International Journal of Engineering & Technology, *7 (2.28) (2018) 332-336*

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Test case prioritization and selection technique in continuous integration development environments: a case study

Lei Xiao [1, 3] *, Huaikou Miao [1, 2], Ying Zhong [3]

[1] School of Computer Engineering and Science, Shanghai University, Shanghai, 200444, China
[2] Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai, 201112, China
[3] College of Computer and Information Engineering of Xiamen University of Technology, Xiamen, 361024, China
**Corresponding author E-mail: : lxiao@xmut.edu.cn*

Abstract

Regression testing is a very important activity in continuous integration development environments. Software engineers frequently integrate new or changed code that involves in a new regression testing. Furthermore, regression testing in continuous integration development environments is together with tight time constraints. It is also impossible to re-run all the test cases in regression testing. Test case prioritization and selection technique are often used to render continuous integration processes more cost-effective. According to multi objective optimization, we present a test case prioritization and selection technique, TCPSCI, to satisfy time constraints and achieve testing goals in continuous integration development environments. Based on historical failure data, testing coverage code size and testing execution time, we order and select test cases. The test cases of the maximize code coverage, the shorter execution time and revealing the latest faults have the higher priority in the same change request. The case study results show that using TCPSCI has a higher cost-effectiveness comparing to the manually prioritization.

Keywords: *Software Process; Software Development Life Cycle; Traditional Models; Agile Models; Evaluation Metrics.*

1. Introduction

Nowadays, iterative or incremental software development model have become very popular, which is considered as an essential method for improving efficiency and quality, reducing the cost and shortening the cycle of software development. Under the development circumstance, engineers merge code at frequent time intervals [7], [6]. The integrate work involves many tasks, including software configuration management, version control, automatic build and testing [17]. Before a new or changed code is submitted to the code repository, the tester firstly test the code on the locally machine, then submitting to the code base, and all the modules affected by the new or change code will be implemented regression testing in available time. So regression testing become very important and frequent in continuous integration development environment, where tester quickly provide faults reports are a main goal with a tight testing time constraint. Therefore it is essential to looking for a cost-effective regression testing technique to keeping the cost of software testing at a low level, The most straightforward techniques to improving the effectiveness of regression testing is test case prioritization and test case selection technique [19], which is a trade-off approach to maximize the quantity of test cases and find more faults in available testing time.
Test case prioritization and selection technique has been studied for a long time [19], [13], [16], [15], [3], [12], [8], [14]. As a “traditional” test case optimization approach which is mainly applied to detected faults as early as possible and reduce the test execution time in regression testing. Without loss of generality, the approach also can be used in continuous integration development environment. Nevertheless, the integration process has its particularity, for example, the faults revealed by the recently, namely current-1, continuous integration cycle have the highest priority than all the previous cycles. For rigorous time control in continuous integration environments, the test cases which spend the shorter execution time will be firstly executed under the same faults detection ability. Admittedly, selecting the test cased that execute quickly will result in increasing the number of executed test cases within a given time. In addition, most of the existing test case prioritization and selection technique show that the test case having find faults in the past are more likely find faults in the future [2], [1], based on this, it is a good strategy to select faults history information as one of prioritization criteria.
In this paper, we devote ourselves to seek to an approach used in continuous integration software development environments to prioritize test cases and acquire its execution subset that can detect as many faults as possible in a given time. The approach is called Test Case Prioritization and Selection Technique in Continuous Integration Development Environments (TCPSCI). Finally, we implement a case study using publication data sets from Google to investigate the technique effectiveness. The study results show that our approach can improve the regression testing faults detection rate and execute the more test cases within a limit time.
In general, it is difficult to acquire the code feature corresponding to test cases in system level testing, so the TCPSCI approach just consider four inputs: (1) Launch time: the beginning time of each change (2) Execution time: the execution time of each test case (3) Change request: the number of each continuous integration cycle (4) Size: the size of the code corresponding to test case, the values are SMALL, MEDIUM and LARGE. All above inputs are available from the Google publication data sets. We conjecture that the failure history of test cases are more likely to find faults in new continuous integration cycle regression testing, however, not all the failure history test cases have the same priority in continuous

integration environments, the test case have the highest in recently continuous integration cycle than the previous. This is because when the new test cases are created test the new feature of the software and maybe the other test cases are deleted for the feature they tested are obsolete. Furthermore in the same continuous integration cycle the test cases with shorter execution time and larger size have a higher priority.

The main contributions of this paper are following:

1) We proposed an algorithm to re-order the test cases executing in regression testing in continuous integration software development environments, which based on the schedule of continuous integration cycle, the execution time and the code size coverage by the link test case.
2) We implement a case study to evaluate the algorithm, using representative open data sets from a famous software company. The results show that the approach we proposed can improve the effectiveness of regression testing in continuous integration software development environments.

The rest of this paper is organized as follows. Section 2 introduces the background and related work about regression testing, test case prioritization and test case selection, section 3 describes the main idea of the TCPSCI algorithm, section 4 discuss the result of the case study. Finally, section 5 concludes this paper and gives the research directions in the future.

2. Background and related work

2.1. Regression testing

Regression Testing will to be performed when software changes, it aim to avoid the new components destroy the existing components. Therefore, in continuous integration development environment, regression testing execution become more frequently. The definition of regression testing process is as following:

1) Select T^{0} T and use T^{0} to test P^{0} in order to establish the correctness of P' about T^{0}.
2) Create a set of new functional or structural tests T^{00} for P^{0} if necessary, and use T^{00} to test P^{0} in order to establish the correctness of P' about T^{00}.
3) Create T^{000}, a new test suite, and test history for P' from T , T^{0} and T^{00} respectively. As shown in above process, P^{0} is a modified version of P, and P is the current version of the program. T is a test suite of P. The method of selecting T^{0} is important [13].

2.2. Test case prioritization and selection problem

Re-running all the test cases is the most straightforward method that can ensure that all faults will be detected of the software. However, re-running all the test cases is very expensive, it is almost impossible because of the limitation of time and resources. So, many researchers present the other techniques to improve the effectiveness of regression testing, such as test case prioritization and test case selection .The former is using a certain criteria to reorder test cases to increase the chance of early fault detection [16]. The latter is given P^{0} is the modified version of program P, T is a test suite of satisfaction P, find a subset of T, T^{0}, with which to test P^{0}. The complete definition of test case prioritization and test case selection are given by Yoo et al [19].

In recent decades, there are a number of approaches on test case prioritization that have been proposed. For instance, Rothermel et al. proposed the technique based on the statements and branch coverage of test cases [16], [15]. Qu Bo et al. proposed technique based on test suit design information [12]. Yang et al. put forward test cases prioritization based on the requirement [3]. Korel et al. presented model-based test prioritization which goal is for early fault detection in implementations of model changes in the system [3]. Recently, Henard et al. compared white-box and black-box test prioritization and found the difference between black-box and white-box performance [8].

Because test case prioritization technique just consider the order of test cases, do not omit test cases, in cases omit some test cases is acceptable. Alternatively, test case selection is used that based on test case prioritization when time and cost are more limited. To date, there have some work about test case selection techniques have been proposed, which based on data-flow analysis, symbolic execution or code coverage data approaches [19]. For example, Rothermel et al. designed algorithms which construct control flow graphs for system under testing and its modified version and use these graphs to select tests that execute changed code from the original test
suite [14], [20].

2.3. Regression testing techniques in continuous integration development environments

There have a few work to consider regression testing in continuous integration development environments [20], [5], [9], [18], [11]. The most relevant to our research, Elbaum et al [5] divide the regression process into pre-submit and post-submit phase, consider continual regression test selection in pre-submit phase and regression test prioritization in post-submit. They do the experiment using the Google shared date set of test suite results. Yoo et al [20] present an approach using multi-objective regression testing optimization: dependency coverage, fault history, execution time and failing tester, in Google's continuous integration of post-submit test phase .They illustrate that the regression testing time can be reduced by between 33%-82%, using the technique. Spieker el al [18] proposed a test prioritization and selection approach based on reinforcement learning. It is a lightweight technique which mainly uses reinforcement learning to select and prioritize test cases. They just consider the duration, failure history and previous last execution of test cases in continuous integration development environments, not included the size of code corresponding to test case. The essay: data visualization-google shared dataset of test suite results reported by Fishelovich, Busany et al. show that the probability of finding faults of code size are large, medium and small from high to low in turn. Our work regard code size link to test case as one of a prioritize index.

3. Our methodology

3.1. Regression testing in continuous integration development environments

The integration of each time will invoke a regression testing in the software life cycle. Especially in continuous integration development environments, frequent building process will results in more regression testing. The main character of continuous integration development process is quick feedback. At each cycle, the new source code is submitted to the code-base, all regression testing will be completed under the ideal circumstance, but in fact, it is impossible, because of tight time control mechanism in integration development environments. Just re-execution part of test cases is the best choice in order to improve productivity instead of re-running all the test cases in most of software companies.

3.2. Continuous test cases prioritization and selection

The goal of test cases prioritization is to find an appropriate regression testing execution order for revealing the faults early. Test cases selection aim to find a subset of test cases in order to reveal more faults when just executing part of test case in regression testing. In continuous integration development environments, test cases grow quickly, regression testing is frequently and development time is constraint. So trying to maximize the execution quantity of the test cases and find more faults at the given testing time domain is a good strategy.

In our work, given the set of test cases $T = fT_1; T_2; ; T_ng$, all the test cases belong to different change request, before regression testing, all the test cases have been executed at least one time. Every test cases have the following attributes after executing system testing in continuous integration development environments: change request, launch-time, execution-time, size of code corresponding to the test case and execution status: PASSED or FAILED. Generally, we consider that a test case with high coverage rate have the higher priority. What's more, based on history execution information test case prioritization technique [11] shows the hypothesis is right, in most situation, if a test case find faults in the past, it more likely find faults in the future, as a result, the test case which has failed status precede over the other which have passed status. The test cases of the maximize code coverage, the shorter execution time and revealing the latest faults have the higher priority in the same change request. In the context, the continuous test cases prioritization and selection strategy is described as following, the test cases prioritization objective function f is defined in Eq. (1):

$$f = (\text{maximize } (p); \text{minimize}(ET)) \tag{1}$$

Where p is the priority values of each test case and ET is the execution time respectively. The problem of test cases prioritization is to find the test cases execution order:

$$8T_i \,(i = 1; \quad ; n)\ f(T_i) \quad f(T_{i+1})$$

Due to "quick feedback and fix" in continuous integration development environments, we hope the test case which can find faults and have shorter execution time can execute early. To address this problem, we need to hunt for suitable prioritization criterion and execution time of each test case, once the test case has been executed; it is easy to acquire the execution time. The key to the problem is to solve the maximize p traditionally; we believe that the test cases have the higher priorities which have detected faults in the previous test process. In changing development environments, constantly create new test cases and delete obsolete test cases. Therefore we consider that the test cases of find faults in the most recent test execution are superior to the others which find faults at earlier than before. So the details of our approach are described as following:

Step1 acquire the test cases in the same change request Step2 reorder the change request according to the launch time of each change request, the higher priority is, the later the launch time is.

Step3 the test cases belong to the same continuous integration cycle, namely, have the same change request number, the test case prioritization criterion is firstly schedule the failure history test cases, get From the value of status, if the test cases have same status, we consider the size and the execution time, the large size and the shorter execution time have the higher priority.

For example, suppose we have ten test cases which belong to three test change request(integration cycle)respectively, shown in Table 1. Among them, change request 1 include test case 1, 2, 3, change request 2 include test case 4, 5 and change request 3 include test case 6, 7, 8, 9, 10. Launch time is the start time of test execution, and execution time is the time of executing each test case. The size represents the code size corresponding to test case, and the status is the result of test execution, failed or passed.

According to our priority technique, in the recent regression testing, the execution order of the ten test cases is illustrated by the Fig. 1.

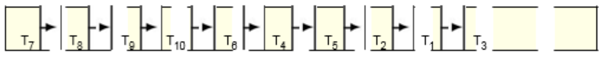


Fig. 1: The Execution Order Using the Test Case Prioritization.

From the Table 1, Change Request 3 the new continuous integration cycle, it has the highest priority than Change Request 2 and Change Request 1,and T_7 and T_8 are the last execution test cases of finding faults, they have the same size, but the execution time of TT_7 is shorter than T_8, in the light of our rule, T_7 is superior to T_8. Because T_6-T_{10} are in a same test suites, T_6, T_9 and T_{10} are superior to the other test case. The size of T_9 is larger than T_6 and T_{10}, so T_9 have the best priority comparing to T_{10} and T_6.In addition to, the execution of T_{10} is shorter than T_6, T_{10} is in front of T_6. Generally, the rest of the test case in Change Request 1 and Change Request 2 have the same rule.

3.3. The measures of our technique

We need a performance indication to evaluate our technique performance, in this paper, we adopt the same evaluation metric as literature [17]. It is Normalized APFD, different from the traditional APFD index [10], the Normalized APFD consider the probability of test case selection and its definition in Eq. (2).

$$NAPFD(TS_i) = p \quad \frac{\text{å}_{t2TS_i^{fail}} rank(t)}{jTS_i^{fail}j\, jTS_ij} + \frac{p}{2\, jTS_ij} \tag{2}$$

Where $p = \dfrac{jTS_i^{fail}j}{jTS_{total_i}^{fail}j}$, TS_i is the subset of T and it is a

Ordered sequence (TS_i T), rank(t) is the execution order of each test case, rank: TS_i ! N, in other word, rank(t) is the position of t in TS_i, and TS_i^{fail} is the subset of TS_i which includes the test case finding faults in the current regression testing, and $TS^{total_i;\,fail}$ includes all the test cases which can detected faults with T . If the regression testing execute all test case, $TS^{total_i;\,fail}$ equal to TS_i^{fail} , p is 1 and NAPFD is the same as APFD, the regression testing just adopt test case prioritization technique not use test case selection technique. From the evaluating indicator, we find that the greater the NAPFD value is, the better the technique performance is.

4. Empirical study

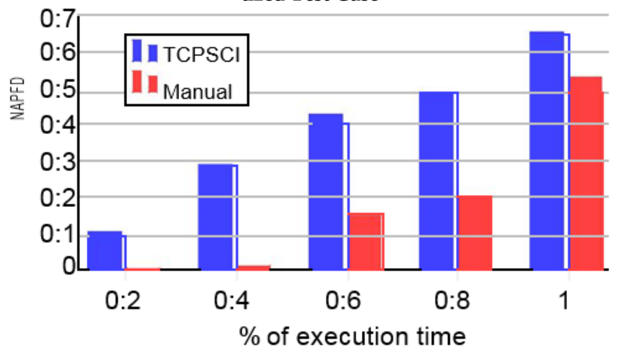
Our work is aimed to evaluate whether the test case prioritization and select technique can find more faults in regression testing when the test execution time is limit in continuous integration development environments. This section we introduce a case study to investigate the effectiveness of our approach, we complemented the experiments is to answer the questions as follows:

1) Can our approach execute as many test cases as possible when the test time is limit?
2) Can our approach firstly execute the test cases which find faults?

In order to finish the experiments, we use open data sets from Google. The dataset is Google Shared Dataset of Test Suite Results (GSDTSR) [4], that is available for use by the software testing researcher, It contains 3.5 million test suite execution results, gathered over a period of 30 days, the dataset details is described in the literature [5], and the dataset include information can satisfy to our experimental requirements. We conduct our experiment according to the percentage of test case execution and the percentage of testing execution time from 20% to 100%, and choose the NAPFD values and the number of detected faults as evaluation indicator. Comparing the manual and TCPSCI technique the results of the study are given in Table 2. The execution time on the percentage of the test cases with NAPFD is shown in Table 2(a), while the number of faults detected is given in Table 2(b).

The results are also depicted in Fig. 2 (a) and 2 (b), respectively.

(A) NAPFD Difference for Manually-Prioritized and TCPSCI Prioritized Test Case



(B) Number of Detected Faults for Manually-Prioritized and TCPSCI Prioritized Test Case

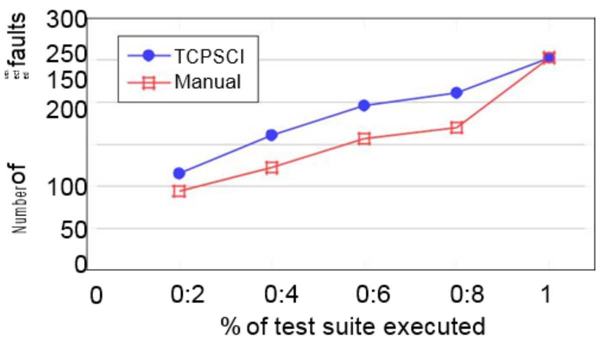


Fig. 2: Test Case Execution Time and Detected Faults.

The study results indicate that the TCPSCI approach can improve the rate of fault detection when the execution time

Table 1: An Example of Testing Execution Information

Change Request	Test Case	Launch Time	Execution Time	Size	Status
1	1	1:00:00:02	13899	LARGE	PASSED
1	2	1:00:00:30	23390	LARGE	FAILED
1	3	1:00:00:33	35789	LARGE	PASSED
2	4	1:00:00:34	15204	LARGE	PASSED
2	5	1:00:00:34	22488	LARGE	PASSED
3	6	1:00:00:35	19646	SMALL	PASSED
3	7	1:00:00:56	582	SMALL	FAILED
3	8	1:00:00:56	7125	SMALL	FAILED
3	9	1:00:00:58	3454	MEDIUM	PASSED
3	10	1:00:00:58	6241	SMALL	PASSED

a) NAPFD percentage test case execution time.

Table 2: Test Case Execution Time and Detected Faults

Percentage of test case execution time						
		20%	40%	60%	80%	100%
Manual	NAPFD	0.0052	0.0114	0.149	0.201	0.529
TCPSCI		0.1070	0.2890	0.427	0.486	0.652
	20%	40%	60%	80%	100%	
Manual	94	122	157	170	253	
TCPSCI	115	161	196	211	253	

b) Number of detected faults based on the percentage of test cases executed

Percentage of test case execution time is limit. Especially execution time is 60%, the NAPFD value is 0.149 versus 0.427 comparing to manual order and the prioritized-order. Furthermore, from Fig. 2(b), we can see that the test case prioritized by the TCPSCI technique always detect more faults comparing the manual order, the percentage of test case execution is 20%, 94 versus 115, 40%, 122 versus 161. The results positively answer the question (2). In summary, applying the TCPSCI technique to regression testing will find more faults in a given time We introduce the test case prioritization and selection approach in continuous integration development environments and use an open dataset from Google to conduct our case study. From the results of the study we can see that the technique can find more faults within a shorter execution time, so it can improve the cost-effectiveness of regression testing. Especially in continuous integration development environments, when test suites grow quick and time is tight limit, it is a suitable approach to use in regression testing. However, our technique have some limitation, such as the evaluate indicator is simple and the case study just using a data set from one company. In additional, we don't compare the technique to the criteria provided by the other literature [9], [18], [11]. Above all are our study directions in the future.

Acknowledgments

This research work is supported by National Natural Science Foundation of China (61572306, 61502294) and Fujian Province Universities and Colleges under Grants (No.JK2015033).

References

[1] M. J. Arafeen and H. Do. Test case prioritization using requirements-based clustering. In Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, ICST '13, pages 312–321, Washington, DC, USA, 2013. IEEE Computer Society.
[2] R. Carlson, H. Do and A. Denton. A clustering approach to improving test case prioritization: An industrial case study. In Proceedings of the 2011 27th IEEE International Conference on Software Maintenance, pages 382–391, Washington, DC, USA, 2011. IEEE Computer Society.
[3] X. Chen, J.-H. Chen, X.-L. Ju, and Q. Gu. Survey of test case prioritization techniques for regression testing. Journal of Software, 24(8):1695–1712, 2014.
[4] S. Elbaum, A. Mclaughlin, and J. Penix. The google dataset of testing results. https://code.google.com/p/ google-shared-dataset-of-test-suite-resutls, 2014.
[5] S. Elbaum, G. Rothermel, and J. Penix. Techniques for improving regression testing in continuous integration development environments. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, pages 235–245. ACM, 2014.
[6] M. Fowler. Continuous integration. https: //www.martinfowler.com/articles/ continuousIntegration.html, 2006.
[7] Glover, P. M. Duvall, and S. Matyas. Continuous Integration: Improving Software Quality and Reducing Risk. Pearson Education, 2007.
[8] Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon. Comparing white-box and black-box test prioritization. In Proceedings of the 38th International Conference on Software Engineering, pages 523–534. ACM, 2016.
[9] P. Kandil, S. Moussa, and N. Badr. Cluster-based test cases prioritization and selection technique for agile regression testing. Journal of Software Evolution & Process, 29(6), 2017.
[10] G. Malishevsky, G. Rothermel, and S. Elbaum. Modeling the cost-benefits tradeoffs for regression testing techniques. In Proceedings of the International Conference on Software Maintenance, pages 1–10, 2002.
[11] D. Marijan, A. Gotlieb, and S. Sen. Test case prioritization for continuous regression testing: An industrial case study. In Proceedings of the 2013 IEEE International Conference on Software Maintenance, ICSM '13, pages 540–543. IEEE Computer Society, 2013.
[12] B. Qu, C.-H. Nie, and B.-W. Xu. Case prioritization based on test suite design information. Chinese Journal of Computers, 31(3):431–439, 2008.
[13] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 22(8):529–551, 1996.
[14] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. ACM Transactions on Software Engineeering Methodology, 6(2):173–210, 1997.
[15] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: An empirical study. In Proceedings of the International Conference on Software Maintenance, pages 179–188, 1999.

[16] G. Rothermel, R. J. Untch, and C. Chu. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 27(10):929–948, 2001.
[17] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 12–22. ACM, 2017.
[18] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2017, pages 12–22. ACM, 2017.
[19] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. Software Testing Verification & Reliability, 22:67–120, 2012.
[20] S. Yoo, R. Nilsson, and M. Harman. Faster fault finding at google using multi objective regression test optimisation. In Proceedings of ACM SIGSOFT Symposium on the Foundations of Software Engineering, FSE 2011. ACM, 2011.