Extracting Historical Information and Fault Localization Information that may Contribute to the Effectiveness of Test Case Prioritization

Chu-Ti Lin, Sin-Ting Guo, Jiun-Shiang Wang, Chang-Shi Tsai
Department of Computer Science and Information Engineering
National Chiayi University
Email: {chutilin, s1010434, s1010437, s1000437}@mail.ncyu.edu.tw

***Abstract*—The goal of test case prioritization is to schedule the test cases in an order so that the faults in the software will be detected early during the regression testing. So far, a lot of test case prioritization techniques have been proposed and received considerable attention. However, there is currently no universally best approach because the existing techniques prioritize test cases under different considerations. Thus, we plan to propose a hybrid technique that takes into account more than one consideration and hope that this hybrid technique can further improve the effectiveness of test case prioritization. Although we did not construct the hybrid technique yet, we already have some preliminary experimental results that are collected from an investigation into real subject programs and their associated test suites. Therefore, this paper is dedicated to presenting our preliminary observations in this ongoing research project. Additionally, this paper also presents two assumptions based on the observations. These two assumptions should be useful for proposing the hybrid test case prioritization techniques in the future.**

Keywords-software maintenance; regression testing; test case prioritization; fault localization

I. INTRODUCTION

In software development/maintenance process, a lot of test cases will be designed and executed to ensure the correctness of software functions. Each test case consists of a set of inputs and the expected results, and the collection of test cases is called a test suite. It is inevitable that the functionality of a software system may change during software development/maintenance. Each time the software is modified, regression testing is necessary to ensure the quality of the software system. That is, the software testers or the test harness execute test cases to ensure the correctness of the new functions and the original test cases should also be re-executed to guarantee that all of the unmodified functions still work correctly [1]. Once they reveal the faults, the software developers can start to locate the cause of the failure and design the fix.

Unfortunately, it is difficult to execute all test cases in a short time. If the faults can be revealed early during the regression test, the software developers can start to deal with the faults early and the efficiency of software development may then increase. To address this problem, test case prioritization techniques are proposed to improve the effectiveness of regression testing by ordering the test cases, so that the most beneficial test case is executed first and software faults will be detected in early stage. Let T be a test suite, PT be the set of all possible permutations of the test cases in T, and f be a function from PT to the real numbers, which is used to evaluate the effectiveness of the prioritization. Elbaum et al. [2] defined the test case prioritization problem as finding $T' \in PT$ such that $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$. In other words, the goal of test case prioritization is to schedule the test cases in an order so that the tests with better fault detection capability can be executed at an early position in the test suite.

So far, many test case prioritization techniques have been proposed (e.g., [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]). Those techniques prioritize test cases under various considerations and the focuses of their empirical studies are diverse. There is currently no universally best approach that can work well for all cases. Thus, it may be interesting to propose a hybrid technique that takes into account more than one considerations to improve the effectiveness of prioritization. Although we did not construct a complete hybrid test case prioritization technique yet, we have some preliminary observations and experimental results. Thus, this paper will present our preliminary results of the research project.

The remainder of this paper is organized as follows. Section II briefly reviews several kinds of well-known test case prioritization techniques. Section III explains our motive and idea about how to improve the effectiveness of test case prioritization, and further shows the investigation into the historical information and fault localization information of the subject programs and the associated test suites. Section IV furnishes some concluding remarks.

II. REVIEW OF TEST CASE PRIORITIZATION TECHNIQUES

A. Code-based test case prioritization

The majority of the existing test case prioritization approaches are code based and they schedule test cases according to the results of the analyses on the source code elements of the program [3], [4], such as statements, basic blocks, branches, define-use pairs, functions. The underlying assumption for most of the code-based techniques is that a test case has a better fault detection capability if it covers more code elements. Thus, these

techniques give a higher priority to the test case that covers more code elements than others. In addition to the total coverage achieved by a test case, Elbaum et al. [2] also suggested the additional coverage for prioritizing test cases, i.e. it only takes into account the code elements that were not yet be covered by the previous test cases. The greedy algorithm associated with either of the total coverage and additional coverage is frequently used for prioritizing test cases [4], [6].

B. History-based test case prioritization

The code-based prioritization techniques shown in Section II.A only consider the code elements collected from the program of a specific software version, not the information collected across multiple versions. However, [7], [8], and [10] indicated that the test of the previous software versions may produce some data that are useful for prioritizing the test cases in the test suite of the current software version. Thus, several researchers have proposed history-based test case prioritization techniques [7], [8], [10]. For example, Kim and Porter [7] proposed a technique to prioritize the test cases using the test results of the previous software versions. In recent years, Liu et al. [8] also tried to prioritize the test cases based on the information from both historical data and code elements. Moreover, Lin et al. [10] posited that the reference value of the immediately preceding test results should be version-aware and thus presented a version-aware history-based test case prioritization technique.

C. Prioritizing test cases based on fault localization information

Statistical fault localization is a technique that automatically locates the faults in the program, thus significantly reducing the time taken to remove the faults. Statistical fault localization techniques determine the fault-prone code elements by analyzing the execution results and coverage achieved by all of the test cases in the test suite, and then schedule the code elements in an order accordingly. As the result, if there is a high probability that faults are located in a code element, this code element will be examined early in the debugging process. So far, several statistical fault localization techniques have received considerable attention [12], [13].

In recent years, several test case prioritization techniques schedule test cases according to statistical fault localization information. For example, Kim and Baik [11] indicated that if there is a high probability that faults are located in a specific code element in the immediately preceding software version, then the probability that faults exist in the same code element in the successive software version should be low. Based on this concept, Kim and Baik [11] proposed a fault-aware test case prioritization technique called FATCP. With the exception of the first software version, FATCP uses the greedy algorithm to schedule the test cases based on the Tarantula fault localization information that is collected from the test result of the immediately preceding software version. It is noted that, for the first software version, FATCP still schedule the test cases based on the code coverage because no historical information is available.

III. IDEAS AND PRELIMINARY RESULTS

A. Motive and ideas of improving the effectiveness of test case prioritization

The technique FATCP reviewed in Section II.C assigns a priority to each test case based on fault localization information, and then prioritizes the test cases accordingly. The empirical studies in [11] showed that incorporating fault localization information into test case prioritization techniques indeed improves the fault detection capability. Although this technique uses the test result of the immediately preceding software version to calculate the priority of each test case, but it does not consider the test results of the other previous versions. However, the history-based prioritization techniques reviewed in Section II.B will consider the historical information from the other previous versions when evaluating the priority of the test cases of the current version. The empirical results in [7], [8], [10] show that the historical information collected from all of the previous versions should be useful. Given that both historical information and fault localization information are effective for test case prioritization, we plan to incorporate fault localization into the history-based prioritization technique, and then propose a hybrid technique to further improve the effectiveness of test case prioritization. Section III.B of this paper will describe our preliminary experimental results and then propose the underlying assumptions for constructing the hybrid test case prioritization technique.

B. Preliminary experimental analyses and the underlying assumptions for the development of the hybrid technique

Table I furnishes descriptive statistics that we collected from the Software-artifact Infrastructure Repository (SIR) [14]. Please notice that the SIR programs and test suites are frequently chosen benchmarks for evaluating test suite reduction methods. In order to come up with reasonable and valid assumptions for the new hybrid technique, we conducted an investigation into the historical information and fault localization information from the SIR subject programs and the associated test suites. The results indicate two observations that are shown in Tables II and III, respectively.

TABLE I. DETAILS OF THE SIR SUBJECT PROGRAMS AND THE ASSOCIATED TEST SUITES

Subject Programs	Num. of test cases	Num. of test requirements	Num. of software versions
printtokens	4,130	140	7
printtokens2	4,115	138	10
replace	5,542	126	32
schedule	2,650	46	9
schedule2	2,710	72	10
tcas	1,608	16	41
totinfo	1,052	44	23
Space*	13,585	1,067	39

* Except for space, the other programs belong to the Siemens programs.

Table II shows the influence caused by the test results of the immediately preceding software versions. That is, we

compared the probabilities that a code element is associated with faults in the current software version between the case that the code element was executed by a failed test case in the immediately preceding version and the case that the code element was executed by a passed test case in the immediately preceding version. According to the table, the probabilities for the former case are higher than those of the latter for most of the subject programs.

TABLE II. INFLUENCE CAUSED BY THE TEST RESULTS OF THE IMMEDIATELY PRECEDING VERSIONS

Subject Programs	If a *failed* test case executed a code element the immediately preceding version[a]	If a *passed* test case executed a code element the immediately preceding version[b]
	Prob. that this code element is associated with faults in the current version	
printtokens	12.11%	3.20%
printtokens2	11.17%	8.05%
replace	5.68%	2.98%
schedule	6.73%	4.41%
schedule2	1.70%	2.13%
tcas	5.61%	3.21%
totinfo	19.14%	10.25%
space	18.93%	15.14%
Average	10.13%	6.17%

[a] Passed test case: a test case that does not reveal faults.
[b] Failed test case: a test case that reveals faults.

Table III shows the results of the fault-prone analyses for the code elements covered by failed and passed test cases, respectively. That is, we compared the probabilities that the code elements involve faults between the case that the code elements are covered by a failed test case and the case that the code elements are covered by a passed test case. The results indicate that the code elements covered by failed test cases are indeed more fault-prone that those covered by passed test cases.

TABLE III. FAULT-PRONE ANALYSES FOR THE CODE ELEMENTS COVERED BY FAILED AND PASSED TEST CASES, RESPECTIVELY.

Subject Programs	The probability that code elements covered by a *failed* test case involve faults[1]	The probability that code elements covered by a *passed* test case involve faults[2]
printtokens	73.10%	61.19%
printtokens2	68.71%	59.28%
replace	60.40%	52.31%
schedule	82.50%	73.31%
schedule2	85.76%	82.16%
tcas	83.82%	49.50%
totinfo	65.55%	57.51%
space	24.81%	25.71%
Average	68.08%	57.62%

[1] For all of the code elements covered by a specific failed test case, we sum up the probabilities that the code element involves faults.
[2] For all of the code elements covered by a specific passed test case, we sum up the probabilities that the code element involves faults.

Based on the aforementioned experimental results, we have the following assumptions for the hybrid technique to be developed.

(1) If a code element was executed by a failed test case in the immediately preceding software version, then there is a higher probability that the code element will be associated with a fault again in the successive version; conversely, if the code element was executed by a passed test case in the immediately preceding software version, then there is a lower probability that it will be associated with a fault in the successive version.

(2) If the code elements covered by a test case are more fault-prone, then there is a higher probability that this test case can reveal faults.

These two assumptions are extracted from the historical data and the fault localization data of the SIR programs and the associated test suites However, the Siemens programs are of a small size while the space program is of a medium size. We cannot definitively ensure that our observations will stand for other programs. Thus, before constructing the hybrid technique based on these two assumptions, we will reduce this threat by conducting additional experiments with larger subject programs and test suites.

IV. CONCLUSION AND FUTURE WORK

Due to the effectiveness of two kinds of test case prioritization techniques (i.e., the history-based techniques and the techniques that consider fault localization information), we want to propose a hybrid test case prioritization technique to further improve the capability of detecting faults early in the process of regression testing. Before proposing the hybrid technique, we conducted an investigation into several subject programs and the associated test suites and obtained two preliminary observations. Based on the two observations, we also proposed two assumptions for the hybrid technique to be developed. This paper presents the aforementioned preliminary results. In the near future, we plan to carefully examine the effectiveness of the two assumptions. If they are generally reasonable and valid for most of subject programs and test suites, we will develop the hybrid technique accordingly.

ACKNOWLEDGMENT

This work was supported by the National Science Council, Taiwan, under Grants NSC 102-2221-E-415-009.

REFERENCES

[1] D. Binkley, "Semantics Guided Regression Test Cost Reduction," *IEEE Trans. on Software Engineering*, Vol. 23, No. 8, pp. 498-516, August 1997.

[2] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Prioritizing Test Cases for Regression Testing," *Proceedings of the ACM/IEEE International Symposium on Software Testing and Analysis* (ISSTA 2000), pp. 102-112, August 2000.

[3] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Prioritizing Test Cases for Regression Testing," *IEEE Trans. on Software Engineering*, Vol. 27, No. 10, pp. 929-948, October 2001.

[4] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," *IEEE Trans. on Software Engineering*, Vol. 28, No. 22, pp. 159-182, February 2002.

[5] A. G. Malishevsky, J. R. Ruthruff, G. Rothermel and S. Elbaum, "Cost-cognizant Test Case Prioritization," Technical Report TR-UNL-CSE -2006-0004, March 2006.

[6] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Test Case Prioritization: An Empirical Study," *Proceedings of the 15th IEEE International Conference on Software Maintenance* (ICSM 1999), pp. 179-188, September 1999.

[7] J. M. Kim and A. Porter, "A History-based Test Prioritization Technique for Regression Testing in Resource Constrained Environments," *Proceedings of the 24th ACM/IEEE International Conference on Software Engineering* (ICSE 2002), pp. 119-129, May 2002.

[8] W. N. Liu, C. Y. Huang, C. T. Lin and P. S. Wang, "An Evaluation of Applying Testing Coverage Information to Historical-value-based Approach for Test Case Prioritization," *Proceedings of the 3rd Asia-Pacific Symposium on Internetware* (Internetware 2011), pp. 73-81, December 2011.

[9] B. Jiang, Z. Zhang, W. K. Chan, T. H. Tse and T. Y. Chen, "How Well Does Test Case Prioritization Integrate with Statistical Fault Localization?" *Information and Software Technology*, Vol. 54, No. 7, pp. 739-758, July 2012

[10] C. T. Lin, C. D. Chen, C. S. Tsai, and G. Kapfhammer, "History-based Test Case Prioritization with Software Version Awareness," *Proceedings of the 18th International Conference on Engineering of Complex Computer Systems* (ICECCS 2013), pp. 171-172, July 2013.

[11] S. Kim and J. Baik "An Effective Fault Aware Test Case Prioritization by Incorporating A Fault Localization Technique," *Proceedings of the 2010 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM 2010), Article No. 5, September, 2010.

[12] J. A. Jones and M. J. Harrold. "Empirical Evaluation of the Tarantula Automatic Fault-localization Technique," *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering* (ASE 2005), pp. 273-282, November 2005.

[13] Y. Yu, J. A. Jones and M. J. Harrold, "An Empirical Study of the Effects of Test-suite Reduction on Fault Localization," *Proceedings of the 30th ACM/IEEE International Conference on Software Engineering* (ICSE 2008), pp. 201-210, May 2008.

[14] "Software-artifact Infrastructure Repository (SIR)," Site: http://sir.unl.edu/portal, Available Date: 2014/03/08.