ISSN 2278-3091
Volume 8, No.1.4, 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse0481.42019.pdf
https://doi.org/10.30534/ijatcse/2019/0481.42019

Measuring the Efficiency of MFWA Technique for Prioritizing Event Sequences Test Cases

Johanna Ahmad[1], Salmi Baharom[2], Abdul Azim Abd Ghani[3], Hazura Zulzalil[4], Jamilah Din[5]
[1]Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Malaysia, johanna@ump.edu.my
[2,3,4,5]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia, {salmi,azim,hazura,jamilahd}@upm.edu.my

ABSTRACT

The software development phase would frequently go through several changes and modifications. These are challenges that the tester of a system needs to face to ensure that the quality of the system aligns with the budget, resources, and time to deliver. Numerous techniques have been proposed to solve these problems and one of them is the test case prioritization (TCP) technique. The TCP technique is widely used for single event test cases. Thus, this paper would like to propose the Multifactor Weightage Approach (MFWA) using combinations of six factors to prioritize event sequence test cases. The percentage of test effort efficiency was used to measure the efficiency of the comparison technique. The results showed that the MFWA was more efficient compared to the random technique in terms of detecting faults earlier.

Key words: Software testing, test case prioritization, event sequences, multifactor, efficiency.

1. INTRODUCTION

A software testing process may consist of several steps, such as planning, designing, specifying, executing, and quantifying either in regression or non-regression testing [1]. These are important steps and sometimes, this process may stay in the maintenance phase, especially when there are frequent changes during the developmental phase. Any changes must be tested, which could lead to an exhaustive testing. A product that consists of 33,000 test cases might require 1,100 machine hours to make sure all test cases are executed. This step might become more complicated for event sequence test cases, which can be up to an infinite number and can cause a considerable degree of redundancies. Whenever changes are made, the number of test cases will increase, in addition to the existing number of test cases, to test the new functionalities. Therefore, several techniques were proposed to solve this problem, such as test minimization, test case prioritization, and test selection. The test case prioritization (TCP) technique has been used since 1997 to provide new test case ordering.

Over the past few years, the TCP technique has been developed using more than one factor. Some researchers believe that adopting multiple criteria can improve the performance in detecting faults earlier and reduce time during testing phase [2]. The weightage approach was adopted in the TCP technique to provide ordering for test cases, whereby lead by test cases with the highest priority value. However, how to cater issue of similar priority value become one of the most significance issues [3].

Thus, this study is proposing the Multifactor Weightage Approach (MFWA), with combinations of six factors specifically for event sequence test cases. The aim is to produce a unique weight for each test case and avoid using the random technique. An event is known as an external observable phenomenon, such as a system response, an environmental or a user stimulus [4]. One of the challenges in performing event-driven testing is the large number of possible events that need to be tested. The organization of this paper is as follows: Section 2 will present a review of previous works on the TCP technique. The proposed MFWA that uses six factors will be described in Section 3. The experiments and results of the comparison technique will be explained in Section 4. While the conclusions of this study will be presented in Section 5.

2. RELATED WORK

There are two types of TCP, as defined by [5], namely, the general TCP and the version-specific TCP. In general, the TCP technique and the testing execution of test suite *T* are applied to the base version of a program *P*, with the objective that the prioritized test suite *T* will be more successful than the original test suite. Meanwhile, the version-specific TCP technique refers to a case where the test suite *T* is prioritized, with the objective of finding the new ordering of test suite *T* execution for a specific version of *P'* and *P*. Test cases that have higher values will be prioritized during the execution process [6].

In [7] applied fault coverage to measure the capability of test cases to detect faults earlier. Meanwhile, the execution time measurement was used to achieve the objective of this study, which was to select test cases that manage to cover every fault in minimum execution time. Test cases with higher ratio of fault coverage will be executed earlier than others. The prioritization algorithm proposed by [8] consists of three approaches, namely, random weight, equal weight, and fault-prone weight. The importance of each event is used to classified the event type before the process of assigning

weight. Three experiment results showed that the prioritization algorithm based on the weighted concept was capable of improving the rate of fault detection [8]. [9] applied Harrold-Gupta-Soffa, traditional greedy, and 2-optimal greedy algorithms, as well as the algorithm for the TCP and test suite reduction. The application of greedy for the prioritization algorithm showed that a large test suite with a short execution time is better than a small test suite with a longer run-time. The coverage effectiveness metric, which is based on the cumulative coverage of the tests over time, also showed that the prioritization algorithm had reached its maximum improvement.

In [10] enhanced the tie-breaking technique by applying a traditional greedy algorithm. The proposed technique adopted the random technique only whenever two or more test case has equal priority value, with respect to the coverage criteria. In this study, the random technique was applied during the elimination process to exclude test cases that would likely detect faults than the preserved ones. In 2016, [11] proposed an algorithm for an enterprise cloud application using the use-case weight approach. The proposed approach was developed to avoid simultaneous execution of test cases across different servers. Every use-case point would have its own weight, which would be assigned based on its criticality. However, if the use-case weight remains not unique, then, the arity of the test case would be used. The arity is determined by how frequent the test cases access the services. If the arity approach is unable to solve the issue, the ties would be randomly broken.

3. PROPOSED APPROACH

This study assigned weight based on the cumulative weight gained from all potential factors, which were combined based on the 2017 SLR analysis [9]. The potential factors were complexity, redundancy, permutation, frequency, fault matrix, and distance. Distance was only applicable in cases two or more test case shared the same priority value, after going through all five factors. It can be concluded that distance was used in solving the same priority value issue. Each potential factor contributed to the production of the final test case weight, which was measured on a 10-point scale. This distance may break the ties and thus, avoid the random technique. The process of implementing the MFWA is depicted in Figure 1. The process of assigning weight is from one factor to another, until the final weight for the test case is produced.

The step-by-step process to produce the test case weight is explained as follows. Two considerations have been defined in selecting the subject programs for the experiment. The first consideration was of utmost importance since the concept of hold data state memory was one of the properties used to calculate the redundancy weight. Furthermore, the properties of an event sequence test case would consist of data state memory. Redundancy was the second factor for the MFWA. All selected benchmark programs have memories (i.e., holds data state value during the test case execution). The second consideration was the need to avoid biasness. Therefore, all five programs in the benchmark repository were developed using independent sources. These independent sources are available to the public. The selected subject programs were the Gomoku, Sodoku, HashTable, Circular Queue, and Bank. The size of the subject program is not important for this research due to the difficulties in obtaining subject program that have memories. However, this limitation is under consideration in future work. The description of each subject program is listed and summarized in Table 1. The step-by-step process to produce the test case weight is explained in the following section.

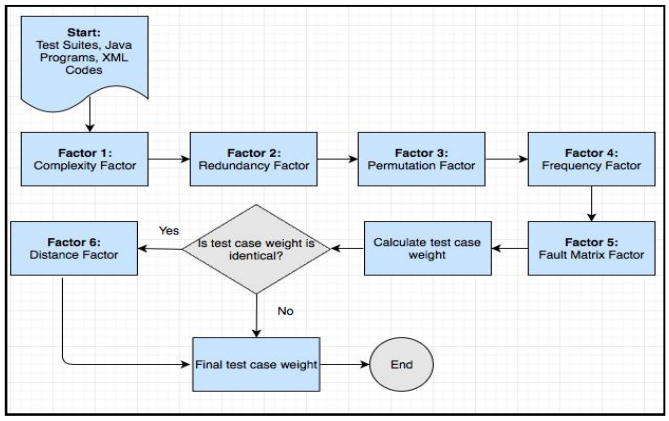

Figure 1: Steps in the MFWA Technique

Table 1: A Summary of the five subject programs

Subject Program	Lines of Codes	Total No. of Method
Gomoku	389	3
Sodoku	196	3
Hash Table	216	3
Circular Queue	69	3
Bank	280	4

3.1 Complexity

Complexity is defined as the degree of difficulty in verifying and understanding the design or implementation of a system [10]. With respect to complexity, the measurement included the aspects of the structural design of the program, the way the computational program is being handled, the algorithms in the program, as well as the logical and functional aspects of the program. Many researchers believe that the complexity of a system or a program needs to be calculated when predicting the reliability and maintainability of the system [11]. Thus, numerous complexity metrics have been proposed over the years. Based on the literature review, five of the most popular complexity metrics are the Lines of Codes (LOC), the Information Flow Complexity (IFC), the Unique Complexity Metric (UCM), the McCabe's Cyclomatic Complexity (McCabe), and the Function Point (FP). These complexity metrics were selected as the comparison software metrics for this study. Based on the analysis on the strengths and weaknesses of each software metric, the UCM was found to be the most suitable metric in measuring the event complexity. Furthermore, the event sequence test cases

consisted of a combination of events, whereby each event has its own complexity.

The complexity value obtained from this factor was used to calculate the frequency of the weight later. Each test case may have different combinations of events, while each event may have its own behaviour. Thus, the capability of each test case in detecting faults may differ from one test case to another. During the UCM calculation, the weight was given based on the classification defined by the concept of Basic Control Structures (BCSs) [10], as shown by Equation (1). The complexity was actually directly dependent on the cognitive weight of the BCS.

$$Unique\ Complexity\ Metric = \sum_{i=1}^{n} (SOO_i + CW_i) \quad (1)$$

3.2 Redundancy

[12] stated that a large number of test cases may caused redundancy. The sequence of an event test case can produce a large test size since it is the behaviour of the event sequence test case to have large input of sequences, which can sometimes be infinite. In some cases, the combination of events may be the same, but the value of the internal data state might be different [13]. In this study, the concept of the data state referred to the concept of two dimensional arrays, [i][j], whereby *i* represents the event sequence number, while *j* denotes the content of the respective event.

As the first hierarchy for the redundancy factor, the redundancy that occurred in the test case itself was defined first, which was categorized as Redundancy Type 1. Meanwhile, Redundancy Type 2 belonged to the case where the value of the data state in the test case is a subset of another test case within the test suite. It would wasting time, cost and resources to execute the same test case. Previous researches have focused on test suite reduction once the redundant test case exists. However, in this study, all test cases were executed and no test reduction process was involved. Two steps of calculations were required before the weight for Redundancy Type 1 could be produced. The dissimilarity of weight in the test case ($DWtc_j$) was calculated once the value for the number of data state ($No.of\ ds_{tcj}$) and the number of the redundant data state ($No.of\ redundant\ ds_{tcj}$) for the test cases were calculated. The values ranged between 1 and 10. $DWtc_j$ was calculated using the following Equation (2):

$$DWtc_j = ((\text{No of } ds_{tcj}) - (\text{No of redundant } ds_{tcj})) / 10 \quad (2)$$

Equations (3) and (4) were used to produce the weight for Redundancy Type 2. These values were used to calculate the $DWts_j$, which can be calculated using the following Equations (3) and (4):

$$\text{No of non} - \text{redundant } ds_{tsj} = (\text{No of } ds_{tsj}) / (\text{No of redundant } ds_{tsj}) \quad (3)$$

$$DWts_j = ((\text{No of non} - \text{redundant } ds_{tsj}) / (\text{No of } ds_{tsj}))/10 \quad (4)$$

3.3 Permutation

With the realization of the guarantee that the combination of events would be sufficient for a high coverage of fault detection, permutation was identified as another factor that must be considered. The t-way test was applied in this study to generate optimum test cases with enough coverage to detect faults during the test. The t-way concept was able to reduce the number of tests to be conducted during the testing phase. For example, for a system that consists of 4 events and 10 components, the $4^{10} = 1{,}048{,}576$ test configuration would be needed to ensure that the coverage is fulfilled [14]. The sequence covering array (SCA) algorithm was used to ensure that all t-way sequences have been tested during the testing phase. [15] have also applied the SCA in improving the efficiency of GUI testing, with the objective that each node in the sequence test case would contain a set of events that has been defined. The SCA has been defined as follows:

Definition A sequence covering array or SCA (*N,S,t*) means that the *N x S* matrix entries are from a finite set *S* of *s* symbols, while every *t-way* combinations would occur at least one row and each row would consist of permutations of the *s* symbol.

This concept was applied based on the original covering array proposed by [16]. A covering array is defined as CA_γ *(N;t,k,v),* whereby *N x k* is the array. Normally, in software testing, $\gamma = 1$. A sequence covering array is generated based on the concept of covering the array algorithm [16]. Table 2 shows HashTable program pairwise interactions. The Gomoku, Sodoku, Circular Queue, and Bank programs went through the same processes. The listed pairwise events were considered as a minimum coverage of interactions to cover the whole system during testing.

Table 2: Pairwise Interactions for HashTable Program

No	Pairwise Event
1	(hashFunction, findKey)
2	(hashFunction, display)
3	(hashFunction,hashFunction)
4	(findKey, hashFunction)
5	(findKey, display)
6	(findKey, findKey)
7	(display, hashFunction)
8	(display, findKey)
9	(display, display)

3.4 Frequency

Frequency was chosen as one of the potential factors due to the behaviour of the event sequence test case that has a repetition of events. In this study, frequency was calculated based on the number of times that the pairwise event interactions occurred in a test case. Frequency has been widely used in the TCP technique and has been applied in various research areas. The complexity values obtained from the first factor were used to calculate the frequency weight in

this phase. The complexity factor and the permutation factor were combined with the frequency factor to produce the frequency weight. The frequency weight was calculated for each pairwise event using the following Equation (5):

Frequency weight = (no of pairwise events that exist in a test case x event complexity value) (5)

3.5 Fault Matrix

The fault matrix is responsible for validating the capability of the test case in detecting seeded mutants. The SLR analysis that was conducted during the early stage of this study showed that the fault matrix has become the most popular factor used by numerous researchers [9]. They believe that mutation could help improve the effectiveness of the testing phase. Meanwhile, other researchers have combined other factors with the fault matrix. They believe that combinations of more than one factor can increase the performance of the TCP technique. Furthermore, it can break the ties when more than one test case is sharing the same priority value. In this study, the Jester Mutation was selected as the mutation tool [17]. Jester will automatically create mutants for the program based on the list of mutation operators. Nevertheless, the number of mutants was varied when injected in each of the Java programs.

In this study, two steps were needed for the fault matrix algorithm before the fault matrix weight could be produced. The first step was to calculate the number of killed mutants (*nkm*) by each test case and the number of live mutants (*nlm*). In the first step, the ordering was based on the highest number of test cases with the highest number of detected mutants. This was followed by other test cases that have fewer number of mutants detected [18]. However, for the second ordering, the selection process differed from the method proposed by [18]. To avoid selecting test cases that killed the same mutants, the next process in this study was to find the test case that managed to kill live mutants that have been left out by the ordered test case. In this study, the weight of the fault matrix was given based on the weightage concept. After the fault matrix's weight was produced, the test case went through the final sorting for the fault matrix, whereby the ordering of the test cases was based on their weight.

3.6 Distance

The idea of using distance was initiated with the aim of producing a unique weight. This unique weight was proposed to solve the limitations faced by the current TCP technique for solving the same priority value issue. In this study, the Jaccard Distance, also known as the Jaccard Similarity Coefficient, was applied in its place. [19] applied the Jaccard Distance to compare the similarities and diversities between sample sets. Meanwhile, this study used the Jaccard Distance to solve the same priority value issue by measuring the similarities of the data state values among the test cases. The Jaccard Distance was calculated using the following Equation (6) [20]:

$$\text{Jaccard Distance } (p_a, p_b) = 1 - \frac{|p_a \cap p_b|}{|p_a \cup p_b|} \quad (6)$$

where p_a *and* p_b represent the test case numbers and they consist of different sets of event sequences. According to [19], the value of the Jaccard Distance may vary between 0 and 1. If the distance value is zero, it means that p_a *and* p_b are the same. However, if the distance value is 1, this would indicate that there is no similarity between p_a *and* p_b**.** In this study, the distance value was calculated based on the similarity of data state.

4. EXPERIMENT AND ANALYSIS

This study has compared the time efficiency between the MFWA and the random technique. Random technique has been chosen since this study want to focus on how to define an indicator to prove that test cases with the same priority value have different fault coverage. The intention of this research is to avoid the random technique used to solve the issue of the same priority value, since it has been proven to be ineffective [21], [22]. The ordering of test suites for the Gomoku program, the Circular the Queue program, and the HashTable program uses the pure random technique. Meanwhile, test suites for the Sodoku program and the Bank program are taken directly from the benchmark repository without any changes. In this case, it is not certain whether the ordering of test cases in both test suites are pure random technique, random technique, or based on a guideline.

The experiment was conducted on five Java programs as subject programs. All five subject programs were taken from the benchmark repository. Two considerations to select the subject program; the program must have a variable that holds data state value during the test case execution, and all programs must be developed by independent sources to avoid biasness. In line with the consideration and intention of measuring the efficiency of the proposed technique during the prioritization process, the MFWA was compared with the random technique. The random technique is known as the fundamental testing method, whereby it simply selects test cases in a random order. Meanwhile, the MFWA combines six factors to produce the unique priority value. The efficiency of both techniques was measured using the following Equation (7):

$$\text{The percentage time effort efficiency (TEE)} = \frac{|Time| - |Time|_{technique}}{|Time|} \times 100 \quad (7)$$

where |Time| is the time taken to apply all test cases, while $|Time|_{technique}$ refers to the time taken by the random technique and the MFWA to detect faults.

Equation (7) is widely used in current reduction techniques [19], [23], [24]. However, this study had only applied this equation to measure the efficiency of each technique in detecting faults. Efficiency was measured in terms of rank prioritization. Any ordering test cases that can detect all faults earlier would be considered as the most efficient. Table 3 lists

the analysed time efficiency for both techniques. It shows that the percentage of time effort efficiency for MFWA was higher than the percentage of time effort efficiency for the random technique.

Table 3: Time effort efficiency (TEE) percentages and differences for MFWA and random technique

Subject Program	TEE for MFWA	TEE for random technique	Differences
Circular Queue	89.52	32.07	57.45
Bank	68.93	62.71	6.22
Sodoku	96.63	88.07	8.56
Gomoku	2.54	1.94	0.6
HashTable	95.07	45.6	49.47

5. CONCLUSION

This study has proposed an approach, namely, the MFWA, which was applied using the weighted priority concept to sort test case ordering. The new test case ordering was to avoid using the random technique, once more than one test case share the same weightage during the prioritization process. As depicted in Figure 1, all test cases went through five processes. Upon detecting that more than one test case share the same final weight, the respective test cases were grouped together and went through the distance factor process. The MFWA was compared to the random technique since this technique will simply fetch any test case that has the same weightage. The MFWA was considered as more efficient than the random technique since it was able to detect faults earlier.

ACKNOWLEDGEMENT

The authors would like to acknowledge Universiti Putra Malaysia for the financial support under the Putra Grant – Putra Graduate Initiative (IPS); Project code-GP-IPS/2018/9621100.

REFERENCES

[1] S. Nayak, C. Kumar, and S. Tripathi, "**Effectiveness of prioritization of test cases based on Faults**," *2016 3rd Int. Conf. Recent Adv. Inf. Technol. RAIT 2016*, pp. 657–662, 2016. https://doi.org/10.1109/RAIT.2016.7507977

[2] H. Srikanth and M. B. Cohen, "**Regression testing in Software as a Service: An industrial case study**," *2011 27th IEEE Int. Conf. Softw. Maint.*, pp. 372–381, 2011. https://doi.org/10.1109/ICSM.2011.6080804

[3] J. Ahmad and S. Baharom, "**A Systematic Literature Review of the Test Case Prioritization Technique for Sequence of Events**," *Int. J. Appl. Eng. Res.*, vol. 12, no. 7, pp. 1389–1395, 2017.

[4] F. Belli, M. Eminov, and N. Gokce, "**Prioritizing Coverage-Oriented Testing Process - An Adaptive-Learning-Based Approach and Case Study**," *31st Annu. Int. Comput. Softw. Appl. Conf. - Vol. 2 - (COMPSAC 2007)*, no. Compsac, pp. 197–203, 2007. https://doi.org/10.1109/COMPSAC.2007.169

[5] A. G. Malishevsky, G. Rothermel, and S. Elbaum, "**Modeling the cost-benefits tradeoffs for regression testing techniques**," *Softw. Maintenance, 2002. Proceedings. Int. Conf.*, pp. 204–213, 2002.

[6] G. Rothermel, R. H. Untch, C. Chu, M. J. Harrold, and I. C. Society, "**Prioritizing Test Cases For Regression Testing Prioritizing Test Cases For Regression Testing**," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, 2001. https://doi.org/10.1109/32.962562

[7] M. Tyagi and S. Malhotra, "**Test case prioritization using multi objective particle swarm optimizer**," *2014 Int. Conf. Signal Propag. Comput. Technol. (ICSPCT 2014)*, pp. 390–395, 2014. https://doi.org/10.1109/ICSPCT.2014.6884931

[8] C. Y. Huang, J. R. Chang, and Y. H. Chang, "**Design and analysis of GUI test-case prioritization using weight-based methods,**" *J. Syst. Softw.*, vol. 83, no. 4, pp. 646–659, 2010. https://doi.org/10.1016/j.jss.2009.11.703

[9] J. Ahmad and S. Baharom, "**Factor Determination in Prioritizing Test Cases for Event Sequences : A Systematic Literature Review**," *J. Telecommun. Electron. Comput. Eng.*, vol. 10, no. Xe-ISSN: 2289-8131, pp. 1–6, 2018.

[10] S. Misra and I. Akman, "**A unique complexity metric,**" *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5073 LNCS, no. PART 2, pp. 641–651, 2008. https://doi.org/10.1007/978-3-540-69848-7_52

[11] B. Ceylan and M. M. Inceoğlu, "**A Unique Complexity Metric**," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5072, no. PART II, pp. 641–651, 2008.

[12] J. F. Silva Ouriques, E. G. Cartaxo, and P. D. Lima Machado, "**Revealing influence of model structure and test case profile on the prioritization of test cases in the context of model-based testing**," *J. Softw. Eng. Res. Dev.*, vol. 3, no. 1, p. 1, 2015. https://doi.org/10.1186/s40411-014-0015-5

[13] S. Baharom and Z. Shukur, "**State-Sensitivity Partitioning Technique for Module Documentation-based Testing**," *Bus. Transform. through Innov. Knowl. Manag. An Acad. Perspect.*, vol. 1, no. Midd, pp. 472–483, 2009.

[14] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "**Constructing Test Suites for Interaction Testing**," *Softw. Eng. 2003. Proceedings. 25th Int. Conf.*, 2003. https://doi.org/10.1109/ICSE.2003.1201186

[15] X. Yuan, M. Cohen, and A. M. Memon, "**Covering Array Sampling of Input Event Sequences for Automated Gui Testing**," *Proc. Twenty-second IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp.

405–408, 2007.
https://doi.org/10.1145/1321631.1321695

[16] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, "**Combinatorial methods for event sequence testing**," *Proc. - IEEE 5th Int. Conf. Softw. Testing, Verif. Validation, ICST 2012*, pp. 601–609, 2012.

[17] I. Moore, "**Jester-a JUnit test tester**," *Proc. 2nd XP*, pp. 84–87, 2001.

[18] A. M. Sultan, "**An Optimized Test Case Generation Technique For Enhancing State-Sensitivity Partitioning**," Universiti Putra Malaysia, 2017.

[19] A. B. Sanchez, S. Segura, and A. Ruiz-Cortes, "**A Comparison of Test Case Prioritization Criteria for Software Product Lines**," *Softw. Testing, Verif. Valid. (ICST), 2014 IEEE Seventh Int. Conf.*, pp. 41–50, 2014.
https://doi.org/10.1109/ICST.2014.15

[20] B. Jiang, Z. Zhang, W. K. Chan, T. H. Tse, and T. Y. Chen, "**How well does test case prioritization integrate with statistical fault localization?**," *Inf. Softw. Technol.*, vol. 54, no. 7, pp. 739–758, 2012.
https://doi.org/10.1016/j.infsof.2012.01.006

[21] P. Tonella, P. Avesani, and A. Susi, "**Using the Case-Based Ranking Methodology for Test Case Prioritization**," *22nd IEEE Int. Conf. Softw. Maint.*, pp. 123–132, 2006.
https://doi.org/10.1109/ICSM.2006.74

[22] A. Ammar, S. Baharom, A. A. A. Ghani, and J. Din, "**Enhanced Weighted Method for Test Case Prioritization in Regression Testing Using Unique Priority Value**," in *Information Science and Security (ICISS), 2016 International Conference*, 2016.
https://doi.org/10.1109/ICISSEC.2016.7885851

[23] S. Parsa and A. Khalilian, "**On the Optimization Approach towards Test Suite Minimization**," *Int. J. Softw. Eng. Its Appl.*, vol. 4, no. 1, pp. 15–28, 2010.

[24] P. Udupa and S. Nithyanandam, "**An Efficient software testing by test case reduction, prioritization and prioritized parallelization**," *Int. J. Pure Appl. Math.*, vol. 118, no. 22, pp. 1879–1885, 2018.