ICIC Express Letters
Part B: Applications
Volume **13**, Number **1**, January **2022**

ICIC International ©2022 ISSN 2185-2766
pp. **81–88**

A STRUCTURAL ANALYSIS BASED TEST CASE 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
[*]Corresponding author: ved_ymca@jcboseust.ac.in; nareshchauhan19@jcboseust.ac.in

Received January 2021; accepted April 2021

Abstract. *The size of test suite increases as the software evolves. Due to time, resource and budget constraints, it is imperative to prioritize the execution of test cases so as to increase the possibility of early detection of faults. In this paper, for prioritizing the test cases, eight factors have been proposed. The proposed factors are determined by structural analysis of the object oriented software. The proposed technique creates the intermediate representation of the software. Further, by analyzing the intermediate representation, the independent paths are identified and mapped with the test cases which execute the independent paths. Test cases are prioritized on the basis of the coverage of the factors. For experimental evaluation and analysis, the presented approach was applied on two software modules showing the efficacy of the proposed approach.*
Keywords: Object oriented testing, Test case prioritization, TCPOOS, Factor oriented test case prioritization, Structural based test case prioritization

1. Introduction.

Test case prioritization technique has become a very effective technique to detect the faults as earlier as possible. Prioritization of test cases can be performed at various stages like potential of fault detection, statement coverage and branch coverage. Due to large functionality of the software, there is large test suite to test the software. It is not necessary that every test case incurs a fault. For executing all the test cases, testing team requires more resources and time, thereby increasing the cost of the testing. Hence, due to testing, the project may go out of budget or may get delayed. The order of test cases also affects the process of testing and it also helps in reducing the cost of testing of project. The cost to fix the bug in early stages incurs less cost as compared to fixing the bug at later stages. It may be possible that earlier test cases reported the entire bugs that are also reported by the test cases which are executed later.

In this paper, a technique for test case prioritization is presented. In the presented approach, the software is converted to an intermediate representation which is used to find all the independent paths of the software. The complexity of each of the independent paths is determined by using some factors which are identified by structural analysis of the software. These considered factors have been assigned weights which show the probability of the factors to introduce the error in software. The more the complexity of the path, the higher the chance of the bug to be reported by the test cases executing the corresponding independent path. For experimental analysis, the approach was applied to some Java programs. The result shows the efficacy of the proposed approach. The presented paper has been summarized in five sections. In Section 1, the literature review has been presented. In Section 2, discussions about the survey and considered factors have been

DOI: 10.24507/icicelb.13.01.81

given. Sections 3 and 4 give the proposed work and result after applying the approach that has been presented followed by the conclusion in Section 5.

Related work. Shahid and Ibrahim [1] showed that test cases that cover more methods have the higher probability to detect faults earlier. By using the code coverage in the testing, they can be leveraged for additional gain through prioritization. Gupta and Gustafson [2] applied the class dependency model (CDM) on object oriented programs. They analyzed the CDM to determine where the faults are concentrated in the hierarchy of the testing order of class. Panigrahi and Mall [3] presented a model based TCP which represents the objects relations. They consider the affected elements of program as well as the elements which are indirectly tested by test case for prioritizing the test cases. They [4] also presented a technique that is based on the analysis of dependency model of the source program. The union of forward slicing corresponding to each change in model is used to determine the affected nodes. The test cases are selected on the basis of covering the affected nodes and further prioritized on the basis of weight assigned to the affected nodes. Musa et al. [5] presented a technique based on analysis of dependency graph model and used the genetic algorithm to optimize the selected test cases. The test cases are ordered by computing the fitness value using the previous history of fault severity. Belli et al. [6] used unsupervised neural network and fuzzy c-means clustering algorithm to make the preference group. They order the test case using the degree of their preference. The preference degree is determined of each test case by computing mean of clustering of event using the 13 attributes. Sultan et al. [7] presented a test case prioritization technique using the dependence graph and genetic algorithm. Chen et al. [8] proposed a clustering based adaptive random sequence technique to prioritize the test cases. They used the Moclustering_means, MOClustering_medoids and DMClustering. Yadav and Dutta [9] have also used the K-mean clustering approach to prioritize the test cases of the object oriented software. Bello et al. [10] proposed cost-cognizant test case prioritization technique for object oriented software. They used the path based integration testing to determine the feasible execution paths and extract these paths from the java system dependency graph using forward slicing. Mohd-Shafie et al. [11] used the selective and even-spread count-based methods with scrutinized ordering criterion to prioritize the test cases. Rahman and Sexsena [12] proposed a fuzzy logic based model for prioritizing the test cases. They used system state diagram and risk information associated with the test cases.

Shram and Sehgal [13] proposed a technique to prioritize the test cases using the bat algorithm. They also compared their results with the results of other algorithms like the ant colony optimization, and greedy algorithm and found the effectiveness of the proposed approach. Afzal et al. [14] used the complexity of path to prioritize the test cases. The complexity of the path is determined using the Haltead's metric. Rehman et al. [15] used the historical data to prioritize the dissimilar test cases. Kumar and Mathew [16] analyzed the software to build a system dependency graph based model. The model is used to generate the test cases and determine the state of a program can be saved or refused for executing another test cases. The structural complexity is used to prioritize the test cases. The complexity [17] of method is also used to prioritize the test cases. The complexity of the method is computed by using some factors.

A critical study of above literature indicates that the researchers focus on identifying the various factors and new technique of test case prioritization which helps to provide quality software in minimum efforts. They used the factors like methods, and object relations. However, it has been observed that there should be some program structure related factors which are used to prioritize the test cases with the goals to detect the maximum faults as earlier as possible and reliable software. In this paper, a structured factors based test case prioritization technique for object oriented software is presented.

2. **Survey to Find the Weight to the Proposed Factors.** All the proposed factors have been assigned a weight on the basis of possibility of faults introduced by the factors. To verify and assign the significant weight of factors, a survey has been performed. The conducted survey focuses on factors, which affect the testing of the software. The participants involved in survey are the software developer, tester, tech lead, etc., having average experience of 8 years in software industries. To examine the view of participants, the survey questionnaire was submitted among several software developers and testers in various software industries.

Based on the criticality of the factors, a weight is assigned to the proposed factors. The assigned weight shows the capability of introducing the errors in the program. The weight metrics of the proposed factors are as shown below in Table 1.

Table 1. Proposed factors and assigned weight

S. No	Factor	Weight
1	Class/Interface	.05
2	Type casting	.15
3	Exception handling	.3
4	Method overriding	.2
5	Native method	.1
6	Nested class	.05
7	Conditional statements	.05
8	Number of methods	.1

3. **Proposed Work.** The proposed approach works at three levels. At the first level, intermediate representation of the program objected oriented control flow graph (OOCFG) is created by analyzing the structure of the program of the graph. At the second level, by analyzing the OOCFG graph, all the independent paths of a program are determined and thereby test cases are selected corresponding to every independent path. Finally, at the third level, the test cases are prioritized on the basis of coverage of factors.

3.1. **Representation of the program in the intermediate form.** In this section, program is represented in the intermediate representation. For representation of program, some symbolic notations are presented which are shown in Figure 1. The intermediate representation shows execution flow of the program. Since the program structures of object oriented program are different from the convectional program, here some representations are presented which represent the features of the OOP, i.e., class, interface, method, method overriding, nested class, exception handling, etc.

3.2. **Identification of independent path.** By using the representations showing in Figure 1, the OOCFG of program is created which are further analyzed to identify all the independent paths. After determining all the independent paths are mapped to test cases.

3.3. **Test case prioritization.** Mapped test cases are prioritized on the basis of the proposed 8 factors. The test cases are prioritized on the basis of the coverage of the factors. Test case with the highest coverage value has the highest priority of execution as these factors show the criticality of the test case based on coverage of factors. Thus, a test case with the highest criticality will have the higher probability of error to be found out.

By using Table 1, these test cases are prioritized using Formula (1)

$$\text{TCPW} = \sum_{i=1}^{N} fvalue_{ij} * fweight_j \tag{1}$$

where *fvalue* is the values of factors covered by test cases, *fweight* is the weight assigned to the factor which shows the criticality of the factor, and TCPW is the calculated weight of the test cases. On the basis of TCPW, the test cases are prioritized. The more the complexity of the test cases, the more the probability of the error to be detected by test cases.

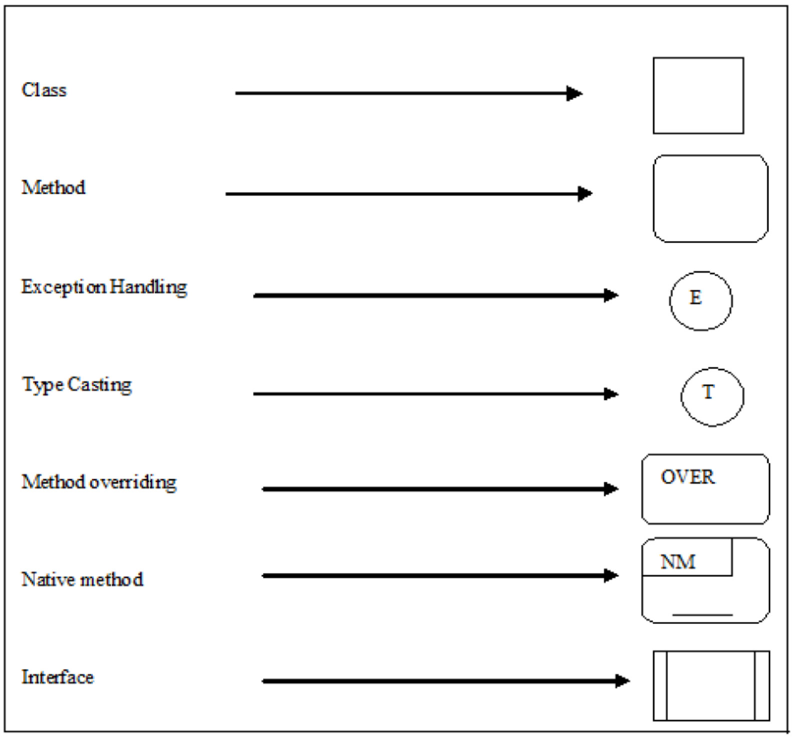

FIGURE 1. Representation of various features

4. **Result and Analysis.** For evaluation and analysis of the proposed approach, it has been verified and analyzed by applying on a case study of software. The considered case study performs the various functionalities like to calculate the gross salary, saving, deduction, taxable income, and tax to be paid by the employee. To determine the efficacy of the proposed approach, some faults have been added in the software intentionally and compared with code coverage (CC) [1] and path complexity (PC) [14] based test case prioritization techniques. The OOCFG of considered case study is shown in Figure 2.

After analyzing, the above graph independent paths are determined. To test the considered software, each and every independent path needs to be tested. So test cases should be selected or designed for each independent path. All the independent paths and IDs of test case are shown in Table 2.

After determining the independent paths and mapping the test cases corresponding to all paths, now test cases are prioritized. Table 3 shows the factors covered by the test cases. The test cases are prioritized on the basis of TCPW obtained by applying Formula (1). The highest the value of TCPW of the test cases, the highest the priority of the test case to be executed. Table 4 shows the TCPW of the all selected test cases. By using the calculated TCPW of each test case from Table 4, the prioritized order of the test cases is TC10, TC11, TC5, TC6, TC3, TC4, TC8, TC9, TC1, TC2, TC7.

The APFD (average percentage of faults detected) graph shown in Figure 3 shows that the APFD value obtained from the proposed approach is better than the other similar approaches. The result shows the efficacy of the proposed approach.

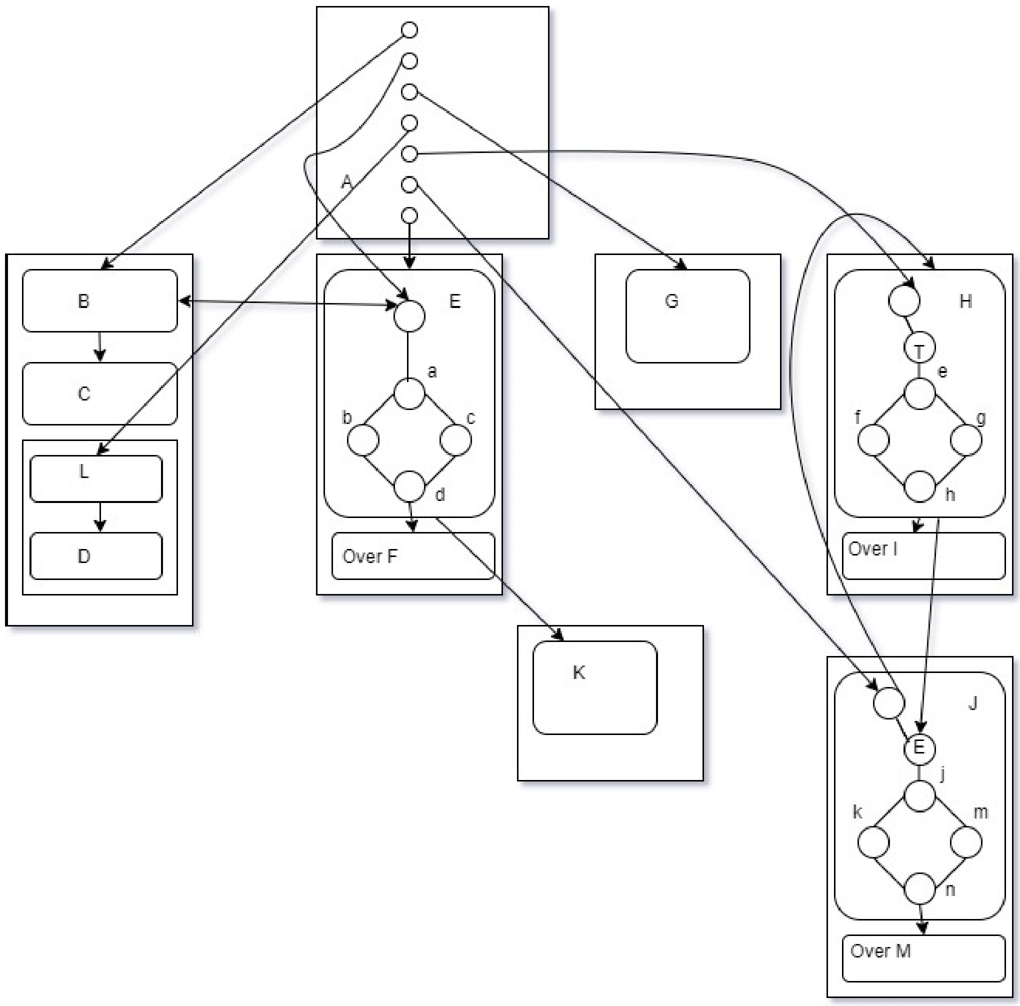

Figure 2. OOCFG of considered case study

Table 2. Independent path and test corresponding to the independent paths

S. No.	**Independent path**	**Test case ID**
1	A, B, C	TC1
2	A, L, D	TC2
3	A, E, B, Ea, Eb, Ed, F	TC3
4	A, E, B, Ea, Ec, Ed, F	TC4
5	A, E, B, Ea, Eb, Ed, K	TC5
6	A, E, B, Ea, Ec, Ed, K	TC6
7	A, G	TC7
8	A, H, He, Hf, Hh, I	TC8
9	A, H, He, Hg, Hh, I	TC9
10	A, j, H, J, Jj, Jk, Jm, M	TC10
11	A, j, H, J, Jj, Jk, Jn, M	TC11

The same approach was applied on software banking information system [19] that is implemented in the C++. The APFD graph of comparison of the proposed approach, non-prioritized approach and other existing similar approaches [1,14] is shown in Figure 4.

Table 5 shows the APFD obtained from the proposed approach and other approaches for discussed case studies.

Table 3. Factors covered by the test cases

S. No	Factors to be covered	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10	TC11	Weight of factors
1	No. of classes	2	2	3	3	4	4	2	2	2	3	3	.05
2	No. of nested classes	0	1	0	0	0	0	0	0	0	0	0	.05
3	No. of methods	2	2	3	3	4	4	1	1	1	2	2	.1
4	No. of override methods	0	0	1	1	0	0	0	1	1	1	1	.2
5	Exception handling	0	0	0	0	0	0	0	0	0	1	1	.3
6	Type casting	1	0	1	1	1	1	0	1	1	1	1	.15
7	No. of native methods	0	0	0	0	0	0	0	0	0	0	0	.1
8	Conditional statement	0	0	1	1	1	1	0	1	1	1	1	.05

Table 4. Calculated value of TCPW

S. No	Factors to be covered	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10	TC11	Weight of factors
1	No. of classes	.1	.1	.15	.15	.2	.2	.1	.1	.1	.15	.15	.05
2	No. of nested classes	0	.05	0	0	0	0	0	0	0	0	0	.05
3	No. of methods	.2	.2	.3	.3	.4	.4	.1	.1	.1	.2	.2	.1
4	No. of override methods	0	0	.2	.2	.2	.2	0	.2	.2	.2	.2	.2
5	Exception handling	0	0	0	0	0	0	0	0	0	.3	.3	.3
6	Type casting	.15	0	.15	.15	.15	.15	0	.15	.15	.15	.15	.15
7	No. of native methods	0	0	0	0	0	0	0	0	0	0	0	.1
8	Conditional statement	0	0	.05	.05	.05	.05	0	.05	.05	.05	.05	.05
	TCPW	.45	.35	.85	.85	1.0	1.0	.2	.6	.6	1.05	1.05	1

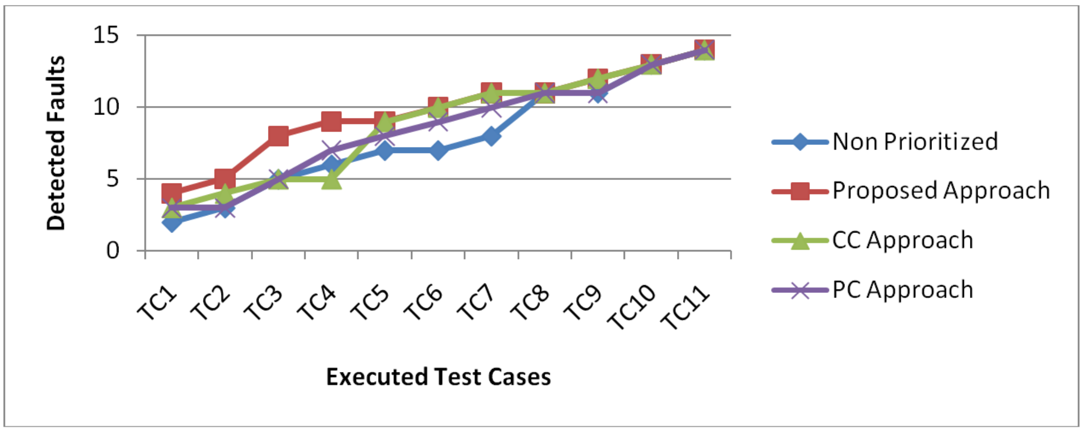

Figure 3. Comparisons between the proposed approach and other approaches

5. **Conclusion.** In this paper, a novel approach for test case prioritization technique is presented. It is observed from the related work that researchers are not using structured based factors and some researchers considered dependency model, object relation, coverage of method, prior information of the faults, etc. The presented approach prioritized the test cases on the basis of eight program structure factors. The factors are identified by structural analysis of the program. Some symbolic notations are presented in the approach which are used to convert the source program into intermediate representation which help to identify independent path and structured factors which pose the higher

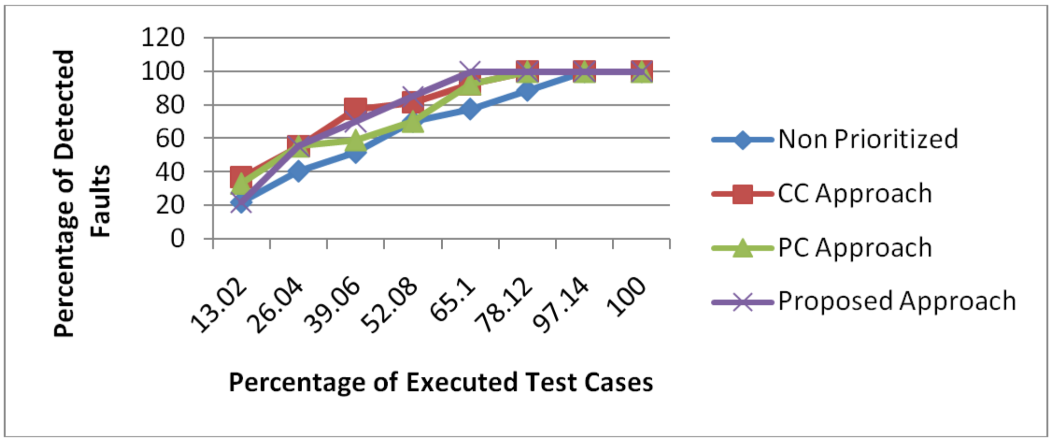

Figure 4. Comparison between the proposed approach and other approaches

Table 5. APFD results of case studies

Applied strategy	APFD results (case study 1)	APFD results (case study 2)
Non prioritized	52%	60.62%
Path complexity based approach	56%	68.9%
Code coverage based approach	58%	72.7%
Proposed approach	64%	73.9%

chance of the faults. The applicability of the factors and their assigned weights are verified by performing the survey in reputed software industries which work in various biggest projects. Structural analysis of the source program helps to provide reliable and quality software. The result and comparison of the proposed approach with other existing similar approaches show the presented approach is effective to find the maximum faults as earlier as possible, helps to reduce the testing cost, time and provides the reliable software. To enhance the effectiveness of the proposed approach, the new factors may be added by analyzing the industry data and applying the proposed approach on the industry project.

REFERENCES

[1] M. Shahid and S. Ibrahim, A new code based test case prioritization technique, *International Journal of Software Engineering and Its Application*, vol.8, no.6, pp.31-38, 2014.

[2] P. Gupta and D. A. Gustafson, Analysis of the class dependency model for object-oriented faults, *International Journal of Advances in Engineering & Technology*, 2012.

[3] C. R. Panigrahi and R. Mall, Test case prioritization for object oriented programs, *SETLabs Briefings*, vol.9, no.4, 2011.

[4] C. R. Panigrahi and R. Mall, A heuristic-based regression test case prioritization approach for object-oriented programs, *Innovations in Systems and Software Engineering*, vol.10, pp.155-163, 2014.

[5] S. Musa, A.-B. M. 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.

[6] N. Gökçe, F. Belli, M. Eminli and B. T. Dincer, Model-based test case prioritization using cluster analysis: A soft-computing approach, *Turkish Journal of Electrical Engineering & Computer Sciences*, vol.23, no.3, pp.623-640, DOI: 10.3906/elk-1209-109, 2015.

[7] A. B. M. Sultan, A. A. A. Ghani, S. Baharom and S. Musa, An evolutionary regression test case prioritization based on dependence graph and genetic algorithm for object oriented programs, *The 2nd International Conference on Emerging Trends in Engineering and Technology*, London, UK, 2014.

[8] J. Chen, L. Zhu, T. Y. Chen, D. Towey, F.-C. Kuob, R. Huang and Y. Guo, 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.

[9] D. K. Yadav and S. K. Dutta, Test case prioritization using clustering approach for object oriented software, *International Journal of Information System Modeling and Design*, 2019.

[10] A. Bello, A. B. M. Sultan, A. A. A. Ghani and H. Zulzalil, Evolutionary cost-cognizant test case selection and prioritization for object-oriented programs, *International Journal of Engineering and Advanced Technology (IJEAT)*, vol.8, no.6S3, 2019.

[11] M. L. Mohd-Shafie, W. M. N. Wan-Kadir, M. Khatibsyarbini and M. A. Isa, Model-based test case prioritization using selective and even-spread count-based methods with scrutinized ordering criterion, *PLoS ONE*, https://doi.org/10.1371/journal.pone.0229312, 2020.

[12] W. Rahman and V. Sexsena, Fuzzy expert system based test cases prioritization from UML state machine diagram using risk information, *International Journal of Mathematical Sciences and Computing*, vol.3, no.1, pp.17-27, 2017.

[13] A. Shram and N. Sehgal, Enhanced test case prioritization technique using bat algorithm, *International Journal of Advance Research, Ideas and Innovations in Technology*, vol.4, no.2, 2018.

[14] T. Afzal, A. Nadeem, M. Sindhu and Q. uz Zaman, Test case prioritization based on path complexity, *International Conference on Frontiers of Information Technology (FIT)*, 2019.

[15] M. A. Rehman, M. A. Hasan and M. S. Siddik, Priotizing dissimilar test cases in regression testing using historical data, *International Journal of Computer Applications, Foundation of Computer Science (FCS)*, vol.180, 2018.

[16] V. Kumar and S. Mathew, Test case prioritization and distributed testing of object oriented program, *Turkish Journal of Electrical Engineering & Computer Sciences*, 2019.

[17] Vedpal and N. Chauhan, Test case prioritization technique for object oriented software using method complexity, *International Journal of Innovative Computing, Information and Control*, vol.14, no.1, pp.341-354, 2018.

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

[19] *http://cppprojectcode.blogspot.com/.*