International Journal of Soft Computing 10 (2): 183-188, 2015
ISSN: 1816-9503

Efficiency of Test Case Prioritization Technique Based on Practical Priority Factors

Thillaikarasi Muthusamy and K. Seetharaman
Department of Computer Science and Engineering, Faculty of Engineering and Technology,
Annamalai University, 608002 Annamalai Nagar, Tamilnadu, India

Abstract: Test metrics provides the current level of development in testing and gives a projection to proceed along with testing activities by allowing us to set targets and future trends. The objective of test metrics is to determine the planned and actual quantities of effort, time, resources which requires to complete the development of the software project. Test cases increases the ability to meet some targets such as code coverage and rate of fault detection. Here, researchers present a new metrics based on varying requirement priorities, test case priorities, test case execution time and fault severities. The case study demonstrates that the rate of test case priority per unit time can be increased and improved on quality testing and customer satisfaction. To review the practicality of this technique, researchers apply it as a realistic example from the industrial projects. Here, researchers summarize about test process measurement and analyze the effectiveness of metrics by cost, time, quality of test process based on the result of proposed metrics.

Key words: Regression testing, test case, test case prioritization, fault severity, rate of fault detection

INTRODUCTION

In the software development life cycle, regression testing is a necessary component that expose software errors after changes done in the program. A set of test cases should be exhibited adequately to test the software. When the software is scrutinized, a new set of test cases are appended to the test suite to test the changed requirements. It is not practical and efficient to re-execute test for every program if any change occurs. This is liable to increase the size of test suite, cost and time constraints. The problem of regression test case selection is solved by prioritizing test cases. Test case prioritization techniques can organize the test cases in order to increase the effectiveness of testing and also avoids the test suite minimization problems.

Recently, the experimental studies of Jeffrey and Gupta shows that fault detection capability can be enhanced by specific retaining of test cases. Numerous techniques for prioritizing test cases have been proposed which is described in the next study of related research. This study investigates the use of an advanced approach for prioritizing the test cases in regression testing. The main objective is to run test cases based on practical weight factors which increases the possibility of fault detection and detects the severe faults in early stages. The rest of this study is structured as follows:

LITERATURE REVIEW

Test case prioritization (Elbaum *et al*., 2002) is most widely used technique in regression testing (Elbaum *et al*., 2002; Rothermel *et al*., 2001). In general test case prioritization sorts existing test cases for regression testing according to the performance. The metric of Average Percentage Of Fault Detected (APFD) is used for evaluating test case prioritization techniques. Researchers had found that various prioritization techniques is used to measure APFD values besides it produces statistically significant results. The APFD measures the average number of faults identified in a given test suite. The APFD values ranges from 0-100 and percentage of fault are detected by plotting the area under the curve towards the percentage of test case executed.

The prioritization techniques presented by Rothermel *et al*. (1998) involves coverage of test cases. Here, we rate the test cases based on statement/function/ branches which was covered by each test cases. Rothermel *et al*. (1998) used TCP techniques that contains 3 different groups such as comparator group, statement level group, functional level group. Each group includes fine granularity techniques such as random ordering, optimal ordering, total statement (total-st), total branch (total-br), total function (total-fn), additional statement (addtl-st), additional branch (addtl-br), additional function

Corresponding Author: Thillaikarasi Muthusamy, Department of Computer Science and Engineering, Faculty of Engineering and Technology, Annamalai University, 608002 Annamalai Nagar, Tamilnadu, India

(addtl-fn). Total statement (total-st) in the TCP techniques evaluate the coverage of statements in a program. Prioritization test cases is done for total statements which cover and sorted in the order of coverage achieved. The entire branch and function is similar to the total statement (total-st) which use the function and branch coverage rather statement coverage information. Additional statement (addtl-st) in the TCP techniques select a test case that covers maximum number of statements which are not included in each test case. When two test cases cover the same number of additional statements in a round, it randomly picks one, the additional function (addtl-fn) and additional branch (addtl-br) are similar to additional statement (addtl-st) which uses branch coverage and function coverage information, respectively.

The code coverage strategies (Elbaum *et al*., 2002) were measured using weighted Average Percentage of Fault Detected (APFD) (Rothermel *et al*., 2001), Average Percentage of Branch Covered (APBC), Average Percentage of Decision Covered (APDC) and Average Percentage of Statement Covered (APSC). APFD measure the rate of fault detected by a test suite. The main objective of the above technique is to increase the rate of fault detection at earlier stages of testing process (Rothermel *et al*., 2001).

The topic of test suite minimization (Park *et al*., 2008) is closely related to the test case prioritization. A test suite of test cases is a set of requirements which must satisfy the desired test coverage of the program and subsets of the test suite. Moreover, each subset is associated with redundant test cases to be removed so that a minimal test suite can be constructed. Hung *et al*. (2012) projected a cost awareness for prioritization technique according to various test costs and fault severities of each test cases. Moreover, it proposes Genetic algorithm to determine the effective prioritization order. It is validated using two UNIX programs and evaluate the effectiveness using APFD.

The topic of test suite minimization (Rothermel *et al*., 1998) is closely related to the test case prioritization. A test suite of test cases is a set of requirements which must satisfy the desired test coverage of the program and subsets of the test suite. Moreover, each subset is associated with redundant test cases to be removed so that a minimal test suite can be constructed.

Harrold *et al*. (1992) approaches the static program to detect affected program changes that are associated with definition-use. Harrold and Rothermel (1997) presented a model for changed code. This approach constructs control flow graph and modified version to select test cases that executes changed code. Agrawal *et al*. (1993) proposed dynamic segment based and the relevant segment based approaches to determine the test cases in a test suite which generate different outputs on new and old program.

Jeffrey and Gupta (2006) present a new approach to test case prioritization using relevant segments. Implementation of this technique is done by three different heuristics based on relevant segment and compares the effectiveness with traditional techniques only for requirement coverage.

Kim and Porter (2002) introduced a technique that uses historical execution of data for prioritizing the regression testing. Further, difficulty of test case prioritization is considered as a probabilistic approach so that selection of each test case in a test suite is based on the execution history that will be used in the next test session. Test cases are prioritized according to the execution history, faculty detection effectiveness and the coverage of the program, respectively.

Qu *et al*. (2007) suggested an algorithm based on test history and run-time information to prioritize test cases in run-time environment. This algorithm classifies test cases by the faults exposed and then prioritizes test cases according to the output from the related test cases. Park *et al*. (2008) proposed a historical value model which quantifies the historical value of test cases based on practical and historical test case costs.

Other approaches includes cost-benefits for regression testing (Malishevsky *et al*. (2006) and measures the impact of test case reduction on fault detection capability with the branch coverage techniques. These techniques permanently discard a subset of test cases that will need for regression testing while test case prioritization technique preserves all test cases and restructure them in test list.

PROPOSED PRIORITIZATION TECHNIQUE

In this study, researchers present the proposed set of practical prioritization factors and the Prioritization algorithm.

Factors to be considered for prioritization: The comprehensive set of weight factors is considered for test case prioritization so that the software test engineers will not neglect these practical weight factors while running a test case prioritization process. Researchers proposes 10 factors classified into 4 groups which are: time factors, defect factors, requirement factors and complexity factor.

Time factors: In this technique, researchers consider two time factors are as follows:

Execution Time (ET): It is the measure of total time required for the execution of test suite. Weightage can be assigned by three states such as high, medium, low ranges from 1-10 scale assignment. High denotes the scale from 8-10 points, medium denotes the scale from 4-7 points and low denotes the scale from 1-3 points. Douglas suggested that time consumption factor is a common factor for software testing and applied for test case prioritization.

Validation Time (VT): It is the measure of total time required for validating the expected result and actual result. Weightage can be ranged from 1-10 scale assignment. Kalyana described that time consumption for validation factor is the most important metrics to validate results and find a defect.

Defect factors: In test case prioritization, test cases detects bugs which have higher priority so that those bugs will be fixed and required to re-execute again. This factor is most widely used metrics in software testing as defect discovery rate.

Defect Occurrence (DO): It is the measure of detecting defects after the execution of test cases. Scale assignment is done by true and false conditions where true denotes 10 points and false denotes 0 points. Julie and Mark reported that this factor is widely used in defect measurement system and is recorded in defect reports.

Defect Impact (DI): It is a scale of measure for classifying seriousness of defects. When the system progress to several versions the development team can use the empirical data collected from previous version to identify the specific requirement that can be error prone. In this method, researchers estimated to calculate defect impact based on the severity of the fault identified in the previous run. When t number of faults identified by the ith test case then severity value of ith test case can be shown as:

$$S_i = \sum_{j=1}^{t} SV \quad (1)$$

Max(S) is the severity value of test case among all the test cases then the defect impact of ith test case can be calculated as:

$$DI_i = \left(\frac{S_i}{Max(S)} \right) \times 10 \quad (2)$$

The business was impacted for the end user by defect severity level. The high severity defect means low product/software quality and vice versa.

Requirement factors: In this technique, four requirement factors for the test cases as they impact much on new software. Each of the four factors are discussed as follows:

Customer assigned Priority (CP): It is a measure of importance to the customer requirement. So, the program with high customer importance must be executed earlier to improve customer satisfaction. The customer assigns the values for each requirement ranging from 1-10 where 10 denote highest customer priority.

Implementation Complexity (IC): It is the measure of complexity in implementing the requirement by development team. Requirements with high implementation complexity will have high number of faults. Each requirement is assigned to the values ranging from 1-10 where 10 indicate high implementation complexity.

Requirement Change (RC): It is based on the total number of times that a requirement has been changed in their development cycle. The 50% of faults in the projects is identified in requirement phases due to changes in requirement stages. The volatility changes for all the requirements are normalized to 10 point scale.

Requirement Coverage (ReqCov): It is the measure of total number of requirements covered by each test case in a test suite. Researchers can assign weight in 10 point scale where 1 is the minimum value and 10 is the maximum value. Literature review explains requirement coverage and helps to validate all requirements that are implemented in the system.

Complexity factor: The complexity factor determines the total effort required to execute the test cases. Several studies shows that the complexity of test cases is one of the most important factors for regression test case prioritization. Researchers can consider the following complexity factors.

Test case complexity (Tcplx): It measures the difficulty and complexity while running the test case during testing process. It also refers to the effort needed to execute the test cases. Weightage is assigned by true or false conditions where true denotes 10 points and false denotes 0 points for representing the complexity of test cases.

Test Impact (TI): It is based on the impact of test cases during software testing. This factor helps to identify the

importance of test cases which are not executed. Weightage can be given in three states such as high, medium, low where high denote 8-10 points, medium denotes 4-7 and low denote 1-3 points in scale assignment.

The proposed Prioritization algorithm: Values of all 10 factors are assigned for each test case during test design analysis and progress continually during software development process. Researchers can compute Weighted Prioritization Value (WPV) for each test case as:

$$WPV = \sum_{i=1}^{10} (PF\ value_i \times PF\ weight_i) \quad (3)$$

Where:

WPV = Weight prioritization for each test case calculated from 10 factors

$PF\ value_i$ = A value assigned to each test case

$PF\ weight_i$ = A weight assigned for each factor

The WPV is used to compute the Weighted Priority (WP) for its associated test cases. Let there be n total requirements for a product and test case j maps to i requirements. Weighted Priority (WP) is calculated as:

$$WP_j = \left(\frac{\sum_{x=1}^{i} PFV_x}{\sum_{y=1}^{n} PFV_y} \right) \quad (4)$$

By calculating these values we can prioritize the test cases based on WPV and WP for each and every test case in the test suite. Now, researchers introduce the proposed technique in an algorithmic form here under:

- Input: test suite T and factor values for each test case
- Output: prioritized test suite T'

Algorithm:

Step 1. Begin
Step 2. set 'T' empty
Step 3. for each t∈T do
Step 4. Calculate Weighted Prioritization Value (WPV) using Eq. 4
Step 5. Calculate Weighted Priority (WP) using Eq. 5
Step 6. end for
Step 7. Sort T in descending order based on the values of WP for each test case
Step 8. Let 'T' be T
Step 9. end

VALIDATION METRICS

Validation of the proposed Prioritization algorithm is carried out through three different validation metrics. First validation metric is based on the severity of fault detected for the product/software and the second metric is number of acceptable test cases based on size of prioritized test cases and third metric is based on total time taken for prioritization of test cases. Researchers present these validation metrics in the study:

Defect severity: Efficiency of testing process can be improved by focusing on the test case that contains high number of severe faults. So, each fault severity value is assigned by impact of fault on the product. Severity value is assigned in 10 point scale assignment as shown:

- Very high severe: SV of 32
- High severe: SV of 16
- Medium severe: SV of 8
- Less severe: SV of 4
- Least severe: SV of 2

Total Severity of Fault Detected (TSFD) is the summation of severity values of all faults known in a product. F number of faults identified for a product and TSFD can be computed as:

$$TSFD = \sum_{i=1}^{f} sv_i \quad (5)$$

Average Severity of Faults Detected (ASFD) can be computed with f defects as:

$$ASFD = \left(\frac{\sum_{j=1}^{f} sv_j}{TSFD} \right) \quad (6)$$

Acceptable test case size: Here, it is suggested to use the number of acceptable test cases in a validation metric because the size of prioritized test cases has an impact on the time, cost and effort during the execution of regression testing. This metric is the number of acceptable test cases expressed in percentage as:

$$Size\ (\%) = \left(\frac{Size\ of\ each\ test\ case}{Total\ test\ case\ size} \right) \times 10 \quad (7)$$

Where:

Size (%) = The size of acceptable test cases expressed in percentage

Size of each test case = The no of test cases generated by each method excluding low priority test cases

Total test case size = The total number of test cases used in the experiment

Total prioritization time: This is the total number of times the prioritization methods are running in the experiment. This is the time related metric used during pre-process and post-process of a test case prioritization:

$$TPT = T_{Assigning\ weight} + T_{TCP\ Run} \quad (8)$$

Where:

TPT = The total time consumed in running the Prioritization Methods

$T_{Assigning\ weight}$ = The total amount of time consumed in assigning weights and computing weight prioritization values

$T_{TCP\ Run}$ = The total time to run the Test Case Prioritization Methods including ordered test cases

EXPERIMENTATION AND ANALYSIS

The proposed test case Prioritization algorithm was implemented in the working platform of JAVA (Version JDK 1.6). Here, the application projects for regression testing was implemented. In this example a test suite has been developed which consisting of 10 test cases and it covers a total of 5 faults. The regression test suite T contains 10 test cases with default ordering {T1, T2, T3, T4, T5, T6, T7, T8, T9, T10}. The faults covered by each test case and execution time required are shown in Table 1.

The evaluation of proposed approach is carried out by assigned severity values and Table 2 shows the number of defects identified by each test case and the total time taken to detect the defects and severity values against those faults.

Table 1: Test cases with faults covered and execution time for student project

Test cases	Fault covered	Execution time (units)
T1	1, 5, 3, 4	5.2
T2	4, 2, 5	16.4
T3	2, 3, 5	8.2
T4	2, 1, 3	10.3
T5	4, 5	6.5
T6	3	7.1
T7	4, 1	3.7
T8	5, 2, 4, 3	14.8
T9	1, 2, 3	20.9

Table 2: Time taken to find defects and the severity value

Test case/fault	T1	T2	T3	T4	T5	T6	T7	T8	T9
F1	x			x			x		x
F2		x	x	x				x	x
F3	x		x	x		x		x	x
F4	x	x			x		x	x	
F5	x	x	x		x			x	
No. of faults	4	3	3	3	2	1	2	4	3
Time (msec)	5	16	8	10	6	7	4	15	21
Severity value	8	6	13	16	7	4	20	32	10

Execute the test case first which has highest severity values. The total severity becomes for the test cases as given below: T1 = 8, T2 = 6, T3 = 13, T4 = 16, T5 = 7, T6 = 4, T7 = 20, T8 = 32, T9 = 10. These severity values assign to these detected faults are shown in Table 2. According to the approach test cases will be executed in the order: T8-T7-T4-T3-T9-T1-T5-T2-T6.

The analysis of PFV and defect impact of two projects is shown in Table 3. PFV is calculated from all the test cases taken for a project are grouped into four factors. This result shows the higher percentage of defect instigated from the test cases along with PFV values range of 5 or higher. Figure 1 evident that the test cases with higher DI originate with higher range of PFV.

Effectiveness of the proposed technique: Here, we can compare the effectiveness of the proposed technique by the three validation metrics by means of random order execution and ordered execution (based on the proposed approach). Ten test cases are taken for comparison both in prioritized and in random order. Figure 1 shows that more number of defects can be detected in execution of test cases.

The evaluation results for test case prioritization technique are shown in Fig. 2. Validation metrics are defect severity, acceptable test case size, total prioritization time for both random and prioritization time is evaluated and shown in Fig. 3.

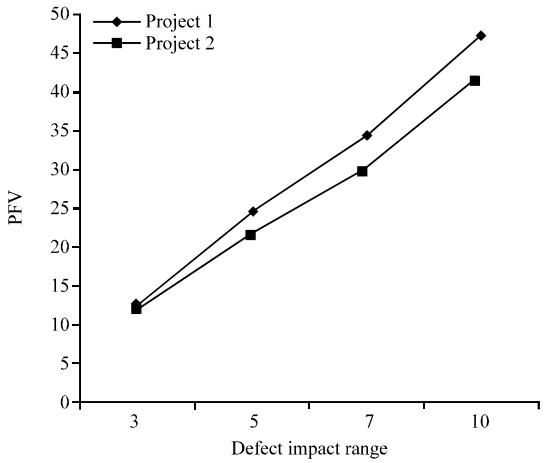

Fig. 1: PFV vs. defect impact range

Table 3: PFV range for test cases and defect impact

PFV values	Project 1 DI	Project 1 APFV	Project 2 DI	Project 2 APFV
1-3	12.0	10.0	4.5	14.0
3-5	23.5	12.4	19.8	12.6
5-7	32.8	35.2	27.5	27.8
7-10	45.2	36.8	39.2	30.5

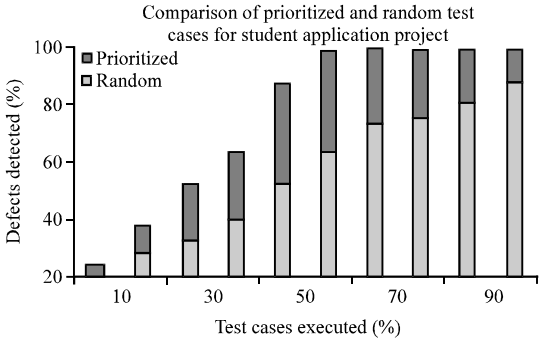

Fig. 2: TSFD is higher for prioritized test case reveals more defects

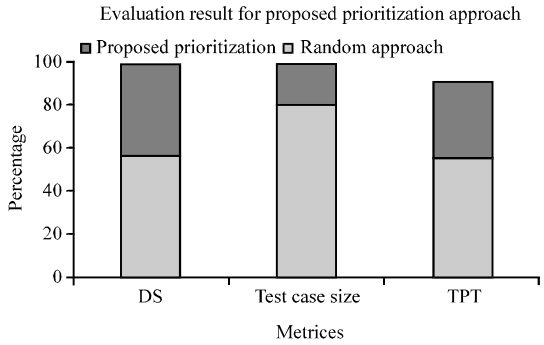

Fig. 3: Evaluation result of Test Case Prioritization Methods

CONCLUSION

Here, researchers propose a new prioritization technique for Prioritizing System level test cases and to improve the rate of fault detection in regression testing. Further, researchers have projected new practical set of weight factors that are used in the test case prioritization process. The new set of factors is composed of four groups: time factor, defect factor, requirement factor and complexity factor for regression test cases. The proposed Prioritization algorithm is validated by three validation metrics: defect severity, acceptable test case size and total prioritization time. Finally, the experimental results shows that this technique improves the rate of fault detection in comparison with random ordered test cases. Also, it reserves the large number of high priority test with least total time during a prioritization process.

REFERENCES

Agrawal, H., J.R. Horgan, E.W. Krauser and S.A. London, 1993. Incremental regression testing. Proceedings of the IEEE International Conference on Software Maintenance, September 27-30, 1993, Montreal, Quebec, Canada, pp: 348-357.

Elbaum, S., A. Malishevsky and G. Rothermel, 2002. Test case prioritization: A family of empirical studies. IEEE Trans. Software Eng., 28: 159-182.

Harrold, M.J. and G. Rothermel, 1997. Aristotle: A system for research on and development of program analysis based tools. Technical Report OSU-CISRC-3/97-TR17, Ohio State University.

Harrold, M.J., R. Gupta and M. Soffa, 1992. An approach to regression testing using slicing. Proceedings of the IEEE-CS International Conference on Software Maintenance, November 9-12, 1992, Orlando, FL., pp: 299-308.

Hung, Y.C., K.L. Peng and C.Y. Haung, 2012. A history based cost cognizant test case prioritization technique in regression testing. J. Syst. Software, 85: 626-637.

Jeffrey, D. and N. Gupta, 2006. Test case prioritization using relevant slices. Proceedings of the 30th Annual International Computer Software and Applications Conference, September 18-21, 2006, Chicago, USA., pp: 411-420.

Kim, J.M. and A. Porter, 2002. A history-based test prioritization technique for regression testing in resource constrained environments. Proceedings of the 24th International Conference on Software Engineering, May 19-25, ACM Press, pp: 119-129.

Malishevsky, A.G., J.R. Ruthruff, G. Rothermel and S. Elbaum, 2006. Cost-cognizant test case prioritization. Technical Report TR-UNL-CSE-2006-0004, Department of Computer Science and Engineering, University of Nebraska-Lincoln. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.112.9150.

Park, H., H. Ryu and J. Baik, 2008. Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. Proceedings of the 2nd International Conference on Secure System Integration and Reliability Improvement, July 14-17, 2008, Yokohama, pp: 39-46.

Qu, B., C. Nie, B. Xu and X. Zhang, 2007. Test case prioritization for black-box testing. Proceedings of the 31st Annual International Computer Software and Applications Conference, Volume 1, July 24-27, 2007, USA., pp: 465-474.

Rothermel, G., M.J. Harrold, J. Ostrin and C. Hong, 1998. An empirical study of the effects of minimization on the fault detection capabilities of test suites. Proceedings of the 14th IEEE International Test Conference on Software Maintenance, November 16-20, 1998, Bethesda, Maryland, pp: 34-43.

Rothermel, G., R.H. Untch, C. Chu and M.J. Harrold, 2001. Prioritizing test cases for regression testing. IEEE Trans. Software Eng., 27: 929-948.