectiveness Swarm Intelligence algorithms, where Artificial Bee Colony (ABC) and Ant Colony Optimization (ACO) algorithms were implemented to find the fault detected and execution time as these are the curial aspects in software testing to ensure good quality products are produced within the timeline. As web applications are commonly used by a board population, this research was carried out on an Online Shopping application represented as Case Study One and Education Administrative application known as Case Study Two. In recent years, TCP has been implemented widely, but none has implemented on web application which was conducted to fill the gaps and produce a new contribution in this area. The outcome was compared using Average Percentage Fault Detected (APFD) and execution time. For Case Study One, the APFD value was 0.80 and 0.71 while the execution time was 8.64 seconds and 0.69 seconds respectively for ABC and ACO. For Case Study Two, the APFD values were 0.81 and 0.64 while the execution time was 8.83 seconds and 1.22 seconds for ABC and ACO. It was seen that both algorithms performed well in their respective ways. ABC had shown to give a higher value for APFD while ACO had converged faster for execution time.

Keywords: Test case prioritization, artificial bee colony, Ant Colony Optimization (ACO), APFD, execution time

1. Introduction

Regression testing is portrayed as the technique engaged with retesting the adjusted code of the system in test and ensuring that no new slip-ups have been brought into after the modification of recent codes. The techniques in regression testing includes regression test selection, test suite minimization and test case prioritization. Test Case Prioritization (TCP) is a technique that assigns a priority to each test case. Priority is assigned based on a set of criteria, and the highest-priority test cases are scheduled first. The criterion might be that the test case with the fastest fault coverage receives the highest priority. This method has the advantage of neither discarding or permanently removing test cases from the test suite [1]. TCP is aimed to schedule the execution order of test cases to improve test effectiveness. The most common problem that TCP is used to solve is to minimize costs and the time taken during testing process.

Recently, several researches performed a study on test case prioritization by implementing GA, FA, Bat Algorithm (BA), etc. The study outcome showed various results based on the domains used as well as the algorithms. The FA showed a good response however, it slightly outperformed in terms of execution time [2]. Besides that, the research that was carried out to test the effectiveness between GA and ACO showed a positive outcome that ACO was a better

*Corresponding author: nurezayana@uthm.edu.my

algorithm when implemented in time-constrained environment [3]. Hence, this research was carried out in order to test and compare the effectiveness of two different algorithms under Swarm Intelligence, where ABC and ACO algorithms were implemented in the study. The test was executed to find the fault detected and execution time when both algorithms were implemented as these are the curial aspects in software testing to ensure good quality products are produced within the timeline.

Besides that, this research was also carried out on web application test cases. As web applications or dynamic websites are commonly used by the broad population around the world for information, e-commerce, and e-learning, this research was conducted to test the effectiveness of both algorithms in prioritizing the web application test cases [4]. This would help to improve the fault detected and execution time as these are the most crucial aspects in software testing to maintain good quality assurance within a proposed period of time. In recent years, TCP has been implemented in many studies but none has implemented on web application which was conducted in this research to fill the gaps and produce a new contribution in this area.

2. Literature Review

2.1 Software Testing

Software testing technologies have evolved as a prominent software engineering technique in recent years, assisting in cost management, quality improvement, time and risk reduction [5]. Software testers have had to come up with new techniques to estimate their projects as testing practices have evolved [6]. Software testing is a type of quality assurance activity that involves assessing a system under testing (SUT) by watching it run in order to find flaws. When the SUT's exterior performance differs from what is anticipated of it according to its specifications or any other description of the acceptable result, a failure is discovered [7]. The test case is an important part of the testing process. In essence, a test case specifies the conditions under which the SUT must be run in order to detect a failure. Software testing is a critical component in determining the product's quality and utility, as well as ensuring that the final outcome is in the customers' best interests [8].

2.2 Test Case Prioritization

The method of selecting test cases is to arrange them in a logical sequence based on several factors that improves the test cases' efficacy in reaching a performance target. Test case prioritization (TCP) is the process of scheduling test cases in this manner [9]. Running all test cases in an existing test suite can take an excessive amount of time. Regression testing for the test case is performed using test case prioritization strategies in order to optimize some of the objective functions [10]. TCP ranks test cases based on a set of criteria, with the highest-ranked test cases being executed first [11]. TCP is classified into several approaches which are history-based, code coverage based, search based and requirements-based [12].

The evaluation metrics used in TCP is known as Average Percentage Fault Detected (APFD). This measurement estimates the weighted average of the level of faults detected when implemented on the algorithms over the existence of a test suite. APFD values range from 0 to 100 where the larger numbers imply quicker is the better fault detection rate. The metrics calculation will be explained further in Section 3. Besides that, the another important measures taken into count for TCP is execution time. This parameter plays a vital role in order to assure the project timeline is not exceeded. Execution time is measured in order to track on how long does it take to prioritize test cases based on the faults and severity of the test case. When algorithms are implemented on TCP, the best cost which is represented by the distance is produced to determine the shortest path in prioritizing the test cases.

Table 1 - Comparison of algorithms used in TCP

Author	Algorithm	Purpose	Results
Test Case Prioritization Using Firefly Algorithm for Software Testing	Firefly Algorithm	To use the Firefly Algorithm with a fitness function generated using a similarity distance model to prioritize test cases optimally.	Firefly Algorithm performed better in terms of time execution and APFD.
Improved Meta-Heuristic Technique for Test Case Prioritization	Ant Colony Optimization	To use an improved meta-heuristic approach ACO method in a time constrained setting to find the best optimal path by prioritizing test cases.	It is easier to find the highest number of faults while executing the smallest amount of test instances by using the proposed algorithm.
Test case prioritization based on historical failure patterns using ABC and GA	-Artificial Bee Colony -Genetic Algorithm	To use Artificial Bee Colony Optimization and Genetic Algorithm to enhance fault detection by using historical execution of regression cycles.	The fault detection capabilities have significantly improved using ABC.

Flower Pollination Algorithm for Test Case Prioritization in Regression Testing	Flower Pollination Algorithm (FPA)	FPA is used to prioritize test cases from the original test suite to shorten the time it takes to run regression tests	FPA for TCP converged early with just one repetition required
Test Case Prioritization Using Bat Algorithm	Bat Algorithm	To implement a relatively new nature-inspired optimization method called the Bat algorithm	BA had shown a significant rise in the value of the evaluation metric of APFD.

Table 1 shows the comparison of previous works related to TCP that was implemented using various algorithms.

3. Methodology

The flow of the research is illustrated in Figure 1. Four consecutive phases are involved, which are:

i. Problem Definition
ii. Data Definition
iii. Implementation of ABC and ACO algorithms
iv. Validation and Evaluation of Results

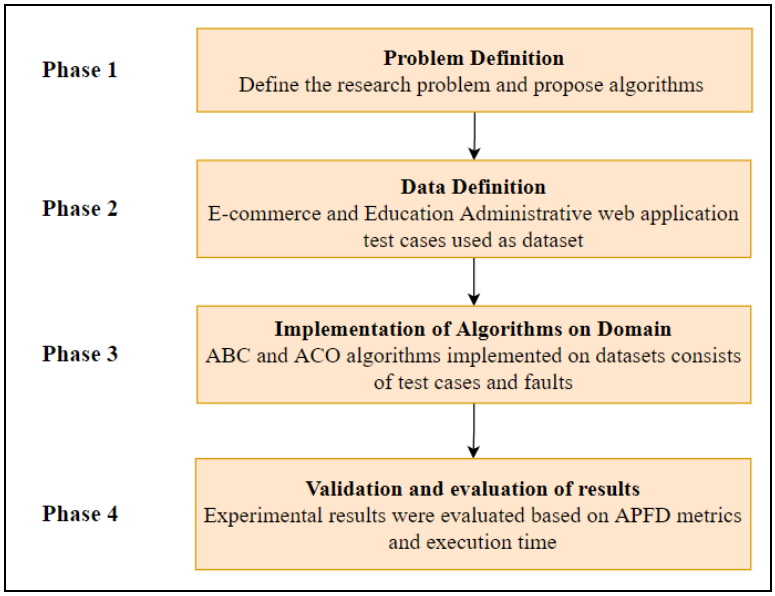


Fig. 1 - Research process flow

3.1 Problem Definition

The beginning stage of this research, a problem was identified in order to come up with the proposed research idea. In this case, the problem obtained was to test and compare the effectiveness of two different Swarm Intelligence algorithms to find the fault detected and execution time when both algorithms are implemented on test case prioritization as these two factors are very important in software testing to assure a successful product can be delivered on time.

3.2 Data Definition

In the phase of data definition, 2 case studies were explored further for this research which comprises of datasets from 2 separate web applications. These datasets are a set of test cases, that were obtained from a software testing company. As the data are confidential, the respective contents will not be disclosed in this report. A summary table with test cases, faults and time for both case studies which were the main data required for this study will be illustrated.

The first case study was conducted using an Online Shopping application. The test cases were generated by the Software Testing Company in Kuala Lumpur. This e-Commerce application consists of 6 functions which are Register, Login, Add To Cart, Check Out, Payment and Delivery. Each function was tested out and a set of test cases were created accordingly. Based on the data provided, there were in total of 10 test cases with 11 faults. Figure 2 (a) shows the summary of test cases and its respective faults along with the time for case study one.

The second case study was based on an Education Administrative application which consists of 8 functions. The application functions were Create Partnership Plans, Setting up a Partnership, several Admission Processes, Online

Registration, Partnership Program Enrolment and Graduation Process. Each function was tested out and a set of test cases were provided accordingly. This application has a total of 8 test cases with 8 faults. Figure 2 (b) displays the dataset used in case study two.

Faults	Test Case									
	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10
F1	/	/			/		/	/		
F2	/		/			/	/			
F3				/						/
F4						/	/	/		
F5										
F6								/		
F7								/		
F8										/
F9									/	
F10		/							/	
F11										
Total Faults	2	2	1	1	1	2	3	4	2	2
Time (s)	1.20	1.40	0.80	1.50	0.60	1.80	3.00	5.20	2.30	2.00

(a)

Faults	Test Case							
	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
F1	/			/				
F2			/			/		/
F3		/			/			/
F4		/						/
F5				/	/			
F6				/		/		/
F7				/	/			/
F8			/				/	
Total Faults	1	2	2	4	3	2	1	5
Time (s)	0.50	2.40	2.80	3.93	3.21	2.70	1.60	6.00

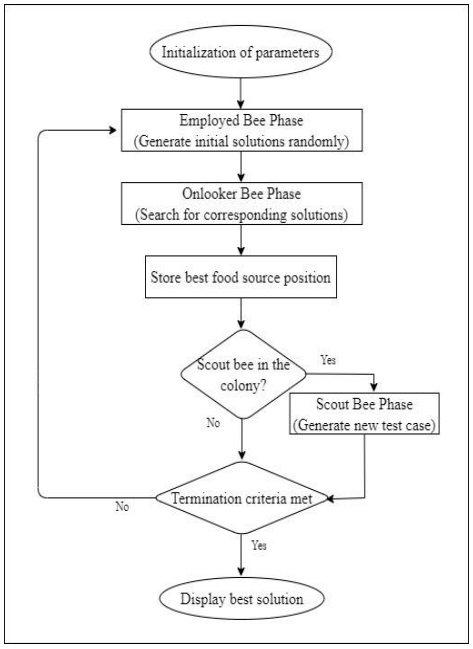
(b)

Fig. 2 - (a) Case study one; (b) case study two

3.3 Implementation of ABC on Test Case Prioritization

The process of ABC algorithm as shown in Figure 3 (a) starts by initializing the iteration and number of bees represented by the number of test cases as shown in Figure 3 (c). As soon as the execution begins, the time measurement is calculated. The test cases are picked based on the distance from the nest which is also known as tour, as shown in the syntax of the algorithm in Figure 3 (b) and the source's profitability which refers to the goodness of the test case for solving a problem. In this case, the distance from the nest is calculated by the number of faults detected in each test case. Then, employed bees are linked to the test cases where they work, and these bees conduct a waggle dance to transmit the information with the other bees. From the information given by the employed bees, onlooker bees wait in the nest for the emergence of the most profitable test case while scouts look for new nearest test cases. The process repeats until the best test cases are obtained based on the tour and cost value captured from the process. Hence, the test cases will be prioritized with the most fault detection and the execution time will be captured when the maximum iteration is reached.

The same flow was implemented on both, case study one and case study two for ABC algorithm and the results were gathered. Figure 3 (b) shows a snippet code of ABC algorithm used in both case studies while Figure 3 (c) shows the initialization values of the constant parameters set.



(a)

```
%ABC Algorithm Main Loop
for it=1:MaxIt

   for i=1:nEmployeed
       bestnewbee.Cost=inf;
       for j=1:nEmployedBee
       newbee.A=bee(i).Tour;
       lon=unidrnd(3);
       if lon==1
           newbee.Tour = swap(newbee.Tour,bee(i).Tour,map);
           newbee.Cost = CF(newbee.Tour,map);
           if newbee.Cost<bestnewbee.Cost
               bestnewbee=newbee;
           end
       elseif lon==2
           newbee.Tour = revers(newbee.Tour,bee(i).Tour,map);
           newbee.Cost = CF(newbee.Tour,map);
           if newbee.Cost<bestnewbee.Cost
               bestnewbee=newbee;
           end
       else
           newbee.Tour = insert(newbee.Tour,bee(i).Tour,map);
           newbee.Cost = CF(newbee.Tour,map);
           if newbee.Cost<bestnewbee.Cost
               bestnewbee=newbee;
           end
       end
       end
       if bestnewbee.Cost<bee(i).Cost
           bee(i)=bestnewbee;
       end
```

(b)

Constant Parameters	Case Study One	Case Study Two
Iteration	100	
Number of bees	10	8
Alpha	1	
Beta	1	

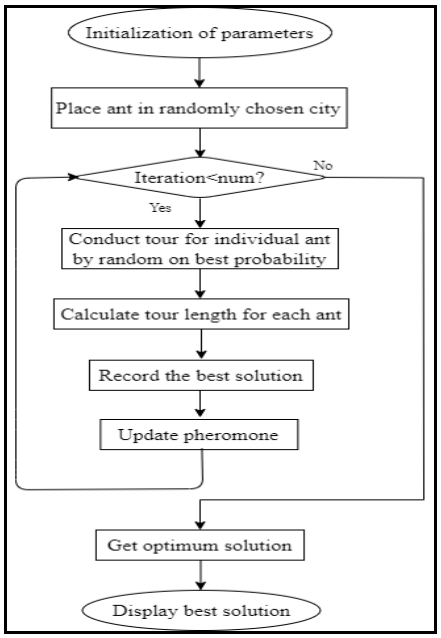
(c)

Fig. 3 - (a) ABC algorithm; (b) ABC snippet code; (c) ABC initialization values

3.4 Implementation of ACO on Test Case Prioritization

As shown in Figure 4 (a), the ACO algorithm process begins with initialization of iterations and artificial ants based on the variables in Figure 4 (c). When the algorithm is executed, the time evaluation will immediately start. Then, the ants will determine which test case to visit next based on the fault detected. The feasible path is selected based on the highest pheromone. The pheromone level is then updated and the process continues by determining if the termination conditions are fulfilled. If the conditions are achieved, the process ends, else it repeats until all test cases have been analyzed. By doing so, the shortest path travelled by the ants will prioritize test cases based on the maximum fault detection in the early stage. All of these processes have been translated into the syntax in the algorithm in Figure 4 (b). Due to ants' natural tendency to choose the shortest path to food sources and return to their nests sooner, the route with the highest proportion of pheromones indicates the shortest path. The execution time will be terminated and evaluated once all iterations have been completed. The shortest path and execution time will then be shown.

The same procedure was used in case study one and case study two for this ACO algorithm. Figure 4 (b) shows a snippet of the ACO algorithm MATLAB code applied for both case studies while Figure 4 (c) shows the initialization values of the constant parameters set.



(a)

```
max_it=100;
best_L=nan(max_it,1);

for t=1:max_it

    for k=1:nAnt
        ant(k).Tour=randi(1,1,[1 nCity]);
    end

    for i=2:nCity

        for k=1:nAnt
            N=find(~ismember(1:nCity,ant(k).Tour));
            ant(k).P=zeros(1,nCity);
            ii=ant(k).Tour(end);

            for j=1:numel(N)
                jj=N(j);
                ant(k).P(jj)=tau(ii,jj)^alpha*etha(ii,jj)^beta;
            end

            ant(k).P=ant(k).P/sum(ant(k).P);
            ant(k).Tour(end+1)=Probability(ant(k).P);
        end

    end
```

(b)

Constant Parameters	Case Study One	Case Study Two
Iteration	100	
Number of ants	10	8
Alpha	1	
Beta	1	
Pheromone value	0.5	

(c)

Fig. 4 - (a) ACO Algorithm; (b) ACO snippet code; (c) ACO initialization values

3.5 Evaluation Parameters

3.5.1 Average Percentage of Fault Detected (APFD)

The weighted average percentage of faults detected (APFD) was introduced by Rotherm el as a measurement to analyze prioritization performance [13]. The weighted mean of the faults covered by the test cases is computed using the Average Percentage of Fault Detected (APFD) value in line with the test case position in the test suite. The mathematical expression is shown in (1):

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n} \quad (1)$$

Where TFi is position of first test case that exposed the faults. Next, n represents total number of test cases and m represents a set of faults detected. The APFD with higher values implies higher fault detection rates [14]. The results with higher APFD value will be the better performing algorithm.

3.5.2 Execution Time

The second evaluation parameter is based on the execution of time when the algorithms were applied. The time is measured using seconds (s) unit. The execution with lowest time is considered the better algorithm. The time calculation equation is shown in (2):

$$\text{Execution time} = \text{Start time} - \text{End time} \quad (2)$$

4. Results and Discussion

4.1 Fault Detection Rate

Average Percentage of Fault Detected (APFD) value is used to calculate the weighted mean of faults covered by test cases in accordance with the test case position of the test suite. If the fault detection is rate is faster, the APFD value is higher. The higher APFD value represents the better performing algorithm. Based on the study conducted using ABC and ACO algorithms on two cases studies, the APFD values were calculated based on the prioritized test cases.

The gathered results were also compared with the original set of non-prioritized test cases. Table 2 represents the values of the prioritized and non-prioritized test cases.

Table 2 - Comparison of APFD

Algorithm	Case Study One	Case Study Two
ABC	0.80	0.81
ACO	0.71	0.64
Original	0.61	0.70

Based on the Table 2, the comparison of APFD values are illustrated through a graph in Figure 5. It is shown that the APFD values for ABC algorithm is the highest for both Case Study One and Two, which are 0.80 and 0.81 respectively. On the other hand, for ACO, Case Study One is 0.71 and Case Study Two is 0.64 while the original non-prioritized ordering is 0.61 for Case Study One and 0.7 for Case Study Two.

Fig. 5 - Comparison graph for APFD

4.2 Execution Time

In the process of SDLC, time plays a very important role in ensuring the end product can be delivered within the project timeline. Hence, in Software Testing execution time should always be taken into consideration in order to avoid delays.

Based on the results collected from the implementation, a comparison between ABC, ACO and original set of test cases is illustrated in Table 3 for both the case studies in terms of execution time.

Table 3 - Comparison of execution time

Algorithm	Case Study One (s)	Case Study Two (s)
ABC	8.64	8.83
ACO	0.69	1.22
Original	19.8	23.14

From the data collected in the above table, a comparison graph as shown in Figure 6 was created to display the difference of the execution time. For Case Study One, it can be seen that the non-prioritized test cases have the highest execution time of 19.8 seconds. In terms of the prioritized test cases, ABC took a longer time which is 8.64 seconds and ACO only 0.69 seconds. Case Study Two also showed the same outcome whereby the non-prioritized test cases took the longest time which is 23.14 seconds while the prioritized, ABC and ACO took 8.83 seconds and 1.22 seconds respectively.

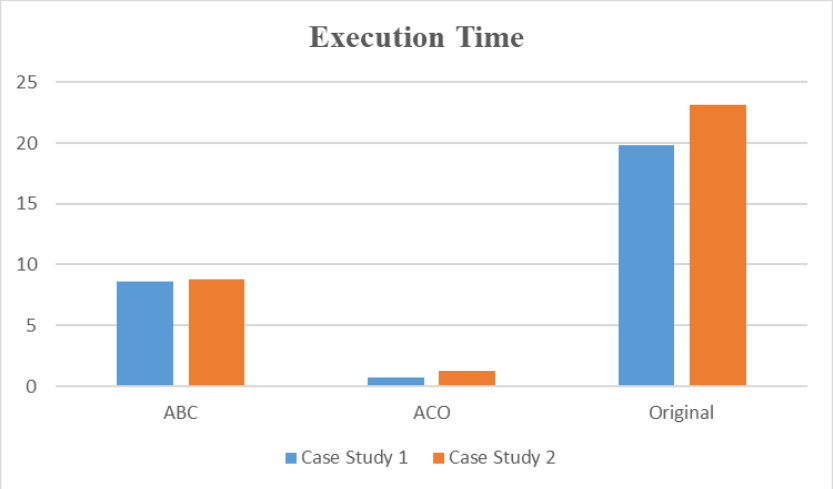


Fig. 6 - Comparison graph for execution time

Based on the overall data gathered from this execution on Case Study One and Case Study Two, it can be seen that both ACO and ABC has its own advantage. In terms of APFD, ABC is shown to be the algorithm that covered the highest fault detection rate, while for execution time, ACO took the shortest time to prioritize the test cases.

5. Conclusion

This research was conducted with the idea of implementing Swarm Intelligence algorithms on Test Case Prioritization to Improve Fault Coverage and Execution Time for Web Applications. The algorithms used were ABC and ACO. The algorithms were implemented on two different datasets namely Case Study One and Case Study Two. Case Study One consist of a set of 10 test cases with 11 faults from an Online Shopping web application while Case Study Two consists of 8 test cases and faults from an Education Administrative Web Application. Both the datasets were obtained from a Software Testing company. At the end of the execution, the results were gathered and discussed in. The results were compared between non-prioritized and prioritized test cases. It can be concluded that both ABC and ACO performed well in its respective ways. When compared in terms of APFD, ABC had shown to give a higher value of fault detection for both the datasets while in terms of execution time, ACO had converged faster during the execution. Hence, both ABC and ACO algorithms are equally good when implemented on test case prioritization. As a conclusion, the implementation of this research was a success as it achieves the objectives that were analysed. This research phase started with problem definition, data definition which was conducted by gathering web application test cases that were obtained from a software testing company, followed by implementation of ABC and ACO algorithms using MATLAB2022 tool and finally validation and evaluation of results based on APFD and execution time.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, for supporting this project.

References

[1] Raamesh, L. (2016). Test case prioritization. Int. J. Appl. Eng. Res., vol. 10, no. 13, 32917-32922.
[2] Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., Hamed, H. N. A., & Suffian, M. D. M. (2019). Test case prioritization using firefly algorithm for software testing. IEEE access, 7, 132360-132373.
[3] Panwar, D., Tomar, P., Harsh, H., & Siddique, M. H. (2018). Improved meta-heuristic technique for test case prioritization. In Soft Computing: Theories and Applications: Proceedings of SoCTA 2016, Volume 1 (pp. 647-664). Springer Singapore.
[4] Khanna, M., Chauhan, N., Sharma, D. K., & Toofani, A. (2017). Test case prioritisation during web application testing. International Journal of Computer Applications in Technology, 56(3), 230-243.
[5] Alaqail, H., & Ahmed, S. (2018). Overview of software testing standard ISO/IEC/IEEE 29119. International Journal of Computer Science and Network Security (IJCSNS), 18(2), 112-116.
[6] Tebes, G., Peppino, D., Becker, P., Matturro, G., Solari, M., & Olsina, L. (2020). Analyzing and documenting the systematic review results of software testing ontologies. Information and Software Technology, 123, 106298.
[7] Durelli, V. H., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R., & Guimarães, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. IEEE Transactions on Reliability, 68(3), 1189-1212.

[8] Alyahya, S., & Alsayyari, M. (2020). Towards better crowdsourced software testing process. International Journal of Cooperative Information Systems, 29(01n02), 2040009.
[9] Mittal, S., & Sangwan, O. P. (2018). Prioritizing test cases for regression techniques using metaheuristic techniques. Journal of Information and Optimization Sciences, 39(1), 39-51.
[10] Shaheamlung, G., & Rote, K. (2020, June). A comprehensive review for test case prioritization in software engineering. In 2020 International Conference on Intelligent Engineering and Management (ICIEM) (pp. 331-336). IEEE.
[11] Bajaj, A., & Sangwan, O. P. (2019). A systematic literature review of test case prioritization using genetic algorithms. IEEE Access, 7, 126355-126375.
[12] Ying, P., & Fan, L. (2017). Methods for test case prioritization based on test case execution history.
[13] Chen, J., Zhu, L., Chen, T. Y., Towey, D., Kuo, F. C., Huang, R., & Guo, Y. (2018). Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. Journal of Systems and Software, 135, 107-125.
[14] Lima, J. A. P., & Vergilio, S. R. (2020). Test Case Prioritization in Continuous Integration environments: A systematic mapping study. Information and Software Technology, 121, 106268.