ICIC Express Letters
Part B: Applications
Volume **13**, Number **8**, August **2022**

ICIC International ©2022 ISSN 2185-2766
pp. **869–877**

A MULTILEVEL SYSTEM TEST CASES PRIORITIZATION TECHNIQUE FOR OBJECT ORIENTED SOFTWARE

Vedpal[1,*] and Naresh Chauhan[2]

[1]Department of Computer Applications
[2]Department of Computer Engineering
J.C. Bose University of Science and Technology, YMCA, Faridabad
NH-2, Sector-6, Mathura Road, Faridabad 121006, Haryana, India
nareshchauhan19@jcboseust.ac.in
*Corresponding author: ved_ymca@jcboseust.ac.in

Received September 2021; accepted December 2021

Abstract. *Testing of software is the most important and expensive activity in the software development life cycle. The testing of the software is performed under various constraints like time, and budget. Every software has a large number of test cases. It is not possible to execute each and every test case because every test case is associated with the time and cost. In this paper, a system test case prioritization technique is presented. The proposed technique works at three levels. At the first level, the requirement is prioritized, at the second level, the modules of the prioritized requirements are prioritized and at the third level, the test cases of the prioritized modules are prioritized. Some factors have been considered to perform prioritization at each level. Every factor has been assigned a weight which is determined using the SPSS Modeler on the basis of the four algorithms. For experimental verification and validation, the proposed techniques have been applied on two software.*

Keywords: Software testing, Test case prioritization, System testing, TCP, Multilevel testing

1. **Introduction.** To ensure the quality of the software, testing of the developed software is required [1,2]. Testing in specialized environments requires more attention with more specialized testing techniques. The testing techniques are dependent on the environment [3] and they may change their working behavior according to the environment. To perform the effective testing within time and budgets [4], the test cases are ordered and executed in such a way that they detect the maximum faults as earlier as possible. The testing of the software has been performed at three levels. These levels are unit testing, integration testing and system testing. In the system testing [5] whole software is tested by considering the various parameters like the load, and stress. For effective system testing of the software, a large number of test cases are executed. The cost of testing of the software includes direct and indirect cost of testing. The direct cost of testing includes the various testing activities, whereas indirect testing cost includes cost occurring due to performing poor testing of the software. The study shows that 18% to 35% of the project budget has been spent on testing and quality assurance of the software [6]. So, it is very beneficial to execute the test cases in some order which helps to detect the maximum faults by consuming less cost and time. The test case prioritization technique schedules the execution of test cases in an order that attempts to increase their effectiveness in meeting some performance goal [7]. Test case prioritization techniques mainly order test cases according to some criteria that aim to increase the rate of fault detection or maximize the code coverage. A lot of work has been published to prioritize

DOI: 10.24507/icicelb.13.08.869

the test cases of the developed software. The researchers presented various test case prioritization techniques to prioritize test cases to perform the effective system testing of the developed software. In this paper, a multilevel system test case prioritization technique has been presented. The proposed approach works at three levels. Firstly, the requirement is prioritized. At the second level, the modules of the prioritized requirements are prioritized. At the third level, the test cases of the prioritized modules are prioritized. To prioritize the requirement, modules and test cases, some factors have been considered. For experimental verification and validation, the proposed approach has been applied on the two software implemented in Java and C++. The finding of the experiments shows the efficacy of the proposed approach in resultant to reduce the testing cost.

2. **Related Works.** Meçe et al. [8] investigated the application of the machine learning techniques to prioritize the test cases. They found that the machine leaning techniques are used to solve the test case prioritization problem. They surveyed the most recent studies in this area and discussed the various techniques, data used to prioritize the test cases, metrices to determine the effectiveness of the proposed approaches, etc. Mahdieha et al. [9] proposed techniques to enhance the coverage-based test case prioritization approaches. In the presented approach, they used the bug history of the implemented software to devise the defect prediction method that was further used to learn a network model. They considered the coverage based on the faults. Zhang et al. [10] presented a unified modeling language model-based test case prioritization technique. The proposed techniques first estimate the probability of the error using C&K metrics. The severity of the errors is estimated using the dependencies based on model slicing. The priority of the test cases is determined on the basis of the probability and severity of the errors. Gökçe and Eminli [11] proposed a model-based test case prioritization technique. They used the neural network classification to prioritize the test cases. In this approach, test cases are divided into five groups. The prioritizing groups label each test case as output depending on an important index weighed by membership degree and frequency of occurrences of all events belonging to the group. Spieker et al. [12] proposed a test case prioritization and selection technique in continuous integration. The objective of the proposed approach is reducing the round-trip time between the developer feedback and code commits on the failed test case. The presented approach RETECS uses the duration of the previous last execution and failure history of test cases for selection and prioritization of the test cases. Singh et al. [13] proposed a machine learning based technique to prioritize the test cases. They examined the correlation of the software quality and object-oriented metrics. They introduced the evaluation of the CK metrices. The various considered metrices are the coupling between objects, depth of inheritance tree, weighted methods per class, number of children, and response for a class (RFC). They also used four other metrics called publicly inherited methods, weighted attributes per class, number of methods inherited and number of methods overridden. The SVM is used to categorize the classes into the preferred and non-preferred classes.

Lachmann et al. [14] proposed a system test case prioritization technique using the supervised machine learning. The proposed approach used the black box meta data like the history and the natural language description of the test case to prioritize the test cases. They trained the rank classification model by applying the SVM rank algorithm. Lachmann [15] also presented machine learning-driven test case prioritization approaches for black-box software testing. The priority value of the test case is calculated by analyzing the meta data and artifacts of the natural language. In this approach, they combined the output of different machine learning algorithms for one version and the output of one algorithm of several versions. Busjaeger and Xie [16] presented a test prioritization technique that integrates the multiple existing techniques via a systematic framework of machine learning to rank. The features Java code coverage, text path similarity, text

content similarity, failure history, and test age are used by the ranking model. Srikanth et al. [17] proposed a requirement-based test prioritization technique using risk factors. They extended their earliest approach PORT 1.0 to PORT 2.0. They used two factors, customer priority (CP) and fault proneness (FP) to prioritize the test cases. From the experimental outcome, they observed that there is a strong correlation between CP and FP. Rahman and Saxena [18] proposed a fuzzy logic-based model to prioritize the test cases. Test cases are prioritized by using the modification in the system and exposure to risk. They captured the conduct of the system by using the state diagram. Musa et al. [19] presented a technique to prioritize the test cases using the analysis of dependency graph model of source code. They used a genetic algorithm that optimized the selected test cases. Ansari et al. [20] proposed an approach using ant colony optimization algorithm to prioritize the regression test cases. The proposed technique first takes the test cases which have covered the maximum faults followed by the selection of test cases that cover the remaining faults. Yoon et al. [21] proposed a technique to prioritize test cases on the basis of correlation of requirement and risk. They determined relevant test cases by computing the risk exposure value of the requirement and by analyzing risk items. Ghai and Kaur [22] proposed a test case prioritization technique using a hill climbing approach. The test cases are prioritized on the basis of their functional importance.

History value-based approach for a cost cognizant technique is used to prioritize the test cases [23]. The past history information is used to determine the cost associated with the test cases, severity of the detected faults and historical value of the test cases. A requirement based on system test case prioritization technique prioritizes the test cases using various factors [24]. The considered factors are requirement change, fault impact, completeness and reusable requirement to prioritize the test cases. Ashraf et al. [25] presented a value based practical swarm intelligence algorithm for prioritizing the test cases. They introduced the combination of six factors for performing the test case prioritization. These factors are the customer priority, requirement volatility, implementation complexity, requirement traceability, execution time and fault impact of the requirement. Chen et al. [26] compared various test case prioritization techniques in terms of the average percentage of faults detected (APFD) and APFDc. The finding result shows the various practical guidelines. Geetha et al. [27] presented a prioritization technique using the prediction of faults in acceptance testing. They used the combination of the optimized multi-level random walk and genetic algorithm. Qasim et al. [28] presented the survey related to the techniques to prioritize the test cases. They found that the 35% test case prioritization techniques are multi-objective and 20% techniques are single objective. Cheng et al. [29] proposed a technique for testing the changes in configuration. The proposed technique Ctest can generate a large number of configuration test cases automatically. The generated Ctest is able to detect the misconfigurations.

By critically reviewing the literature, it has been observed that some critical factors play an important role to perform the effective testing of the software within time and cost. Researchers have presented the various techniques to prioritize the test case by considering the various factors and using different algorithms. However, the researchers did not consider some factors that play an important role to detect the maximum faults earlier. The impact of the considered factors on the severity of detected faults by the test cases was not considered. Researchers have not discussed the viability of the factors and not provided any prediction value of the factors to detect the maximum faults in the software. So, to overcome these discrepancies in this paper, a multilevel system test case prioritization technique has been presented. At each level, some factors have been used to prioritize the requirements, modules and test cases. For experimental verification and validation, the experimented results of the proposed technique have been compared with the existing similar techniques. The comparison results showed the effectiveness of the presented technique.

3. **The Proposed Work.** The proposed approach works in three phases. In the first phase, the requirements are prioritized. In the second phase, the modules of the ordered requirement are prioritized. In the third phase, the test cases of the particular requirement are prioritized. The prioritizations of requirements, modules and test cases are performed on the basis of some factors. Every considered factor has been assigned a positive weight which is determined by using the four algorithms in SPSS. The customer, developer, business analytics, tester, etc. assign a value between 0 to 10 to the considered factors in the respective phases.

3.1. **Determination of the weight for considered factors.** For determination of the contribution weight to each factor, a set of data was collected from various projects implemented by the students. The data collected from the students are analyzed by the four algorithms using SPSS Modeler [30]. The SPSS Modeler provides the strategic technique to determine the meaningful relationship among the large set of data. The SPSS Modeler has various modeling algorithms for specific business expertise. These modeling algorithms are classification, prediction, and segmentation and association analysis. With the help of SPSS Modeler, different relationships in data are investigated by applying different models. These four algorithms are the CHAID, QUEST, C5.0 and C&R Tree [31-34]. The outcomes of all algorithms are analyzed and the contributions of all the considered factors are determined to decide the prediction of faults at the requirement, module and test case level. The average of determined importance value obtained from all algorithms is used to prioritize the requirement, module and test cases.

3.2. **Proposed process of test case prioritization.**

3.2.1. *Prioritization of the requirement.* The requirements are prioritized using the seven factors which are shown in Table 1. These factors are determined by the analysis of the software requirement specification. Every factor has been assigned a positive weight which shows the contribution to predict the occurrence of faults in requirements. Every factor assigns a value between 0 to 10 by the customer, developer, business analytics, etc., respectively. The requirements are prioritized using the calculated value of requirement prioritization value (RPV) which is calculated by Formula (1).

$$RPV = \sum_{i=1}^{n} W_i * V_i \tag{1}$$

where W is the weight of the ith factors and V is the value of assigned to the ith factors of requirement.

Table 1. Proposed factors and weight to prioritize the requirements

Sr. No	Proposed factors	Predicted weight
1	Implementation complexity	0.0925
2	Cost of change	0.0875
3	Business impact	0.0875
4	Requirement severity	0.0725
5	Requirement dependency	0.14
6	Availability of resources	0.1725
7	Customer priority	0.35

3.2.2. *Prioritization of the module.* Modules are prioritized on the basis of the four factors. These factors are impact on requirement, requirement coverage, complexity of module and module dependency. Every factor has been assigned a positive weight which is calculated by applying the four algorithms as shown in Table 2. The values of the factors are assigned by the developer, business analytics, etc., between 0 to 10. For prioritization of the modules, the value of module prioritization value (MPV) is calculated by using Formula (2).

$$MPV = \sum_{i=1}^{n} WMF_i * VMF_i \tag{2}$$

where WMF is the assigned weight to the ith factors and VMF is estimated value of the ith factor of module.

Table 2. Proposed factors to prioritize the modules

Sr. No	Proposed factors	Predicted weight
1	Impact on requirement	0.1075
2	Requirements coverage	0.255
3	Complexity of module	0.205
4	Module dependency	0.43

3.2.3. *Prioritization of the test cases.* In this phase, the test cases of the prioritized module are prioritized. Prioritization of the test cases is performed on the basis of the six factors. These factors are the execution frequency, feature covered by test case, test case effectiveness, test dependency, business impact by test case and fault detection. Every factor has been assigned a positive weight which shows the capability of detection of the faults by the test cases. The weight assigned to the factors is shown in Table 3. The values of factors are assigned by tester and business analytics between 0 to 10. The test cases are ordered on the basis of the calculated value of the test case prioritization value (TCPV). This is calculated by Formula (3).

$$TCPV = \sum_{i=1}^{n} WTF_i * VTF_i \tag{3}$$

where WTF is the assigned weight to the ith factors and VTF is estimated value of the ith factor of test case.

Table 3. Proposed factors to prioritize the test cases

Sr. No	Proposed factors	Predicted weight
1	Execution frequency	0.1375
2	Feature covered by test case	0.2975
3	Test case effectiveness	0.1575
4	Test dependency	0.1675
5	Business impact by test case	0.23
6	Fault detection	0.0175

4. **Result and Analysis.** For experimental verification and validation, the proposed approach has been applied on two software of inventory management [35] implemented in Java and library information system [36] implemented in C++ and the results are compared with the existing similar technique [17]. The considered first software has performed various operations like addition of customer, update of customer data, add,

TABLE 4. Prioritization of requirements

	Customer	Product	Supplier	Warehouse	Sales person	Invoice	Help	Logoff	Exit
Customer priority (CP)	8	8	5	7	5	8	3	5	3
Requirement dependency	8.8	8.8	5.5	5.5	5.5	3.3	0	0	0
Cost of change	8	7	5	5	5	5	0	5	0
Implementation complexity	8	8	5	7	5	5	0	5	0
Business impact by the requirement	9	9	5	8	5	9	0	0	0
Requirement severity	9	9	5	7	5	5	0	0	0
Availability of resources	5	5	5	7	5	3	0	0	0
RPV	7.7745	7.687	5.0825	6.72	5.0825	5.8295	1.05	3.0875	1.75

TABLE 5. Prioritization of modules of highest prioritized requirement

	add_edit_customer	Search	Delete	Print
Impact on requirement	9	7	5	2
Requirement coverage	1.11	1.11	1.11	1.11
Complexity of module	9	5	5	4
Module dependency	7.5	5	0.25	0.25
MPV	6.3205	4.2105	1.9530	1.4255

TABLE 6. Prioritization of modules of highest prioritized module

	TC1	TC2	TC3	TC4	TC5	TC6
Test case effectiveness	0	0	0	0	0	0
Execution frequency	9	9	3	9	3	1
Test dependency	8	8	3	9	2	1
Business impact by test case	9	9	0	9	0	1
Feature covered by test case	5	5	5	5	5	5
Fault detection	6	7	3	9	3	1
TCPV	6.24	6.25	2.4	6.46	2.32	2.04

remove, delete and update the product. To analyze the effectiveness of the proposed approach, some faults are introduced in the software, which are detected by applying the proposed approach. The outcomes of the proposed approach have been shown in Table 4.

Table 4 shows the prioritization of the requirement. The highest prioritized requirements have four modules. These modules are the add_edit_customer, search customer, print, and delete the customer. The prioritization process of the modules using contribution weight is shown in Table 5.

The prioritized order of the modules is add_edit_customer, search customer, delete customer and print. Now the test cases of the highest prioritized modules are prioritized. Table 6 shows the prioritization of the test cases of the add_edit_customer module.

On the basis of the obtained value of the TCPV, the execution order of the test cases of the add_edit_customer module is TC4, TC2, TC1, TC3, TC5, and TC6. The graph in Figure 1 shows the average percentage of faults detected (APFD) of the proposed approach, non-prioritized approach and the PORT 2.0 approach [17].

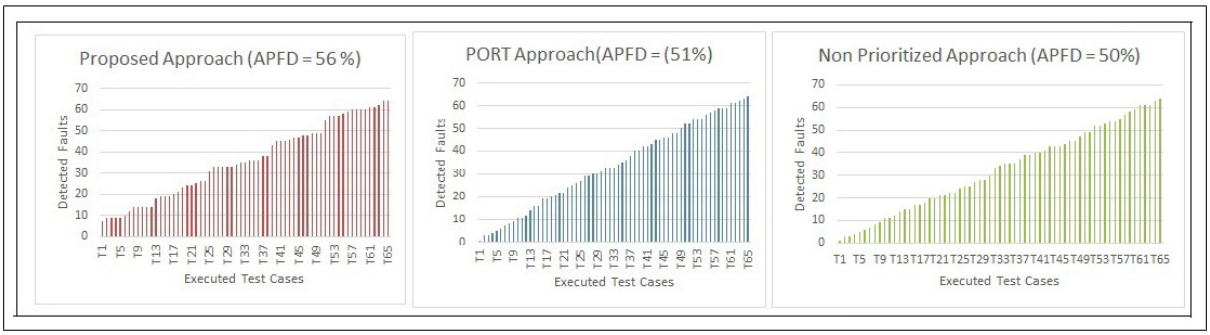

FIGURE 1. APFD graph of non-prioritized, PORT 2.0 and proposed approach

The same approach was applied on the second considered software library information system implemented in the C++ programming language. The considered software performs the various functions like acquisition of books, membership maintenance, book issue, book return, renewal of membership and answer management queries. For experimental verification, 98 faults have been introduced intently in the considered software which are detected using 111 test cases. The experimented results are shown in Figure 2. APFD values of non-prioritized, PORT 2.0 [17] and the proposed approaches of two case studies are shown in Table 7.

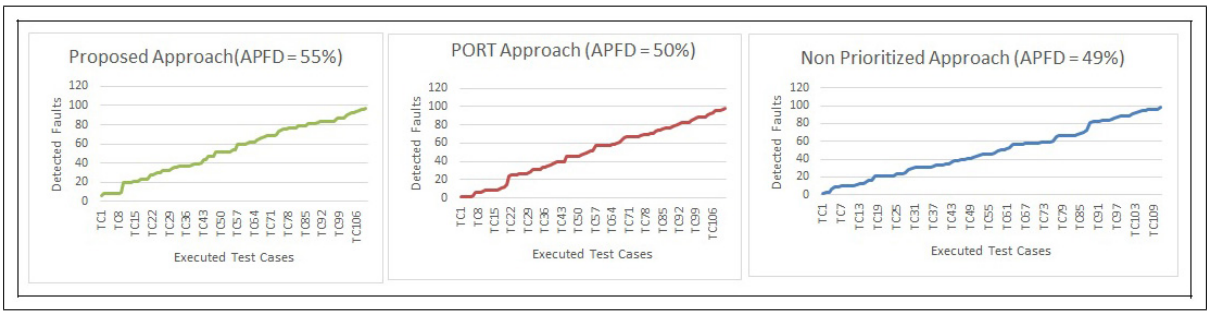

FIGURE 2. APFD graph for non-prioritized, PORT 2.0 and proposed approach for library information system

TABLE 7. APFD values of the non-prioritized, PORT 2.0 and proposed approach

Sr. No.	Name of approaches	APFD case study 1	APFD case study 2
1	Non-prioritized approach	50%	49%
2	PORT 2.0 approach	51%	50%
3	Proposed approach	56%	55%

5. **Conclusion.** In this paper, a system test case prioritization technique has been presented. The proposed approach works at multiple levels. Firstly, the requirement is prioritized. At the second level, the modules of the prioritized requirements are prioritized. At the third level, the test cases of the prioritized modules are prioritized. The prioritizations of the requirement, modules and test cases have been performed on the basis of some factors. Each factor has assigned a positive weight which can be computed by using the four algorithms in SPSS Modeler. These four algorithms are the CHAID, QUEST, C5.0 and C&R Tree. A survey has been performed to collect the data to compute the weight of factors. For experimental verification and validation, the proposed approach has been applied on the two software implemented in Java and C++. The finding of the experiments shows the efficacy of the proposed approach in resultant to reduce the testing cost and time.

REFERENCES

[1] https://freshcodeit.com/, 2021.

[2] https://www.xenonstack.com/insights/what-is-software-quality, 2021.

[3] https://www.testim.io/blog/test-environment-guide/, 2021.

[4] A. Samad, H. B. Mahdin, R. Kazmi, R. Ibrahim and Z. Baharum, Multiobjective test case prioritization using test case effectiveness: Multicriteria scoring method, *Soft Computing Approaches to Continuous Software Engineering*, DOI: 10.1155/2021/9988987, 2021.

[5] N. Chauhan, *Software Testing Principles and Practices*, Oxford University Press, 2010.

[6] https://www.statista.com/statistics/500641/worldwide-qa-budget-allocation-as-percent-it-spend/, 2021.

[7] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, Prioritizing test cases for regression testing, *IEEE Trans. Software Engineering*, vol.27, no.10, pp.929-948, 2001.

[8] E. K. Meçe, H. Paci and K. Binjaku, The application of machine learning in test case prioritization – A review, *European Journal of Electrical and Computer Engineering (EJECE)*, vol.4, no.1, 2020.

[9] M. Mahdieha, S.-H. Mirian-Hosseinabadia, K. Etemadia, A. Nosratia and S. Jalalia, Incorporating fault-proneness estimations into coverage-based test case prioritization methods, *Information and Software Technology*, vol.121, DOI: 10.1016/j.infsof.2020.106269, 2020.

[10] T. Zhang, X. Wang, D. Wei and J. Fang, Test case prioritization technique based on error probability and severity of UML models, *International Journal of Software Engineering and Knowledge Engineering*, vol.28, no.6, pp.831-844, 2018.

[11] N. Gökçe and M. Eminli, Model-based test case prioritization using neural network classification, *Computer Science & Engineering an International Journal (CSEIJ)*, vol.4, no.1, pp.15-25, 2014.

[12] H. Spieker, A. Gotlieb, D. Marijan and M. Mossige, Reinforcement learning for automatic test case prioritization and selection in continuous integration, *Proc. of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA2017)*, Santa Barbara, CA, USA, pp.12-22, 2017.

[13] A. Singh, R. K. Bhatia and A. Singhrova, Machine learning based test case prioritization in object oriented testing, *International Journal of Recent Technology and Engineering*, vol.8, no.3, 2019.

[14] R. Lachmann, M. Nieke, C. Seidl, I. Schaefer and S. Schulze, System-level test case prioritization using machine learning, *The 15th IEEE International Conference on Machine Learning and Applications*, 2016.

[15] R. Lachmann, Machine learning-driven test case prioritization approaches for black-box software testing, *The European Test and Telemetry Conference*, 2018.

[16] B. Busjaeger and T. Xie, Learning for test prioritization: An industrial case study, *Proc. of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'2016)*, Seattle, WA, USA, 2016.

[17] H. Srikanth, C. Hettiarachchi and H. Do, Requirements based test prioritization using risk factors: An industrial study, *Information and Software Technology*, DOI: 10.1016/j.infsof.2015.09.002, 2015.

[18] W. Rahman and V. Saxena, Fuzzy expert system based test case prioritization from UML state machine diagram using risk information, *I. J. Mathematical Sciences and Computing*, vol.1, pp.17-27, DOI: 10.5815/ijmsc, 2017.

[19] S. Musa, A.-B. Md Sultan, A.-A. B. Abd-Ghani and S. Baharom, Software regression test case prioritization for object-oriented programs using genetic algorithm with reduced-fitness severity, *Indian Journal of Science and Technology*, vol.8, no.30, DOI: 10.17485/ijst/2015/v8i30/86661, 2015.

[20] A. Ansari, A. Khan, A. Khan and K. Mukadam, Optimized regression test using test case prioritization, *Proc. of Computer Science*, vol.79, pp.152-160, 2016.

[21] M. Yoon, E. Lee, M. Song and B. Choi, A test case prioritization through correlation of requirement and risk, *Journal of Software Engineering and Applications*, vol.5, pp.823-835, DOI: 10.4236/jsea.2012.510095, http://www.SciRP.org/journal/jsea, 2012.

[22] S. Ghai and S. Kaur, A hill climbing approach for test case prioritization, *International Journal of Software Engineering and Its Applications*, vol.11, no.3, pp.13-20, DOI: 10.14257/ijseia.2017.11.3.0, 2017.

[23] H. Park, H. Ryu and J. Baik, Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing, *The 2nd International Conference on Secure System Integration and Reliability Improvement*, 2008.

[24] R. Kavitha and N. Sureshkumar, Requirement based test case prioritization with equal weightage for factors, *International Conference on Mathematical Computer Engineering (ICMCE)*, 2013.

[25] E. Ashraf, T. A. Khan, K. Mahmood and S. Ahmed, Value based PSO test case prioritization algorithm, *International Journal of Advanced Computer Science and Applications*, vol.8, no.1, 2017.

[26] J. Chen, Y. Lou and L. Zhang, Optimizing test prioritization via test distribution analysis, *Proc. of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE2018)*, pp.656-667, DOI: 10.1145/3236024.3236053, 2018.
[27] U. Geetha, S. Sankar and M. Sandhya, Acceptance testing based test case prioritization, *Cogent Engineering*, vol.8, DOI: 10.1080/23311916.2021.1907013, 2021.
[28] M. Qasim, A. Bibi, S. J. Hussain, N. Z. Jhanjhi, M. Humayun and N. U. Sama, Test case prioritization techniques in software regression testing: An overview, *International Journal of Advanced and Applied Sciences*, vol.8, no.5, pp.107-121, 2021.
[29] R. Cheng, L. Zhang, D. Marinov and T. Xu, Test-case prioritization for configuration testing, *Proc. of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA2021)*, pp.452-465, DOI: 10.1145/3460319.3464810, 2021.
[30] https://www.ibm.com/products/spss-modeler, 2021.
[31] http://www.public.iastate.edu/∼kkoehler/stat557/tree14p.pdf, 2018.
[32] ftp://public.dhe.ibm.com/software/analytics/spss/support/Stats/Docs/Statistics/Algorithms/13.0/TREE-QUEST.pdf, 2018.
[33] https://cran.r-project.org/web/packages/C50/vignettes/C5.0.html, 2021.
[34] http://www.statsoft.com/Textbook/Classification-and-Regression-Trees, 2018.
[35] https://github.com/, 2021.
[36] R. Sahoo, *C++ Projects*, Khanna Book Publishing Co. Pvt. Ltd., 2000.