History-based Test Case Prioritization with Software Version Awareness

Chu-Ti Lin[1], Cheng-Ding Chen[2], Chang-Shi Tsai[1], Gregory M. Kapfhammer[3]

[1]Dept. of Computer Sci. and Info. Eng.
National Chiayi University
Chiayi, Taiwan

[2]Cloud Computing Center for Mobile Applications
Industrial Technology Research Institute
Hsinchu, Taiwan

[3]Dept. of Computer Science
Allegheny College
Meadville, PA, USA

***Abstract*—Test case prioritization techniques schedule the test cases in an order based on some specific criteria so that the tests with better fault detection capability are executed at an early position in the regression test suite. Many existing test case prioritization approaches are code-based, in which the testing of each software version is considered as an independent process. Actually, the test results of the preceding software versions may be useful for scheduling the test cases of the later software versions. Some researchers have proposed history-based approaches to address this issue, but they assumed that the immediately preceding test result provides the same reference value for prioritizing the test cases of the successive software version across the entire lifetime of the software development process. Thus, this paper describes ongoing research that studies whether the reference value of the immediately preceding test results is version-aware and proposes a test case prioritization approach based on our observations. The experimental results indicate that, in comparison to existing approaches, the presented one can schedule test cases more effectively.**

Keywords—Regression Testing, Test Case Prioritization

I. INTRODUCTION

It is inevitable that the functionality of a software system may change during software development and maintenance. Each time the software is modified, regression testing is necessary to ensure the quality of the software system. That is, new test cases will be designed 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. Thus, the test suite gradually grows in size as the software evolves and regression testing also becomes more and more expensive. It is also difficult to execute the entire test suite in a short time. Moreover, the fault detection capabilities of different test cases may not be the same. If the test cases with higher fault detection capability can be executed early in the process of regression testing, a high percentage of faults may still be detected even though the allocated time is not enough to finish the entire regression test suite. As a result, several test case prioritization approaches have been proposed to increase the effectiveness of regression testing [1]-[5]. According to [1], if T is a test suite, PT is the set of all possible permutations of the test cases in T, and f is a function from PT to the real numbers, the test case prioritization problem is to find $T' \in PT$ such that $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.

This research was sponsored by the National Science Council of Taiwan (Grant: NSC 100-2221-E-415-007-MY2).

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 [2], [4]. Even though experimental results have shown that the code-based approaches can improve the fault detection rate of the test suite, most of them are memoryless in that they model regression testing as a one-time activity rather than a continuous process (i.e., they ignore the reference value of preceding tests across multiple software releases) [3]. Therefore, Kim and Porter [3] proposed a history-based test case prioritization approach based on the historical fault information. Their experimental results suggested that historical fault information is valuable for improving the effectiveness of the regression testing process in the long term. Recently, Liu et al. [5] argued that, in addition to historical fault information, the information collected from the source code is also important for test case prioritization. Thus, they suggested prioritizing test cases based on information concerning both historical faults and the source code.

However, both [3] and [5] assumed that the test result of each immediately preceding software version has the same importance for the test case prioritization of its successive version across all versions. This leads us to consider an open research question: is the reference value of the test result of the immediately preceding version of the software version-aware for the successive test case prioritization? This paper will explore this issue, propose a test case prioritization approach based on our observations, and give an empirical evaluation.

II. VERSION-AWARE APPROACH

Table I furnishes descriptive statistics that we collected about the Siemens programs from the Software-artifact Infrastructure Repository (SIR) [6], benchmarks that are frequently used to compare different test case prioritization methods. After analyzing the test results of all versions of the Siemens programs, we found that, for the test cases detecting faults in a specific version, there is a higher probability that they will detect faults again in the successive version. This confirms that the historical fault information deserves to be considered when prioritizing the tests during future regression testing, which is not in conflict with the arguments in [3] and [5]. Moreover, as the programs evolve, software developers are usually more familiar with the programs and their debugging skills should gradually improve. As a result, the phenomenon that a test case detects faults in two successive versions may

get less and less significant. This conjecture conflicts with the assumption from [3] and [5] mentioned in Section I. The Siemens programs – replace, tcas, and totinfo – have at least 23 versions. Since this number of versions is much greater than the number for other programs in SIR, we analyzed their first 23 versions to validate this conjecture. Fig. 1 shows the probability that a test case finds faults in a specific version if it detected faults in the immediately preceding version, as the programs evolve from the 2nd to the 23rd versions. The linear regression plot indicates that the probability tends to decrease as the programs evolve, thus supporting our conjecture.

Recall that Liu et al. [5] argued that source code elements are also important for test case prioritization. Based on Liu et al.'s suggestion and our aforementioned observations, we assume: (i) both historical fault data and source code information are valuable for prioritizing test cases in the later software versions; (ii) the priorities of the test cases that detected faults in the immediately preceding version should be increased; (iii) the increment described in Assumption (ii) is software-version-aware and will linearly decrease as the programs evolve. Accordingly, we evaluate the version-aware priority of a specific test case in the *k*-th version by

$$P_k = \begin{cases} C_{num}, \text{if } k = 0, \\ P_{k-1} + h_k \times C_{num} \times [(Vers - k)/Vers], \text{if } k > 0, \end{cases} \quad (1)$$

where *Vers* is the number of versions of the subject program, C_{num} is the number of branches covered by the test case, and h_k is the historical fault information (it takes 1 if the test case detected a fault in the (*k*–1)-th version; otherwise, it takes 0). For the initial version, the presented approach gives the test case priority based on code coverage only; for other versions, the priority inherits that of the immediately preceding version and is adjusted according to the test result of the immediately preceding version, the code coverage of test case, and the version number of the program.

III. Prelimnary Experimental Analyses

In the experiments, we compared the presented approach with Kim and Porter's, Liu et al.'s, and the randomly created prioritizations (i.e., test cases that are randomly reordered) for the SIR Siemens programs [6]. Additionally, we used the cost-cognizant average percentage of fault detected ($APFD_C$) [7] to evaluate the fault detection rate of each prioritized test suite since it is a commonly-adopted criterion. An $APFD_C$ value ranges between 0 and 100%, with a higher value indicating a better fault detection rate. Table 2 shows the $APFD_C$ value of the four approaches compared in this study. The results indicate that the presented method provides better $APFD_C$ values than the random approach for all of the Siemens programs. It also outperforms Kim and Porter's and Liu et al.'s techniques for most of the Siemens programs. Considering the averages of the $APFD_C$ values across all of the Siemens programs, the presented approach still provides the best fault detection rates, thus demonstrating its potential.

IV. Conclusion and Futrue Work

Using the Siemens programs as experimental subjects, this paper invalidated an unsuitable assumption from most of the existing history-based approaches and presented a software-version-aware approach that considers both source code information and historical fault data. The experimental results indicate that the presented approach provides a better fault detection rate, in terms of $APFD_C$, for the Siemens programs. In future work, we intend to replace (1) by a full-featured model to adjust the software-version-aware test case priority more accurately. We will also conduct more experiments with case study applications that have more source code and tests.

TABLE I. Descriptions and Statistics of SIR Siemens Programs

Subject Programs	Num. of Versions	If a test case failed in a specific version	If a test case passed in a specific version
		Prob. that it fails in the next version	
printtokens	7	6.78%	2.05%
printtokens2	10	22.25%	3.95%
replace	32	7.39%	1.78%
schedule	9	3.79%	1.68%
schedule2	10	7.55%	0.81%
tcas	41	5.61%	2.78%
totinfo	23	21.30%	5.96%

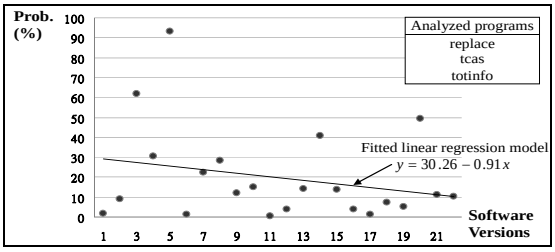

Fig. 1. The probability that a test case detects faults in two successive software versions as the programs evolve.

TABLE II. $APFD_C$ Values for the Compared Approaches

Programs	Kim & Porter's	Liu et al.'s	Random	Presented
printtokens	54.86%	70.12%	49.52%	70.11%
printtokens2	79.25%	72.65%	50.68%	81.95%
replace	72.62%	68.18%	49.42%	76.33%
schedule	67.41%	56.13%	49.94%	63.27%
schedule2	58.25%	51.05%	48.70%	60.27%
tcas	66.52%	60.31%	50.23%	74.13%
totinfo	69.83%	72.32%	48.96%	74.46%
Average	66.96%	64.39%	49.64%	71.50%

References

[1] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Prioritizing test cases for regression testing," Proc. ACM SIGSOFT Symp. Software Testing and Analysis, Auguest 2000, pp. 102-112.

[2] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," IEEE Trans. on Software Engineering, vol. 27, October 2001, pp. 929-948.

[3] J. M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," Proc. ACM/IEEE Conf. Software Engineering, May 2002, pp. 119-129.

[4] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," IEEE Trans. on Software Engineering, vol. 28, February 2002, pp. 159-182.

[5] 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," Proc. Asia-Pacific Symp. Internetware, December 2011, pp. 73-81.

[6] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria," Proc. ACM/IEEE Conf. Software Engineering, May 1994, pp. 191-200.

[7] 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.