International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-12, October 2019

Efficiency Enhancement in Regression Test Case Prioritization Technique

Soumen Nayak, Chiranjeev Kumar, Sachin Tripathi, Lambodar Jena

Abstract: ***Regression testing is an important, but expensive, process that has a powerful impact on software quality. Unfortunately all the test cases, existing and newly added, cannot be re-executed due to insufficient resources. In this scenario, prioritization of test case helps in improving the efficacy of regression testing by arranging the test cases in such a way that the most beneficial (that has the potential to detect the more number of faults) are executed first. Previous work and existing prioritization techniques, though detects faults, but there is a need of improved techniques to enhance the process of regression testing by improving the fault detection rate. The new technique, proposed in this paper, gives improved result than the existing ones. The comparison of the effectiveness of the proposed approach is done with other prioritization and non-prioritization orderings. The result of the proposed approach shows higher average percentage of faults detected (APFD) values. Also, the performance is evaluated and it is observed that the capability of the proposed method outperforms other algorithms by enhancing the fault detection rate.***

Keywords: Software testing, regression testing, test case prioritization, average percentage of faults detected metric.

I. INTRODUCTION

Testing software mainly deals with errors, defects, failures, and incidents. It is the process said to be successful if it can showcase an as-yet-undiscovered error [1]. The prime aim of testing is to uplift the software quality by detecting important bugs. At the same time the cost and effort should be minimized and also the product delivery should be on time. The test suite once used, are often, preserved for future reuse. The test cases (TCs) play a pivotal role in testing process. In constraint resource scenarios, like time, manpower, effort, computational capability of machine etc.,exhaustive testing is not possible where each and every test case will be implemented since the input domain is vast. Hence, effective testing is preferred over the extensive ones in the software industry which is often exhaustive. The process should be planned, itemized, drafted, implemented, well-documented and quality goals should be quantified.

Revised Manuscript Received on October 30, 2019.
***** Correspondence Author
Soumen Nayak, Department of Computer Science and Engineering, Indian Institute of Technology(ISM), Dhanbad- 826004, India. Email: {*soumen.nayak@gmail.com, jlambodar@gmail.com}
Chiranjeev Kumar, Department of Computer Science and Engineering, Indian Institute of Technology(ISM), Dhanbad- 826004, India.
Sachin Tripathi, Department of Computer Science and Engineering, Indian Institute of Technology(ISM), Dhanbad- 826004, India
Lambodar Jena*, Department of Computer Science and Engineering, Siksha O Anusandhan (Deemed to be University) Bhubaneswar, India Email: {*soumen.nayak@gmail.com, jlambodar@gmail.com}

When there is a modification of the software or any new functionality are added to the existing software as a part of integration testing, the software under test (SUT) changes. The new modules may affect the functioning of earlier integrated modules which otherwise were executing flawlessly. It means the new modifications may affect other part of the SUT. Whenever there is software configuration change, there is a probability of other modules getting affected due to bug-fixing. So there is a need to test the whole software again with all the test cases so that a new modification in a certain part does not upset the other portions of the software. It is done in regression testing by selectively retesting a system or component to check that the changes made have not induced any ill-effects and that it still obeys its requirement specifications [2]. It detects faults and it performs tasks in the maintenance level. It instils confidence in the software quality by ensuring the correctness of program and tracking the output. It acts as a feedback for the tester so that early detection of faults is possible and the propagation of error can be halted.

Software testing is a continuous process that starts from project inception and last up to the death of the software. In other words, it took place throughout the lifecycle of the project. The existing test cases are often utilized to test an altered program. Most of the times it is not sufficient to test all the modules with the existing test cases, so new test cases are appended with the existing one to thoroughly test all aspects of software, both functional and non-functional. As the software evolves, the test suite size increases drastically. It is essential to schedule the test case execution to increase the probability of quick detection of bugs because of the resource constraints.

In test case prioritization (TCP) as mentioned in [3][4], the proposed techniques schedule intently the test cases in a manner that boost some testing criterion like targeting code coverage as swiftly as possible, position of high–risk entities like faults, improving fault detection capabilities, and likeliness of identifying errors in regression. It also affect overrun of budget, lags in release of the product and also the slippage of errors into the products that have already been delivered. The test cases are arranged by the test engineers in a fashion that the high priority test cases will execute earlier than others. The test cases are never been eliminated by prioritization techniques, rather it removes the disadvantages of test case selection and/or test case minimization when they dispose test cases. It can increase the probability of effective usage of testing time in case regression test suite (RTS) is terminated abruptly than if prioritization is not imposed on those test cases. The goal of ordering is to rearrange the test cases are such that aims to select the most appropriate test cases that can surface out the errors as soon as possible, based on some rational, non-arbitrary criteria.

The testing criteria can be different and is beneficial for identifying TCs that are possessing different structures, functions and non-function elements in SUT. Therefore, there is a need for an effective metric that has the potential to enhance the potency of the test cases by improving fault detection rate. In this paper to fulfil the above mentioned need, the metric, effectiveness of test case in detecting faults (ETDF), is proposed by virtue of which arrangement of test cases have been done. This arrangement has got the potential to detect maximum faults at the beginning of testing phase. This ordering is compared with other non-ordering and existing ordering techniques and their performances are studied with the help of APFD values and graphs. Our proposed ordering method outperforms all others.

The rest of the paper is classified as follows. In Section 2, the related work on test case prioritization is presented. Section 3, describes the test case prioritization problem. Section 4 describes the proposed work. In Section 5, the experimental evaluation and discussion are dealt. Finally, Section 6 describes the concluding remarks by discussing the outcomes in a nutshell and discusses future enhancements.

II. RELATED WORK

Regression TCP typically arranges the existing test cases that required to be executed in a fashion that can locate errors or faults as quick as possible during regression testing with less time and effort. The highest priority test cases according to some testing goals should be run before the lower priority test cases in the regression testing process. Several researches have been done to find the effective metrics and techniques for prioritization but there is still a quest for efficient prioritization technique that could save a huge chunk of capital and effort. Prior research [5][6] throws light on the fact that the time required for execution for prioritizing the test cases is quite long due to voluminous test suites and several operations on a number of metrics are needed to achieve higher APFD values. In this work, a single efficient metrics is proposed along with an efficient algorithm that can have higher APFD values than the existing prioritization techniques.

Many researchers used the coverage as a metric for prioritization criteria for finding faults earlier in the testing phase. It is based on the fact that the chance of revealing fault is more if more the coverage is achieved by the test cases. Initially, Wong et al. [4] prioritize the test cases with the concept that the cost increases per additional coverage. But the author has not mentioned one probable goal of prioritization is surfacing bugs as soon as possible. More vividly, Rothermel et al.[7] proposed nine coverage based techniques that has potential to reveal faults. The results are computed and compared using Aristotle program analysis system tool that can provide information about control-flow graph and test coverage. The analysis of data shows that the techniques used for TCP can really enhance the process of locating faults than the untreated one and also ascertains certain disadvantages among different prioritization techniques. The work of Rothermel et al. is elaborated by Elbaum et al. [6] for addressing the version specific prioritization. They have compared 12 function based techniques and 4 statement based techniques. The result shows that fine-granularity techniques subdued the coarse-granularity techniques.

Modification based approach aims at prioritizing the test cases by virtue of the changes made to the software. Korel et al.[8][9] proposed two techniques for prioritization, namely system model based on its behaviour and model dependence prioritization techniques. For both the techniques, the effectiveness of prioritization is improved by using fault detection rate metrics. Seven methods have been proposed and compared in the second paper and the result shows model dependence based and heuristic based technique exhibit best effectiveness. The later one requires less information. Fraser and Wotawa [10] described a model-based prioritization technique which is based on property relevance.

Fault based prioritization deals with not only on the execution of test cases but also on fault probability that can cause failure for those test cases [3][11]. R. Krishnamoorthi et al. [12] proposed a requirement based techniques where they organized the system test cases using six parameters. There is an improvement in locating the severe fault detection rate in the result. Kavitha et al. [13] describes two metrics that bettered the effectiveness of testing by following the TCs which detect maximum faults.

History based approach have been proposed by Kim and Porter [14] where historical information has been used to reduce cost and increase the efficacy of the testing process. The main drawback of this method is the execution of the test cases in binary format. Evaluation in the constraint environment is done differently than the non-constrained one. This work has been modified by Fazlalizadeh et al. [15] for quicker fault detection in constraint environment by proposing new equations considering effectiveness of test case, the history of execution, and the assigned least priority to them. The result was matched with random ordering and for visualizing the empirical result, box-plots were used for showing stability and quicker fault detection.

A newly introduced metric has been used by Muthusamy and Seetharaman in their work in [16], that is based upon rational priority factors like varying priorities in requirement, priorities in selecting test cases , the run time required and fault severities. The result affirms that the priority rate of the test case per unit time can be enhanced and elevates the quality of testing and customer satisfaction. Jeffrey and Gupta [17] presented a new approach taking into consideration the coverage requirements of the output of test cases that is present in the relevant slices. Three different slicing approaches based on heuristics are considered to prioritize test case and the result is compared with other traditional approaches for prioritization. It is shown that the effectiveness of the test cases for quicker fault detection is improved.

Many meta-heuristic techniques have been used to deal with the TCP problem. Huang et al. [18] proposed an historical record based cost-cognizant TCP techniques where the record from the past regression testing history is ordered using genetic algorithm.

The result shows the effectiveness of test cases in enhanced if search algorithms are used. Arafeen and Hyunsook [19] proposed a technique that incorporates the requirements-based congregation approach and information from conventional code analysis for prioritization. They did factual study on Java programs and shown that the use of requirement information can be effective during prioritization of test cases. Li et al. [20] have examined empirically the utilization of few meta-heuristic, greedy and search algorithms based on evolutions for ordering of test cases. The outcome affirms the regression test search space types. In this type, the genetic algorithm performance is better than others. Mala et al. [21] contemplated on the resemblances between artificial bee colony (ABC) optimization with ant colony optimization (ACO). The result shows that the prioritization done based on ABC have many merits than based on ACO. In a time controlled conditions, Singh et al.[22] prioritized test cases using ACO algorithm. The suggested method efficiency is compared with other methods using APFD metrics. The result affirms that the proposed technique ordering equals optimal ordering but superior than the rest. Mirarab et al. [23] recommended a technique based on changes made, fault and coverage-based where they took Bayesian networks for prioritization. They compared the result with ten other techniques and the result shows some promising aspects of Bayesian network as the fault grows.

III. THE TEST CASE PRIORITIZATION PROBLEM

Prioritization of RTS delivers an opportunity to the testers and project authorities to choose something after thinking carefully on several possibilities of which and how many tests should be executed under constraint environment. All the test cases might not be eligible for execution, so one needs to find some testing criteria to decide when to stop testing due to several decision factors like time constraint, test criticality or customer requirements, or to increase the likeliness that the testing duration have been used more effectively if regression testing had unexpectedly halted than if test cases were not prioritized. The simplest scheme for determining the priority category (P1-P4) for the test cases is given below:

P1: The test cases are important and need to be executed.
P2: If time is there, the test cases will be executed.
P3: The test case is not that much relevant prior to the software being delivered, but can be tested after the release.
P4: The impact of the test case is not that important, so they are not that much important.

In the prioritization scheme, the central theme is to ensure that the execution of the high priority test cases must be done prior to lower ones and the low priority test cases do not cause any severe failure to the software. So the consequences of not testing the lower priority test cases must be checked before the product release.
The following explains the TCP problem by definition [5]:

Given: A test suite, TS, all possible permutations of TS, PTS, and fn is the function from PTS □ R, R is the real numbers.

Problem Statement: To detect TS' Є PTS such that for all TS'', TS'' Є PTS and TS" ≠ TS' [f(TS') ≥ f (TS'')]

In the problem statement defined above, PTS depicts all possible combinations of orderings of TS and fn is the objective function. fn, when applied on to any arrangements, gives an weighted value for that particular ordering.

For the sake of simplicity, from the problem statement it can be presumed that the higher weighted values are desirable to lower ones. There are possibly many numbers of performance criteria are there for test case prioritization. The goal of this paper is to construct a TCP technique that prioritizes the test cases which can bring about the early identification of faults.

IV. PROPOSED WORK

This section deals with the motivation, the proposed methodology for prioritization and description of the APFD metrics to estimate the competence of test suites.

A. Motivation

Prior to this prioritization of test cases, long time that range from months to years, is required to execute a test suite completely depending on its size. As the complexity increases due to much functionality in the modern-day software, it is not feasible to execute every test case inside a time limit. The tester, by using the efficient prioritization technique, can reschedule the test cases so that early detection of faults is enhanced. The new RTS prioritization method is presented in this paper that orders the test cases with the ambition to maximize the rate of faults location in the constrained environment. It is based upon the metric - Effectiveness of Test Case in Detecting Faults (ETDF). ETDF can be defined as the product of the number of faults found by the test case "N_i" to the maximum number of faults available in the system "TN_i" and the total time taken to reveal the faults "TT_i". Mathematically, it can be represented as:

$$\text{ETDF} = \frac{\text{Number of faults covered by a test case}}{\textit{Total number of faults available in the system}} * \text{Total time taken to detect the faults}$$

$$= \frac{N_i}{TN_i} * TT_i \qquad (1)$$

The test cases will be sequenced according to the value of ETDF. The higher the value of ETDF, the more priority must be assigned to the test case. It must be assured that the more priority test cases must be executed earlier than the lower ones because the more priority test cases can identify more number of faults.

B. Proposed Algorithm for Prioritization

The technique for prioritization is represented in the form of an algorithm given below:

Algorithm: Test Case Prioritization Algorithm

Input: Test suite, TS and Fault Matrix showing the test cases and the subsequent faults covered
Output: Ordered test suite, TS'

Begin

1. Assign TS' to be empty
2. for every test case t_i Є TS do
3. Compute ETDF for each test case using equation (1) as given in section 4.1
4. end for
5. Sort TS in descending order of ETDF values for every test case
6. If (ETDF values are same)
7. Then preference will be given to the test case that has appeared in the fault matrix earlier
8. Set TS' be T

End

The algorithm is established on the information of total count of defects/faults covered by every test case. This is supplied by identifying code coverage of every test case prior to execution of the proposed algorithm. The information provided to the algorithm determines the maximum faults located by every test case and then use the value of ETDF to sort the test cases in descending order.

C. Evaluation of Test Suite Effectiveness

It is necessary to evaluate the effectiveness of the arrangement of the test case using new prioritization technique used in this manuscript. The performance of the prioritization technique should be calculated by the rate of faults located. To measure its effectiveness, the performance can be calculated by Average percentage of faults detected (APFD) metrics. Elbaum et al. [11][3] developed APFD metric to calculate the weighted average of the percentage of faults found during the running of a test suite. Its value lies between 0 to 100. The values that approach towards 100 mean a quicker fault detection rate. Thus, it is a metric that detect faster identification of faults by a test suite.

APFD can be computed using following notation:

$$APFD = 1- ((TF1 + TF2 + TF3 +\ldots..+ TF_m) /nm) + 1/2n \quad (2)$$

Where, TF_i is the exact location of the first test in TS that exposes fault i

m represents total number of faults contained in SUT that is exposed under TS

n represents total count of test cases in TS

V. Experimental Evaluation and Discussion

The vivid description of the experimental set-up and the test suite needed for our study has been extracted from the existing literature [13]. In this, they have taken a Visual Basic project consisting of 4500 LOC tested in the organization CCSQ, Chennai (India). Programs are rigorously tested by manual testing and with the help of QTP 9.5 tool. The given prioritization algorithm was tested by seeding faults. The faulty program is executed and the total count of test cases is executed to find if all the defects are computed. They wrote 10 test cases for detecting all faults in the program. The time required to locate all bugs by every test case is noted. The Table 1 depicts the number of faults caught by individual test case and the time seized to detect faults in ms. The fault matrix is as follows:

Table-I: Test Cases and Faults,
'√' The Corresponding Fault Detected by the Test Case

TCs / Faults	TN1	TN2	TN3	TN4	TN5	TN6	TN7	TN8	TN9	TN10
FN1								√	√	
FN2		√	√		√					
FN3				√		√				√
FN4		√	√							
FN5								√		
FN6								√	√	
FN7				√	√		√			
FN8	√					√				
FN9				√		√				√
FN10	√							√		
N_i	2	2	2	3	2	3	1	4	2	2
TT_i (ms)	9	8	14	9	12	14	11	10	10	13

In the above Table I, number of faults detected, Ni, is given for each test case, TN1...TN10 and the total number of defect/ fault(s), FN1...FN10. There are 10 test cases and 10 faults in total as shown. The ETDF value can be calculated using equation (1) given in section IV.A. Table II, represents the computed value of ETDF for every test case TN1…TN10.

Table-II: ETDF values for the test cases T1…T10

Test Cases	ETDF Values
TN1	1.8
TN2	1.6
TN3	2.8
TN4	2.7
TN5	2.4
TN6	4.2
TN7	1.1
TN8	4
TN9	2
TN10	2.6

Arranging the above mentioned test cases in accordance with the ETDF values for execution, the prioritized order obtained is as follows: TN6, TN8, TN3, TN4, TN10, TN5, TN9, TN1, TN2, and TN7. It should be noted that if the ETDF values are same for two test cases then the former test case in the fault matrix will be preferred.

A. Performance Analysis and Discussion

This sub-section deal with comparative study between different techniques and the performance of each technique has been discussed for choosing a better prioritization technique. For achieving fair comparison, APFD value has been used. APFD metric is used to quantify the rate of defect/fault identification. For every technique, APFD value is obtained. These values are finally compared to obtain the robustness of our proposed prioritization algorithm than the others.

- Comparison of Proposed Order with Other Prioritization Techniques:

Table III, denotes the suggested prioritization algorithm that is matched with several other prioritization algorithms like no order prioritization (NOP), reverse order prioritization (REP), Random order prioritization (RP) and previous work [13]. In NOP, the test cases are ordered in same manner as it is. In REP, the test cases are organized in reverse order as they are produced. And in RP, it is arranged in any manner. One of the instances of RP is taken for our study.

Table-III: Ordering of Test Cases for Various Prioritization Techniques

NOP	REP	RP	PREVIOUS WORK [13]	PROPOSED ORDER
TN1	TN10	TN5	TN8	TN6
TN2	TN9	TN9	TN4	TN8
TN3	TN8	TN7	TN9	TN3
TN4	TN7	TN8	TN6	TN4
TN5	TN6	TN4	TN2	TN10
TN6	TN5	TN1	TN1	TN5
TN7	TN4	TN3	TN5	TN9
TN8	TN3	TN6	TN10	TN1
TN9	TN2	TN10	TN3	TN2
TN10	TN1	TN2	TN7	TN7

- Analysis of APFD*:*

Plotting %age of defect/fault(s) detected on the y-axis and %age of test suite executed on the x-axis. The coloured area under the curve is the APFD value. The APFD percentage for NOP, REP, RP, previous work [13] and the proposed order are given respectively in the Fig. (1-5).

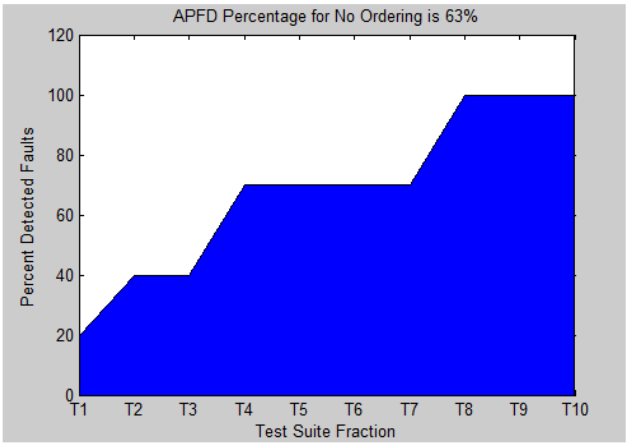

Fig. 1: APFD percentage for NOP

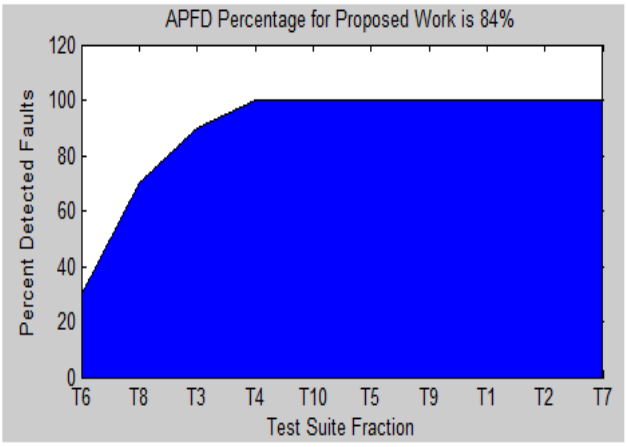

Fig. 2: APFD percentage for REP

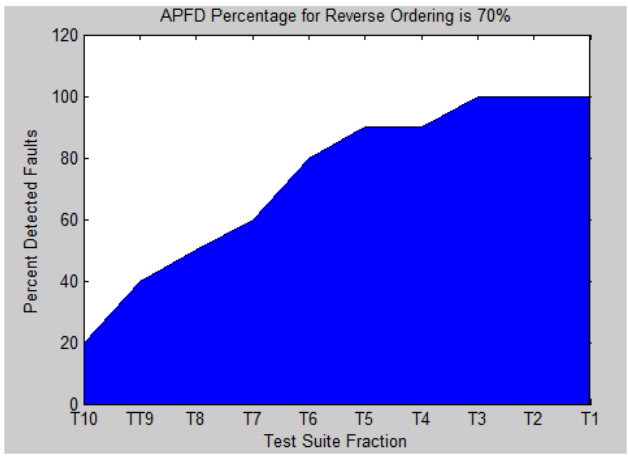

Fig. 3: APFD percentage for RP

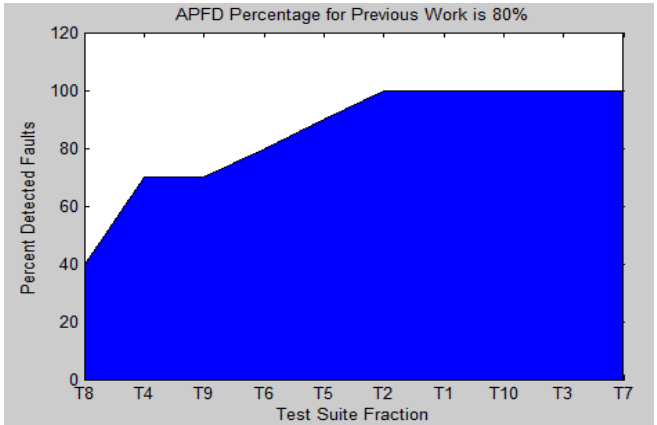

Fig. 4: APFD percentage for Previous Work [13]

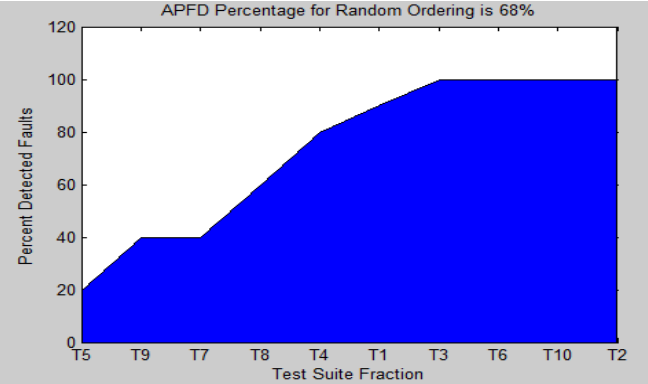

Fig. 5: APFD percentage for Proposed Order

The APFD percentage as computed from the area graph for various prioritization approaches like no ordering, reverse ordering, random ordering, previous work and proposed ordering are represented in Table IV. In the case of the proposed technique, the APFD value is more than all other prioritization techniques. It is also observed that our proposed new algorithm exceeded all other approaches, hence better and more efficient.

Table-IV: Various prioritization techniques and their APFD percentage

Different Prioritization Techniques	Percentage of APFD
No Ordering	63
Reverse Ordering	70
Random Ordering	68
Previous Work [13]	80
Proposed Ordering	**84**

- **Calculation of Total Fault-Coverage:**

This sub-section focuses on detecting all faults by executing least possible number of test cases. The comparative study has been done between the no prioritized and different prioritized test cases as shown in Fig. 6. The new technique needs less than half of the test cases, i.e., 40% of test cases to identify all defects/ faults present in the software, whereas the other techniques such as no prioritization and reverse order prioritization techniques considers 80% of total test case and the previous work algorithm takes 60% of the total test cases in the test suite to find out all faults present in the software under test (SUT).

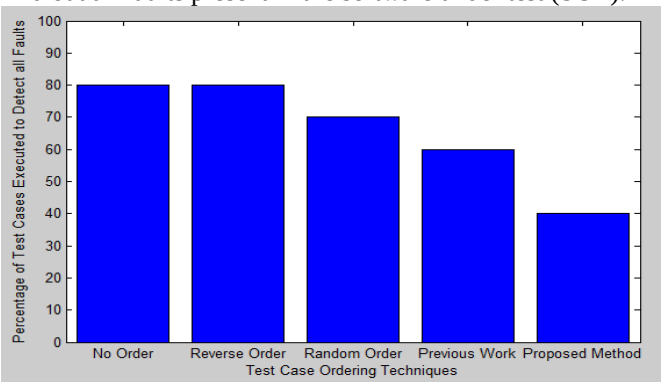

Fig. 6: Test case percentage required to detect all faults by different prioritization techniques.

VI. LIMITATIONS OF OUR WORK

In this part, we emphasize on some of the limitations that can influence the validity of our proposed work and its performance analysis. These limitations are as follows:

- Although APFD metric is widely used for quantifying rate of fault detection, but it has certain limitations like it does not consider cost factor between faults and test cases etc. It only focuses on few dimensions of the effectiveness of ordering. We need to contemplate on other measuring criteria for addressing effectiveness.
- The severity of faults is not considered in our studies. Practically, every fault has got some severity value that might impact the test case effectiveness.
- Test suite organization will probably vary under different procedures and processes, the result might not generalize. This algorithm must be implemented on real-time software before claiming its robustness.

VII. APPLICATIONS OF OUR PROPOSED ALGORITHM

The proposed algorithm can be used in complex test suite prioritization problems. It will save huge cost and effort during software development life cycle as compared to smaller projects. The software tester, using this proposed approach, can easily perform test case prioritization with minimum execution duration and greater fault detection percentage.

VIII. CONCLUSION AND FUTURE SCOPE

This paper discussed about a new prioritization algorithm for test case prioritization for improving regression testing process. It presented the factual study results that scrutinized their efficiency. The thorough analysis of the result is performed for both prioritized and non-prioritized sequences of test cases using the metric APFD. It indicates that the non-prioritized ordering algorithm performs much worse than various other prioritization techniques like reverse ordering, random ordering, existing work ordering and the proposed prioritization technique ordering. And among all prioritization techniques, our proposed approach outperforms all as depicted in the APFD graphs. The criteria that we have chosen for our study is based on fault detection, which is different from code coverage. Also the total number of test cases required is less as matched with

ther prioritized and non-prioritized approaches, when executed to find the entire faults in the software. The effectiveness of our recommended prioritization technique will be more if the severity values that are incorporated to each failure are taken into account. In future work, we will deal with the severity values linked with each test case and analyze the result. Also, other metrics like ANOVA and ANCOVA, instead of APFD, will be applied to analyze the test case prioritization results for achieving more accurate efficiency.

REFERENCES

1. Myers GJ. The Art of Software Testing. John Wiley & Sons, 1979.
2. I.S.G. of Software Engg. Terminology, IEEE Standards Collection 1990, IEEE Std 610.12-1990.
3. Rothermel G, Untch RH, Chu C, Harrold MJ. Test case prioritization: an empirical study. Proc. Int'l Conf. Software Maintenance 1999; 179-188.
4. Wong WE, Horgan JR, London S, Agarwal H. A study of effective regression testing in practice. Proc. Eighth Int'l Symp. Software Reliability Eng. 1997; 230-238.
5. Malishevsky AG, Ruthruff JR, Rothermel G, Elbaum S. Cost-cognizant test case prioritization. Technical report 2006: Department of Computer Science and Engineering, University of Nebraska-Lincoln.
6. Elbaum S, Malishevsky AG, Rothermel G. Test case prioritization: A family of empirical studies. IEEE Trans. Software Engg. 2002; 28 (2): 159-182.
7. Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans. Software Eng. 2001; 27(10): 929-948.
8. Korel B, Tahat L, Harman M. Test prioritization using system models. In the Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM' 08) 2008; 247-256.
9. Korel B, Koutsogiannakis G, Talat LH. Model based test suite prioritization Heuristic Methods and Their Evaluation. Proceedings of 3rd workshop on Advances in model based testing (A – MOST) 2007:London (UK); 34-43.
10. Fraser G, Wotawa F. Test-case prioritization with model-checkers. In SE'07: Proceedings of the 25th conference on IASTED International Multi-Conference 2007: Anaheim, CA, USA (ACTA Press); 267-272.
11. Elbaum S, Malishevsky A, Rothermel G. Prioritizing test cases for regression testing. Proceedings of the International Symposium on Software Testing and Analysis 2000; 102-112.
12. Krishnamoorthi R, Mary SA. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Information and Software Technology 2009; 51(4): 799-808.
13. Kavitha R, Sureshkumar N. Test case prioritization for regression testing based on severity of fault. Int'l J. Computer Sc. and Eng. (IJCSE) 2010; 02(5): 1462-1466.
14. Kim JM, Porter A. A history-based test prioritization technique for regression testing in resource constrained environment. Proceedings of the 24th International Conference Software Engineering 2002; 119-129.
15. Fazlalizadeh Y, Khalilian A, Azgomi HA, Parsa S. Incorporating historical test case performance data and resource constraints into test case prioritization. Lecture notes in Computer Science 2009: Springer; 5668: 43-57.
16. Muthusamy T, Seetharaman K. Efficiency of test case prioritization technique based on practical priority factor. Int'l J. Soft Computing 2015; 10(2): 183-188.
17. Jeffrey D, Gupta N. Experiments with test case prioritization using relevant slices. Journal of Systems and Software 2008: Elsevier; 81(2): 196-221.
18. Huang Y, Peng K, Huang C. A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software 2012: Elsevier; 85(3): 626-637.
19. Arafeen MJ, Hyunsook D. Test case prioritization using requirements-based clustering. IEEE 6th Intl. Conf on Software Testing, Verification and Validation (ICST) 2013: Luxembourg; 312-321.
20. Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Trans. Software Eng. 2007; 33(4): 225-237.
21. Mala DJ, Kamalapriya M, Shobana R, Mohan V. A non-pheromone based intelligent swarm optimization technique in software test suite optimization. IEEE Conf. Intelligent Agent and Multi-Agent Systems 2009; 1-5.
22. Singh Y, Kaur A, Suri B. Test case prioritization using ant colony optimization. ACM SIGSOFT Software Eng. Notes 2010, 35(4): 1-7.

AUTHORS PROFILE

SOUMEN NAYAK received the B.Tech. degree in Computer Science and Engineering from Biju Patnaik University of Technology, Odisha, India, in 2008, and the M.Tech. degree from the Department of Computer Science, Central University of South Bihar, Patna, India, 2014. He is currently a Research Fellow with the Department of Computer Science and Engineering, Indian Institute of Technology (Indian School of Mines), Dhanbad, India. His current research interests include software engineering, software testing and nature – inspired computing.

.

CHIRANJEEV KUMAR received the M.E. degree from the Department of Computer Science and Engineering, Motilal Nehru National Institute of Technology, Allahabad, India, in 2001, and the Ph.D. degree in computer science and engineering from Allahabad University, India, in 2006. He is currently a Professor with the Department of Computer Science and Engineering, Indian Institute of Technology (Indian School of Mines), Dhanbad, India. In his about 18 years of teaching and research career, he has contributed several research papers in leading refereed journals and conference proceedings of the national and international reputes. His main research interests include mobility management in wireless networks, ad hoc networks, and software engineering. He has been a fellow of the Inventive Research Organization since 2016. In 1998, he was felicitated with a Certified Novel Administrator and a Certified Novel Engineer.

SACHIN TRIPATHI received the B.Tech degree from Chhatrapati Shahu Ji Maharaj University, Kanpur, India. He received the M.Tech. and the Ph.D. degree in Computer Science and Engineering from Indian Institute of Technology (ISM), Dhanbad, India. He is currently working as Associate Professor with the Indian Institute of Technology (ISM), Dhanbad, India. He has teaching computer science subjects for over more than eleven years. He has contributed about 55 research papers. He has authored a book titled "Enhancements on Internet Applications: Multicast, Secure E-Mail Messaging and E-Business". He is an active reviewer for some international journals. His research interests mainly focus on Group Security, Ad-hoc and Sensor Network, Artificial Intelligence, and Software Engineering.

.

LAMBODAR JENA received the M.Tech degree from the Department of Computer Science, Indian School of Mines, Dhanbad, India in 2000, and Ph.D. from the Department of Computer Science, Utkal University, Bhubaneswar,India in 2018. He is currently working as an assistant professor in the Department of Computer Science & Engineering, SOA University, Bhubaneswar. In his about 19 years of teaching and research career, he has contributed several research papers in leading refereed journals and conference proceedings of the national and international reputes. His main research interest includes Privacy Preservation, Data Mining and Machine Learning.

.