ISSN:0975-9646

Comparisons of Test t Case Prioritization Algorithm with Random Prioritization

Thillaikarasi Muthusamy[1], Dr. Seetharaman.K[2]

[1](Assistant Professor) Department of computer science and Engineering
[2](Associate Professor) Department of computer science and Engineering,
Faculty of Engineering and technology
Annamalai University, Annamalai Nagar, Tamilnadu, India-608002

Abstract—Test case prioritization organizes test cases in a way to accomplish some performance goals efficiently. Rate of error detection is one most important performance objectives. The test cases must be given in an order that calls down the opportunity of fault detection in the early life cycle of testing. Test case prioritization techniques have shown to be good in improving regression testing activities. In this report, we have suggested an algorithm which prioritizes the system test cases based on the six factors: customer priority, changes in requirement, implementation complexity, requirement traceability, execution time and fault impact of requirement. We conducted a controlled experiment on two industrial data sets to compare the suggested value based test case prioritization algorithm with random prioritization for early rate of fault detection. The average share of fault detection metrics have been employed to evaluate the efficiency of proposed and random priority and it proves that the proposed value based algorithm is more effective than random prioritization to generate sequence of trial examples for early rate of fracture detection.

Index Terms—Test case prioritization (TCP), test case factors, Average percentage of error detection

I. INTRODUCTION

Software testing requires resources and consumes 30- 50% of the total cost of development. Testing is often executed in time to market pressure and is supposed to test whole software in a systematic fashion to achieve quality as a good deal as possible. Testing also includes many other prospects such as delivering error free versions and checking, thorough software iteration in available time and other resources. In this exercise it may not be possible for testers to offer quality product free of bugs to customers so it ultimately promotes the possibility of possible risks in software, while on the other hand time slippage occurs for delivering the satisfactory quality assessment of software [1]. Testing has been traditionally performed in value neutral approach in which all software components are given same testing resources to prove but this eventually does not satisfy the goal customer as approximate 36% of software uses are only often used [2]. Consequently it is meaningless to test the whole software in this way. One character of testing is a regression testing in which software is tried out after making some modifications to it. Regression testing is thought to be really expensive due to repeated execution

It records detailed execution traces of existing test cases. Regression testing involves execution of a great number of test cases and is time consuming [3]. It is impractical to repeatedly test the software by executing a complete lot of test cases under resource constraints. Thus, it is desirable to select partial test cases to try the software for fault identification at an early point. Thither are many selection techniques for this such as test all our random selection techniques. Only the choice of test cases, according to some pre-determined standards for early rate of fault detection might be somehow risky technique as there could be many other unselected test cases, which can result in more fat

Identification. When very high quality software is desired, it might not be wise to discard test cases as in test minimization. Whereas, we can prioritize the test cases by following some criteria such as code coverage, execution time or early fault detection to execute the test cases in that parliamentary law to achieve the particular destination. We can claim benefit of test case prioritization if we conflate it with the value based access. The concept of value has been presented by Barry Boehm. In the suggested attack, more value is awarded to software functions; those are more critical or important to our stakeholders. As the target of the software or any business is to increase the yield on investment (ROI) so by introducing value in testing, testers can focus on more concerned modules of software to satisfy stakeholders. These modules are named with the assistance of stakeholders. Stakeholders give some value to modules and according to some predefined criteria; these modules, assign some value and tried accordingly. This concept is known as value based test case prioritization [2].

Thither are many algorithms used for test case prioritization such as greedy, additional greedy, hill climbing etc. Artificial intelligence is very vast area and many of its algorithms are employed in software testing such as genetic algorithm or particle swarm optimization. There is demand to optimize the testing resources in a manner to provide quality software. In this paper, we present a test case prioritization algorithm in which test cases are prioritized while considering the worth of value in the testing process. Our goal of value based test prioritization is for early error detection. An experiment has been done to compare effectiveness of the proposed and existing technique. The results mean that the proposed AI value based test

prioritization may be a good resolution for a prioritization problem than random technique. The remainder of the paper is formed as follows: Section 2 provides briefly the related work. Part 3 introduces the problem analysis of test case prioritization. Part 4 is about proposed methodology and experimentation. In Section 5 conclusions and future research are discussed.

II. RELATED WORK

There has been using different criterion for test case prioritization in literature such as code coverage [4, 5, 6, 7] or non-code coverage techniques [8, 9] etc. Li et al. [4] Has performed an experiment to compare three greedy algorithms and two meta-heuristics algorithms for test case prioritization under code coverage criteria on six programs. Results indicated that it is the size of the search space that involves the complexity of test case prioritization; not the size of the trial suite itself. Global search techniques and additional greedy algorithm performed better than local search techniques and greedy algorithms respectively. In another paper seven fault versions of C programs are utilized in an experiment to compare different statement level techniques by manually seeding faults in programs. Solutions indicated that optimal prioritization greatly improved rate of error detection [15]. Rothermel et al. Has also used code coverage measures for an experiment on seven fault versions of programs written in C language for early fault detection The results specify that test case prioritization can considerably improve the rate of fault detection of test suites [5]. Artificial intelligence techniques are becoming popular to function for test prioritization. Walcott et al. has used a genetic algorithm for test case reordering for early fault detection criteria by using coverage information [6]. A controlled study experiment along with two case studies has been transmitted to assess effectiveness of parameterized genetic algorithms. Outcomes were compared with other heuristics, including initial, reverse of initial test suit ordering, random and fault aware prioritization. It was needed for each test case to be autonomous from other test cases to maximize the fault detection ability. Results reveal the genetic algorithm as more promising than any other technique under time constraint environment. Fayoumi et al. [7] has used ant colony optimization (ACO) and rough set theory concepts to obtain a best quality test case of unit test for object oriented source code. This approach used method call, giving arguments and command flow dependency graphs. A hybrid novel framework was proposed by inspiring the natural ant. Circulation and exploring the best test case value had been done through Ant colony pheromone matrix Rough set is used as a stopping criteria rule in the proposed model. Bayesian network (BN) approach has also been applied to prioritize the test cases [8]. An empirical study on five Java objects indicates the strength of feedback mechanism of BN approach in terms of early fault detection. Particle swarm optimization (PSO) is an optimization technique of swarm intelligence paradigm. In [9] author has used test case coverage and used PSO to assess the best possible placement of test events in search swarm in modified software units. Existing test case priorities and fitness of test events were used as parameters for new priorities of test examples. PSO was found to be more efficient in term of time and cost than greedy algorithm. Test case prioritization has also been applied to dilute the quality assurance cost as well as for minimizing the fault detection effort. The problem with reducing fault detection effort was that it may cause the information loss, as a resolution of which debugging cost gets increase. And then it was a great challenge to reduce the quality assurance cost which includes both the testing and debugging cost while minimizing the loss of diagnostic fault information. The Author has proposed the on-line greedy diagnostic prioritization approach that uses the observed test result to determine the following test case. In this approach high utility tests were those tests which maximize the reduction of diagnostic cost at each measure on average [10].

III. PROBLEM DEFINITION

Faster detection of faults and early satisfaction of stakeholders are two objectives which can be fulfilled by a value based testing. Traditionally random prioritization is being employed to satisfy these objectives, but it is not giving the desired results all the times. Hence there is a need of better technique for overcoming this issue. For this role we are proposing value based TCP algorithm to reorder the test cases to bring out more optimal solutions. Our proposed TCP algorithm considers the stakeholder's value against requirement and test cases and generate ordering that detects more faults at earlier levels.

IV. PROPOSED WORK

We have used value based approach at two levels 1) on requirement level 2) and on testing level. On first level requirement priority is being applied to incorporate test case value. This requirement priority is provided by the concerned stakeholders. Stakeholders did ranking of these requirements from 1- 10 by categorizing those as catastrophic, medium complexity and less significant. Requirements having more value were more important to stakeholders. On the second level development team was requested to mark the test cases by following the respected requirements. These test cases were graded according to some pre-determined components. These two factors are discussed below. Values of these two grades were computed to get the internet value. Test cases were prioritized according to produced results for earlier error detection. To define the constituents required in our proposed prioritization algorithm, we believe the following components from the literature in our proposed algorithm: (1) customer priority (2) implementation complexity (3) requirement volatility (4) requirements traceability (5) execution time and (6) fault impact of requirement. We talk over these factors here at a lower place. Roughly 36% of the software functions is only constantly used, while 19% only used often and while the rest percentage is not applied at all i.e. 45 % used [2]. Frequent failures are induced by the flaw that is situated along the course of regular execution, and greater effort should be constructed to detect such kind of faults [11, 12]. The perceived customer value and satisfaction can be increased, by giving priority to

client demands for development [13,2]. To addressing this problem, clients were required to place the requirement from 1 to 10 by considering the importance of the necessity. Highest customer priority is denoted by 10. Implementation complexity refers to individual measures of amount of difficulty perceived by the developers of the requirement by the development team. Amland carried out an investigation to ascertain that the functions with greater McCabe complexity are those with a high number of faults [14]. From the total system 20 % modules of the arrangement resulted in 80% of the faults [15, 16, 17]. Roughly 50% of the total faults discovered in a project comprise of those erroneous beliefs that are brought out in the requirement phase [24]. Rigorous defects that deliver to the customer costs hundred times greater averagely to resolve as compared to resolving the same problem in the requirements time [18]. Requirements volatility is measured as the number of times the development cycle of a requirement has been altered with respect to when the requirement was first introduced. It is essentially a judgment of the requirements change with respect to its commencement date. It also ranges from 1 to 10[19, 17]. The relationship between various artifacts in a software development process such as requirements, design and test cases is known as traceability [23]. Literature indicates that Software quality can be ameliorated by bringing into account the requirements traceability [21]. In literature many authors have viewed the implementation time of test case as cost of test case [19, 16, 22, 9]. Test case costs should bear a heavy impact on the test case prioritization. In conditions of test case cost, it can be connected to the resources, such as execution time of the test case, hardware costs or even engineers' salaries. In our case test case cost is the performance time of test example. Fault proneness (FP) of essentials is the identification of the requirements that bear the most failures in the former variant of the development team [16]. As proved from literature the test efficiency can be improved by focusing on features that hold the greatest number of faults [20] [19]. A. Experimental Setup We have performed a controlled experiment on two industrial projects to measure the effectiveness of our proposed prioritization algorithm. We have been providing the documentation of these tasks. We picked use cases and test cases and asked the customers to rate these use cases. A value based requirement prioritization tool was employed to rank these requirements [13]. The special creature was designed to place the requirement at stakeholder and expert level. Project managers did the job of the expert role in these tasks. Microsoft excel 2009 was used as requirement-test case traceability tool. MATLAB 9.0 was used to implement this algorithm. We have used random number generation to produce 20 different orders of test cases. Test events are performed in these societies. No of test cases performed to determine the faults are worked out. For each project mean value of results are figured. The results of fault detection in both the cases are compared to strengthen the potency of the proposed test case prioritization. The following table describes details of the selected tasks.

TABLE I
PROJECTS DESCRIPTION

Attribute Description	Project 1	Project 2
Nature of Project	Web based	Web based
No. Of modules	10	10
No. Of Test Cases	20	20
Complexity level	Medium	Medium
Team size	9	6

There were five stakeholders involved in this operation. Their part in the process is specified below:
The customer was responsible to provide system requirements, requirement's priority and field failure. The developers were asked to place the requirement according to its development complexity level. Demand Analyst /Business analyst records the requirements, their priorities and any changes to the essentials. Maintenance Engineer resolves the field failures defects and links the failure back to the requirements impacted. The tester provides test cases for each requirement, map the requirement to its test case, and executes the test examples.

B. Proposed Algorithm
Following is the proposed algorithm of value based PSO.
1. Get test case factor values for all test cases.
2. Compare values factor wise.
3. Give highest score to maximum and lowest score to minimum value.
4. Assign scores to remaining values by counting the number of terms to which the particular value is greater.
5. Repeat 2 to 4 for all test cases against every factor value.
6. Add these factor values for all test cases.
7. If same value is obtained for more than one test case then decision is taken by comparing
Requirement value of these test events.
8. If the use case rating also becomes equal or more than one test instance of same value belong to
The same use case, and so the test case will be carried out on a first come first serve basis.
In our study we have predicted the effectiveness of our proposed algorithm for early defect detection by using APFD metric. APFD metric is first developed by Elbaum et al. [4, 5]. This metric has been hired to measure the average rate of fault detection per percentage of test suite implementation. The APFD is calculated by taking the weighted average of the
The number of faults detected during the run of the test runs. APFD can be calculated using a note:
Let T be the test suite under evaluation, m be the number of faults contained in the program under test P and n is the total number of test cases while TFi denotes the location of the first test in T that exposes fault myself.
APFD= 1-TF1+TF2+….. + TFM /NM +1/2N.... (1)

C. Experimental Results
Our algorithm is based on analysis of the percentage of test cases performed to find the faults and on APFD metric's results. Abiding by the percentage of executing test cases in

earlier fault detection is important as sometimes regression testing ends without executing all test instances. Outcomes demonstrate that our algorithms can also achieve better execution in this event. For instance, in the first project if only 75% test cases could be melt down due to resource constraint, random strategy could find more or less 66% faults; while our proposed algorithm detects about 88% faults. In a second project if we consume 30% test cases to accomplish; then random strategy could find more or less 27% faults; while our proposed algorithm detects about 40% faults. This shows clear evidence that our proposed algorithm is a lot better in earlier fault detection than random technique. The graphical representation of these outcomes is presented at a lower place. We had also validated our results with the aid of standard APFD metric. We can discover the improved fault detection rate in earlier testing stage through our algorithm which is the proof of our algorithm; as more efficient and beneficial in earlier fault detection goal; whereas gradual improvement in APFD results are obtained by random strategy later in testing stage.

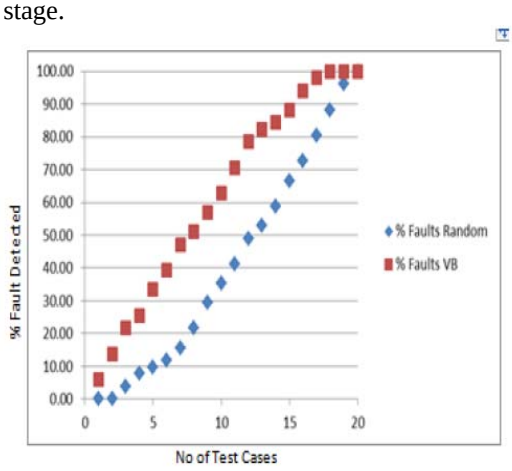

Fig. 1. Project 1 Results

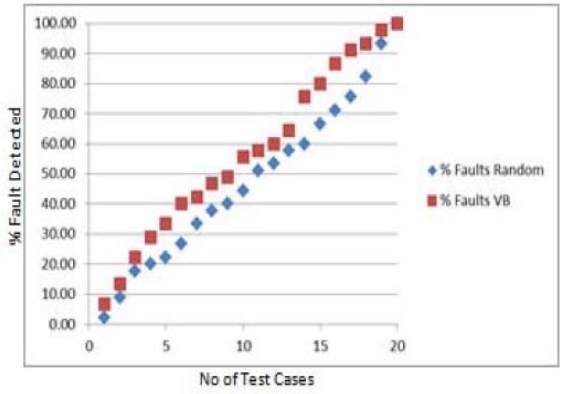

Fig. 2. Project 2 Results

TABLE II
PROJECT 1 RESULTS

Projects	Strategy	APFD results	Executed TC (%)
P1	Random	55%	66%
P1	TCP	78%	88%

TABLE III
PROJECT II RESULTS

Projects	Strategy	APFD results	Executed TC (%)
P2	Random	50%	27%
P2	TCP	67%	40%

In the first project our proposed algorithm has detected 78% faults while random ordering produces 55% of faults, in a second project our proposed algorithm has detected 67% faults while random ordering produces 50% of mistakes which again demonstrates the import of our findings.

V. CONCLUSION

The proposed algorithm is novel approach which introduced various test case prioritization factors in terms of value of stakeholders. The proposed algorithm works at two level 1) requirement level and 2) testing level. Six test case factors were being employed to grade the test cases, while use cases were prioritized by using the existing value based application. Value based algorithm was compared with random prioritization technique on two industrial projects and it demonstrated the effectiveness of value based algorithm for early rate of fracture detection. We are presently working to understand the essence of the proposed algorithm with evolved techniques. Additionally the proposed algorithm is examined on a limited information set. It can be a little creepy validated by taking large size projects having a huge pool of employment cases and trial fonts.

REFERENCES

[1] R. Ramler, S. Biffl and P. Grunbacher "Value-Based Management of Software Testing", Book Chapter
[2] B. Boehm and L. Huang, "Value-Based Software Engineering: A Case Study," IEEE Computer, vol. 36, pp. 33-41, Mar 2003
[3] L. Zhang, S. S. Hou, C. Guo, T. Xie and H. Mei "Time Aware Test-Case Prioritization using Integer Linear Programming", ISSTA'09 , Chicago, Illinois, USA, Jul 2009
[4] Z. Li, M. Harman and R. M. Hierons "Search Algorithms for Regression Test Case Prioritization",IEEE Trans. on Software Engineering, vol. 33, no. 4, Apr 2007
[5] G. Rothermel, R. Untch, C. Chu and M. Harrold, "Test Case Prioritization: An Empirical Study" Int. Conf. on SoftwareMaintenance, Oxford, UK, pp. 179 - 188, Sep 1999
[6] K. R. Walcott and M. L. Soffa "Time Aware Test Suite Prioritization", ISSTA'06, Portland, Maine, USA, Jul 2006
[7] M. Fayoumi, P. Mahanti and S. Banerjee: OptiTest: "Optimizing Test Case Using Hybrid Intelligence" World Congress on Engineering 2007
[8] S. Mirarab and L.Tahvildari "An Empirical Study on Bayesian Network-based Approach for Test Case Prioritization" Int. Conf. on Software Testing, Verification, and Validation 2008
[9] K. H. S Hla, Y. Choi and J. S. Park "Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting", 8th IEEE Int. Conf. on Computer and on Information Technology Workshops 2008
[10] A. G. Sanchez "Prioritizing Tests for Software Fault Localization" 10th Int. Conf. on Quality Software, 2010
[11] J. C. Munson and S. Elbaum, "Software reliability as a function of user execution patterns and practice," 32nd Annual Hawaii Int. Conf. of System Sciences, Maui, HI, pp. 255-285, 1999
[12] J. Musa, "Software Reliability Engineering". New York, NY: McGraw-Hill, 1999

[13] B. Boehm, "Value-Based Software Engineering," ACM Software Engineering Notes, vol. 28, pp. 1-12, Mar 2003.
[14] S. Amland, "Risk Based Testing and Metrics," 5th Int. Conf. EuroSTAR '99, Barcelona, Spain, pp. 1-20, 1999.
[15] R. Krishnamoorthi, S.A. Sahaaya and Arul Mary "Incorporating varying Requirement Priorities and Costs in Test Case Prioritization for New and Regression testing", 2008
[16] X. Zhang, C.Nie, B. Xu and B.Qu "Test Case Prioritization based on Varying Testing Requirement Priorities and Test Case Costs", 2007
[17] H. Srikanth, L. Williams and J. Osborne "System Test Case Prioritization of New and Regression Test Cases", 2005
[18] F. Shull, V. Basili, B. Boehm, W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero and M. Zelkowitz, "What We Learned about Fighting Defects," IEEE Symposium on Software Metrics, Ottawa, Canada, pp. 249-258, Jun 2002.
[19] R. Krishnamoorthi, S.A. Sahaaya and Arul Mary "Incorporating varying Requirement Priorities and Costs in Test Case Prioritization for New and Regression testing", 2008
[20] T. Ostrand, E. Weyuker and R. Bell, "Where the Bugs Are," Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Boston, MA, pp. 86-96, Jul 2004
[21] A. Ahmed, "Software Testing as a Service" Auerbach Publications, New York: 2009
[22] A. M. Smith and G. M. Kapfhammer "An Empirical Study of Incorporating Cost into Test Suite Reduction and Prioritization", 2009
[23] R. Krishnamoorthi and S.A. Mary "Factor oriented requirement coverage based system test case prioritization of new and regression test cases",2009
[24] Standish.Group,"CHAOS." http://www .standishgroup .com /chaos. htm. Proceedings of the World Congress on Engineering and Computer Science 2012 Vol I WCECS 2012, October 24-26, 2012, San Francisco,USA.