Improving Requirements Coverage in Test Case Prioritization for Regression Testing

Rimsha Butool[1], Aamer Nadeem[1], Muddassar Sindhu[2], Qamar uz Zaman[1]
[1]Department of Computer Science, Capital University of Science and Technology, Islamabad, Pakistan
rimshabutool@yahoo.com, anadeem@cust.edu.pk, qamar.zaman@cust.edu.pk

[2]Department of Computer Science, Quaid-i-Azam University, Islamabad
masindhu@qau.edu.pk

Abstract: **Regression testing is performed whenever software undergoes modifications, which may be due to bug fixes or feature enhancements. The purpose of regression testing is to ensure that modifications to the code do not affect the existing functionality. Regression testing is costly because the test suite might be too large to execute in full. To reduce this cost, regression testing has three main approaches, i.e., test suite minimization, test case selection and test case prioritization. Test case prioritization does not eliminate any test case rather it finds an ordered list of test cases to maximize the fault detection rate. Black box prioritization prioritizes test cases based on requirements coverage, while white box approaches prioritize test cases based on code coverage. The focus of this paper is on the black box test case prioritization using requirements coverage for regression testing. Black box prioritization is independent of the code modifications, therefore, it can be started early. The proposed approach prioritizes test cases based on the complexity of requirements covered. The existing requirements based prioritization techniques assign equal importance to each requirement when prioritizing test cases based on requirements coverage, which may not maximize fault detection rate because complex requirements may need to be tested with multiple test cases. Our proposed approach assigns weights to the requirements based on their complexity, and test cases are prioritized using these weights. Comparison with existing approach shows that the proposed approach results in better prioritization of test cases because of higher average percentage of fault detection (APFD).**

Keywords— Regression testing; test case prioritization

I. Introduction

Software development is a complex process and no matter how well conceived and tested before released, software will eventually have to be modified in order to fix bugs or respond to changes in user specifications [1]. Regression testing is performed, whenever software modified and retested to ensure that the newly added features do not affect the existing behavior of the system [2]. Regression testing is a costly procedure and to avoid this costly procedure regression testing is divided into three main approaches, i.e., test suite minimization, test case selection and test case prioritization.

Test suite minimization used to eliminate the most redundant test cases from the test suite [3]. Test case selection selects the most relevant test cases from the test suite, which are valid and negotiate the changed parts of the program [4]. Test case prioritization does not eliminate any test cases somewhat tends to find an ordered list of test cases which gives the maximum benefits to the software testers. Test case prioritization aims to improve fault detection rate [5].

In this paper, our focus is on test case prioritization to find a prioritized list of test cases. Prioritization might be black box or white box. Black box prioritizes test cases based on requirements coverage to maximize the fault detection rate [6]. White box prioritizes test cases based on code coverage to maximize the fault detection rate [19].

Priority list ensures the early rate of fault detection. The focus of the paper is on black box prioritization, i.e., the requirements coverage. Coverage based approach is commonly utilized as a prioritization criterion and requirements coverage is an important domain in this exposure [21]. Thus, while the objective of test case prioritization leftovers that of accomplishing a higher fault detection rate using requirements coverage [7].

Requirement coverage based on the criteria that is incapacitating the critical problem related with the existing state-of-art approaches that some requirements are complex and for the complete coverage of complex requirement; there must be multiple test cases against a complex requirement [8].

In this paper, we propose a technique, which prioritizes the test cases based on the requirement coverage. In the proposed approach, there are multiple test cases against a complex requirement to cover a requirement completely. With the complete coverage of requirements fault detection rate also increased.

Complete coverage of requirements based on a phenomenon of many to many relationships. In many to many relationships, a test case might cover multiple requirements partially and each requirement coverage is considered against multiple test cases. If multiple test cases generated against a complex requirement, then for the

978-1-7281-4001-8/19/$31.00 ©2019 IEEE

complete coverage of a complex requirement all test cases against a complex requirement will be executed.

The rest of this paper is prearranged as follows: Section 2 deliberates the related work knowledge of existing approaches. Section 3 defines proposed approach. Section 4 describes result and discussion. Section 5 presents conclusion and possible future work.

II. Related Work

Arafeen and Do [9], introduced a contemporary approach that used requirements information to prioritize the test cases. Text mining approach is used in this technique to extract useful words. While focusing on those words, related requirements are clustered and test cases are prioritized in these requirement clusters. Within a cluster, test cases are prioritized using code complexity and then clusters are prioritized using code modification information and customer assigned requirements priority. At the end, after having prioritized test cases and clusters, by using different selection methods test cases are selected from clusters by visiting each cluster.

Hettiarachchi et al. [10], present an approach based on requirements and risks to prioritize test cases. The proposed technique consists of 5 steps: In the first step, they identified and evaluated the risks associated with each requirement. In next step, risk weight was calculated by using the factors produced after the first step. In third step risk exposure values are calculated and each risk is assigned a risk-exposure weight. In fourth step, additional factors are considered for prioritization which includes requirements modification information, requirement-modification level, and requirement-volatility. After collecting all the values from above steps, they prioritized requirements using risk-exposure weights and additional factors, after the prioritization of requirements a mapping between test cases and requirements is performed. Ergo, a prioritized list of test cases based on requirements prioritization is obtained

Krishnamurthy and Mary [11], introduce a requirements-based approach discuss the six factors customer priority, changes in requirement, implementation complexity, completeness, traceability and fault impact for calculating the weight of requirements. Their prioritization technique is validated with two analyses. The first analysis is conduct on number of faults detected and the second analysis consider the number of test cases executed to detect the faults. These factors prioritize the system test case [17]. The requirements with the highest priority will be assigned high priority; and compute each test case weight, by computing the fraction of requirement weight. Each of the test cases maps amongst total requirements weight of the project.

Kumar et al. [12], introduce a hierarchical test case prioritization approach that focuses on requirements. The prioritization process performed at three levels. First, each requirement assigned a priority based on 12 different factors: Higher the value of these factors, higher the priority of that requirement. After getting the prioritized list of requirements, a mapping between each requirement and its corresponding modules is performed. After getting the prioritized list of modules, a mapping between a module and its corresponding test cases is performed [18]. The test cases corresponding to modules are then prioritized based on 4 factors including test impact, test case complexity, requirement coverage, and dependency.

Kavitha et al. [13], system level test case prioritization from software requirement specification introduced to improve the rate of severe fault detection based on the three factors: customer priority, changes in requirement, implementation complexity. First, in this approach requirements weight is calculated by using these three factor values and the requirements weight is the mean of these factor values. After calculating the weight of all requirements, test cases weights are calculated; which is the fraction of sum of the mean of all requirements that a test case maps by the sum of the mean of total requirements. The test cases then prioritized and sorted in descending order of their weights. The higher is its (test case) weight; the highest is the priority of that test case.

Srikanth et al. [14], generates requirement based approach that is used to introduced a value based technique named as "Prioritization of Requirements for Testing (PORT)", which contains four factor values: Customer-assigned priority of requirements, developer-perceived implementation complexity, requirements volatility, and fault-proneness. Test cases arranged based on the weighted priority value of requirements. In such a way that test case with high weighted priority value is assigned highest priority and so on.

Tingting et al. [15], introduced an approach based on a test case prioritization algorithm based on the requirement correlations. The requirements priorities obtained from two factors: customer perceived priority (CP) and developer perceived priority (DP). First, requirements priority was calculated and then test cases priority had calculated from the requirements priorities. According to the total weight of each test case, every test case sorted in a prioritized list order.

Existing state-of-art approaches based on requirements coverage and some are requirements priorities based. However, contemporary approaches consider that a requirement might completely covered by one test case only [19]. Hence, some requirements are complex and need more than one test case to be covered completely. To overcome the crucial issue in existing research, a proposed approach is identified that considers complete coverage of requirements and considering the requirements priorities and requirements coverage [20].

III. Proposed Approach

After the critical analysis of related work, it leads towards a particular direction to identify the gaps in existing approaches and towards their solution. This paper focuses on the prioritization of test cases based on prioritized requirements. From the overall analysis of contemporary approaches, there exist one test case against a complex requirement and a single test case cover a complex requirement completely. However, some requirements are complex and need more than one test cases to cover completely.

Proposed approach depends on the complete coverage of requirements and following the strategy of achieving best performance when compared with existing approaches. Different three case studies are taken for the evaluation and comparison with existing criteria. Proposed criteria based on fraction of each requirement coverage and then calculate the test cases weight. At the end, assign priorities to test cases according to the ranking.

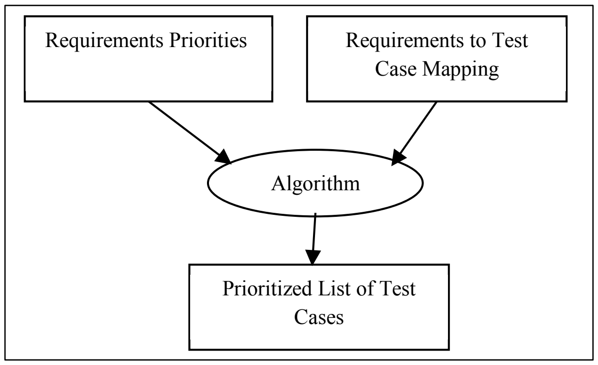
Requirements Priorities
Requirements to Test Case Mapping
Algorithm
Prioritized List of Test Cases

Fig. 1. *Flow of Proposed Algorithm*

Figure 1 delineated the working and control flow of proposed approach. Requirements priorities and requirement to test case mapping are takes as input in this approach like text editable file. Input is given to the algorithm file for further processing. The flow of proposed approach following that first of all fraction of each requirement coverage be there identified and evaluated against test cases. Fraction of each requirement coverage is premeditated using the following formula:

$$\mathbf{Cov\ (r_i, t_j) = 1/k}$$

where k represent test cases against a single requirement, "1/k" is represented as fraction of each requirements coverage and requirement "r_i" be covered by k test cases. The coverage of "r_i" by test cases is represented by "t_j".

In the second phase, Weights of Requirements Covered is premeditated by using the following formula, i.e.,

$$\mathbf{Wt.\ Cov\ (r_i, t_j) = 1/k * weight\ (r_i)}$$

Where 1/k is represented as fraction of each requirement coverage that is multiplied with the weights of requirements covered against these test cases. Weight is defined as priorities of each requirement; the preference of each requirements' weight depends on its complexity.

After calculating all these values in our proposed approach, Weighted Requirements Coverage is premeditated via the following formula, i.e., $\mathbf{Cov(t_j) = \sum_{i=1}^{k} Wt\_Cov(ri, tj)}$ to get a prioritized list of test cases. The proposed algorithm covers a complex requirement completely against multiple test cases. According to the proposed algorithm, equally updated weight is divided into multiple test cases against a complex requirement.

```
Prioritizing Test Cases based on Requirements Priorities

Input

  1. Requirements: $R_i$
  2. Weight of Requirements: $WR_i$
  3. Test Cases against Requirements: $T_j$
  4. i = $R_1, R_2, R_3 \ldots R_n$
  5. j = $T_1, T_2, T_3 \ldots T_j$
  6. X: A temporary set tests used for calculation.
  7. Req count: count the number of requirements covered by test
     case t.

Output

PrT: A prioritized list of test cases.

Procedure

Step#01: If X =T, find t in 'X', such that cov (t). weight $\geq$ 'X'.

Step#02: Repeat steps 2.1 to 2.3 while X` $\neq 0$ and req count $\neq 0$

Step#2.1: Find k = Number of test cases that cover each requirement
$r_i$:

Step#2.2: Calculate coverage of each requirement by test case $t_j$, such
that:        $cov(r_i, t_j) = 1/k$

Step#2.3: Find weighted coverage of each requirement by test case $t_j$
as:          $Wt\_Cov(r_i, t_j) = 1/k * weight(r_i)$

Step#03: Calculate total weight each test case $t_j$ as: $Cov(t_j) =$
$\sum_{i=1}^{k} Wt\_Cov(ri, tj)$

Step#04: Sort the test cases based on their total weight in descending
order as:

PrT = Sorted list of test cases
```

Fig. 2. *Proposed Algorithm*

Figure 2 Delineated the proposed algorithm that is built to prioritize test cases based on requirements priorities. For each test case, we first calculate the proportion of each requirement covered by the test case. These proportions are then multiplied by weights (priorities) of requirements. The sum of weighted requirements coverage is calculated as weight of each test case. Finally, the test cases are sorted in descending order based on their absolute weights. This gives us the prioritized list of test cases.

IV. Results and Discussion

We have used three case studies named as automated teller machine, online food ordering, and process flow manager for evaluating our approach. A brief description of each subject program is given below:

Automated Teller Machine (ATM) is a computer-based machine, which is connected to a network. It provides basic functions to the users, access to the bank accounts, transfer and retrieval of money. When the users complete the transaction then ATM returns to its idle state.

Online Food Ordering system will allow the restaurants to quickly and easily manage an online menu in the foodservice industry. The customer can browse and place an order with just a few clicks. A confirmation receipt generated for every order made by the customer.

Process Flow Manager is independent in their jobs, flow definitions, flow and time dependences, so that they can be reused. The Process Flow Manager administrator can generate calendars that can be used by any user of Process Manager. These are referred to as system calendars.

Table 1. Summary of Subject Programs

Programs	Requirements	Test Cases
Automated Teller Machine	20	16
Online Food Ordering	32	15
Process Flow Manager	35	19

A. *Evaluation and Comparison*

For the evaluation and comparison of proposed algorithm with existing algorithm, three case studies are selected. In these case studies, requirements priorities and requirements to test cases mapping are taken as input. By using these requirements priorities test case priorities are calculated.

We relate our proposed approach with the above-mentioned existing approach. Existing test case prioritization technique considers the number of requirements covered by each test case by using total coverage strategy. The test case that covers a maximum number of requirements and having highest priority is executed first. In existing approach, the impact of prioritization is that test cases with the similar number of requirements is inserted in prioritized list at the top and other test cases with some similar requirements move towards the bottom of the priority list [15].

In proposed approach, we assumed that some requirements are complex and need more than one test cases to be covered completely. However, proposed algorithm completely covers a complex requirement by assigning equally updated weights to all test cases against a complex requirement. Table 2 explains comparison that is plotted against the values of proposed and existing algorithm and in the graph difference between existing and proposed prioritization is plotted in a graphical representation of each case study. This table shows that the execution of prioritization list and also achieving complete coverage o requirements. The graph indicates that at early stages, more number of requirements are covered in our proposed approach.

Table 2. Difference of Weighted Requirements Coverage

Automated Teller Machine			
No. of Test Cases	***Existing Weighted Requirements Coverage***	***Proposed Weighted Requirements Coverage***	***Difference***
1	20.78	25.25	4.47
2	(20.78+20.61)= 41.39	(25.25+25)= 50.25	8.86
3	(41.39+25.25)= 66.64	(50.25+23)= 73.25	6.61
4	(66.64+12.43)= 79.07	(73.25+20.78)= 94.03	14.96
5	(79.07+25)= 104.07	(94.03+20.61)= 114.64	10.57
6	(104.07+12.2)= 116.27	(114.64+16.75)= 131.39	15.12
7	(116.27+12.21)= 128.48	(131.39+15.5)= 146.89	18.41
8	(128.48+12.61)= 141.09	(146.89+12.61)= 159.5	18.41
9	(141.09+8.2)= 149.29	(159.5+12.35)= 171.85	22.56
10	(149.29+16.75)= 166.04	(171.85+12.33)= 184.18	18.14
11	(166.04+9.25)= 175.29	(184.18+12.16)= 196.34	21.05
12	(175.29+23)= 198.29	(196.34+10.5)= 206.84	8.55
13	(198.29+15.5)= 213.79	(206.84+9.25)= 216.09	2.3
14	(213.79+8.96)= 222.75	(216.09+8.96)= 225.05	2.3
15	(222.75+7.75)= 230.5	(225.05+8.2)= 233.25	2.75
16	(230.5+10.5)= 241	233.25+7.75)= 241	0

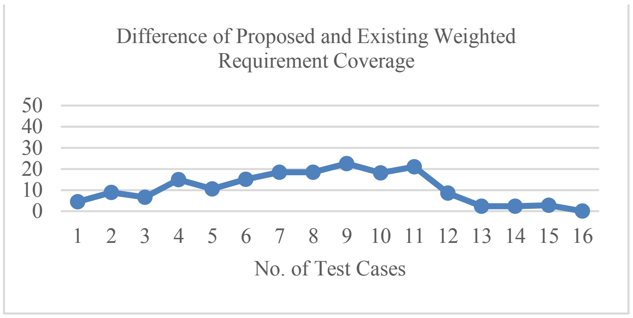

Fig. 3. *Automated Teller Machine*

Table 3. Comparison of existing and proposed approaches

Programs #	No. Test Cases	No. of fault seeded	No. of fault detected	APFD for Existing Approach	APFD for Proposed Approach
Automated Teller Machine	16	8	8	78.9%	81.25%
Online Food Ordering	15	7	7	80.4%	82.3%
Process Flow Manager	19	7	7	81.58%	83.8%

Figure 3 sheds a light on the graphical representation of requirements coverage for automated teller machine case study. This graph represents the number of test case on x-axis and weighted requirements coverage on y-axis. First of all we prioritized test cases against proposed and existing approaches and then executed all test cases. After the execution of all test cases, we plot a difference of proposed and existing approach. Whenever, the difference plotted graph move aloft against the number of test cases then this graphical representation shows proposed approach is providing a better coverage of requirements against existing approach. At the end, all test cases are executed and coverage is also completed against the test cases, then coverage of both the existing and prioritized test cases should be same.

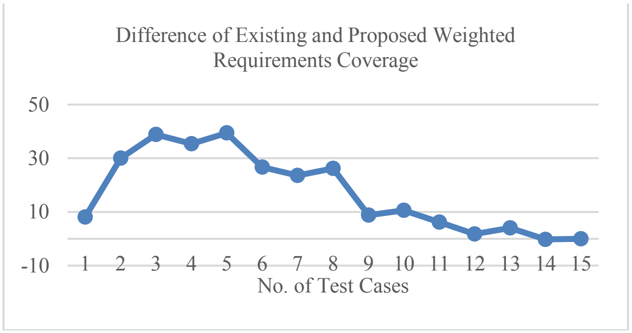

Fig. 4. *Online Food Ordering*

Figure 4 sheds a light on the graphical representation of requirements coverage for online food ordering case study. First of all we prioritized test cases against proposed and existing approaches and then executed all test cases. After the execution of all test cases, we plot a difference of proposed and existing approach. Whenever, the difference plotted graph move aloft against the number of test cases then this graphical representation shows proposed approach is providing a better coverage of requirements against existing approach. At the end, all test cases are executed and coverage is also completed against the test cases, then coverage of both the existing and prioritized test cases should be same.

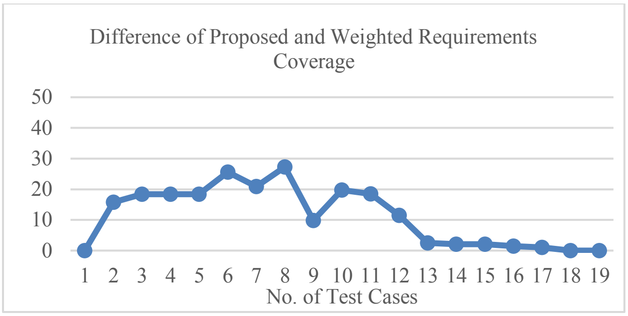

Fig. 5. *Process Flow Manager*

Figure 5 sheds a light on the graphical representation of requirements coverage for case study of process flow manager. Initially, we prioritized test cases against proposed and existing approaches and then executed all test cases. Afterwards, the execution of all test cases, we plot a difference of proposed and existing approach. Whenever, the difference plotted graph move aloft against the number of test cases then this graphical representation shows proposed approach is providing a better coverage of requirements against existing approach. At the end, all test cases are executed and coverage is completed against the test cases, then coverage of both the existing and prioritized test cases should be same.

B. *Comparison on the basis of APFD*

For APFD calculation, we have to seed faults in some requirements. APFD calculated by using following formula:

$$APFD = 1 - \frac{TF_1 + \cdots \ldots . . TF_m}{nm} + \frac{1}{2n}$$

Where T is the test suite including n test cases and F is the set of m faults revealed by the test suite T [15]. For ordering T', let TF_i be the order of the first test case that reveals the *ith* fault. For APFD calculation, faults are seeded randomly in requirements. For the calculation of APFD, only those faults, which are detected by the test suite, are considered and the faults that are undetected; should be ignored [17].

Figure 6 explains bar graph representation against APFD calculation for both existing and propose prioritization approaches. Proposed prioritization performs better as compared with existing algorithm.

For each of these programs, two priority lists are generated for existing and proposed prioritization approaches. Therefore, from the above table, it is concluded that our proposed approach is providing better coverage in terms of fault detection.

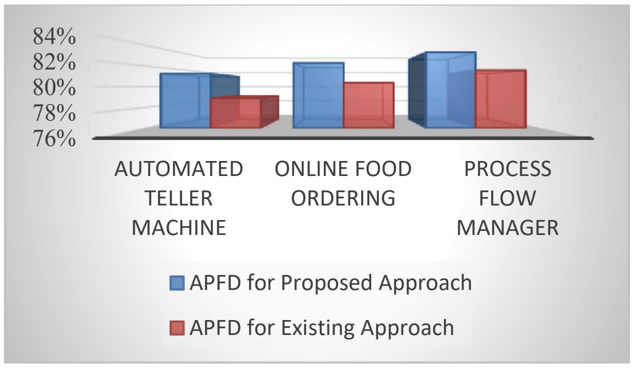

Fig. 6. *Comparison on the basis of APFD*

V. Conclusion and Future Work

In this paper, we have proposed a novel approach to prioritize test cases for regression testing that not only considers requirements coverage but also complexity of the covered requirements. The results show that the proposed prioritization results in higher average percentage of fault detection (APFD). In the current work, we assumed that if a requirement is covered by multiple test cases, then each test case equally covers a part of the requirement. However, in practice, requirement coverage by different test cases may vary. Further research is required to investigate how to find the proportion of requirement covered by each test case. Furthermore, coverage of a requirement by test cases may overlap. Thus, a more sophisticated prioritization algorithm would be required that would consider these aspects of requirements coverage. In addition, experiments on larger case studies is required to assess the usefulness of proposed approach.

VI. References

[1] L. Baresi, M. Pezze, “An introduction to software testing. Electronic Notes in Theoretical Computer Science,” vol.148, pp. 89-111, 2006.

[2] S.G. Elbaum, A. G. Malishevsky, G. Rothermel, “Test case prioritization: a family of empirical studies,” IEEE Transactions on Software Engineering 28(2), pp. 159-182, 2002.

[3] G. Rothermel, M.J. Harrold, "A framework for evaluating regression test selection techniques,” Proceedings of the 16th International Conference on Software Engineering (ICSE), IEEE Computer Society Press, pp. 201-210, 1994.

[4] Rothermel, G, Harrold MJ. (1994). A framework for evaluating regression test selection techniques. Proceedings of the 16th International Conference on Software Engineering (ICSE), IEEE Computer Society Press, pp, 201–210.

[5] Tahvili, S., Afzal, W., Saadatmand, M., Bohlin, M. (2016). Towards earlier fault detection by value driven prioritization of test cases using fuzzy TOPSIS. Information and Software Technology.

[6] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, “Test case prioritization: An empirical study,” Proceedings of International Conference on Software Maintenance (ICSM), IEEE Computer Society Press, pp.179-188, 1999.

[7] G. Rothermel, R.H. Untch, C. Chu and M.J. Harrold, “Prioritizing test cases for regression testing,” IEEE Transactions on software engineering, 27(10), pp. 929-948, 2001.

[8] M. Yoon, E. Lee, M. Song and B. Choi, “A test case prioritization through correlation of requirement and risk,” Journal of Software Engineering and Applications, 5(10), p. 8-23, 2012.

[9] M. J. Arafeen, H. Do, “Test case prioritization using requirements based clustering,” International Conference on Software Testing, IEEE Computer Society Press, pp. 312321, 2013.

[10] C. Hettiarachchi, H. Do and B. Choi, “Effective regression testing using requirements and risks.” In Software Security and Reliability (SERE), Eighth International Conference on pp. 157-166, IEEE, June 2014.

[11] R. Krishnamoorthi, and S.S.A. Mary, “Factor oriented requirement cover age based system test case prioritization of new and regression test cases,” Information and Software Technology, 51(4), pp.799-808, 2009.

[12] H. Kumar, V. Pal, N. Chauhan, “A hierarchical system test case prioritization technique based on requirements,” 13th Annual International Software Testing Conference, pp. 45, 2013.

[13] R. Kavitha, V.R. Kavitha and N.S. Kumar, “October. Requirement based test case prioritization,” In Communication Control and Computing Technologies (ICCCCT), IEEE International Conference on pp. 826-829, IEEE. October, 2010.

[14] H. Srikanth, S. Banerjee, L. Williams and J. Osborne, “Towards the prioritization of system test cases,” Software Testing, Verification and Reliability, 24(4), pp.320-337, 2014.

[15] Ma Tingting, Zeng. Hongwei, Wang. Xiaolin, “Test Case Prioritization based on Requirement Correlations,” School of Computer Engineering and Science, Shanghai University, pp. 1-6, 2016.

[16] H. Do, G. Rothermel, “A controlled experiment assessing test case prioritization techniques via mutation faults,” Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM), IEEE Computer Society Press, pp. 411-420, 2005.

[17] T.Y. Chen, M.F. Lau, “Dividing strategies for the optimization of a test suite,” Information Processing Letters, 60(3), pp. 135-141, 1996.

[18] M.J. Harrold, R. Gupta, M.L. Soffa, “A methodology for controlling the size of a test suite,” ACM Transactions on Software Engineering and Methodology, 2(3), pp. 270-285, 1993.

[19] Henard, C., Papadakis, M., Harman, M., Jia, Y. and Traon, Y.L. (2016). Comparing White-box and Black-box Test Prioritization. In Proceedings of the 38th International Conference on Software Engineering (ICSE). ACM

[20] A. J. Offutt, J. Pan, J. Voas, “Procedures for reducing the size of coverage based test sets,” Proceedings of the 12th International Conference on Testing Computer Software, ACM Press, pp. 111-123, 1995.

[21] W. E. Wong, J. R. Horgan, S. London, H. Agrawal, “A study of effective regression testing in practice,” Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE), IEEE Computer Society, pp.264-275, 1997.