2016 23rd Asia-Pacific Software Engineering Conference

History-based Test Case Prioritization for Failure Information

Younghwan Cho[*,†], Jeongho Kim[*], Eunseok Lee[*]

[*]Dept. of Information and Communication Engineering
Sungkyunkwan University
Republic of Korea
{yhwan.cho, jeonghodot, leees}@skku.edu

[†]Mobile Communication & Business
Samsung Electronics
Republic of Korea
yhwan.cho@samsung.com

***Abstract*— From regression tests, developers seek to determine not only the existence of faults, but also failure information such as what test cases failed. Failure information can assist in identifying suspicious modules or functions in order to fix the detected faults. In continuous integration environments, this can also help managers of the source code repository address unexpected situations caused by regression faults. We introduce an approach, referred to as AFSAC, which is a test case prioritization technique based on history data, that can be used to effectively obtain failure information. Our approach is composed of two stages. First, we statistically analyze the failure history for each test case to order the test cases. Next, we reorder the test cases utilizing the correlation data of test cases acquired by previous test results. We performed an empirical study on two open-source Apache software projects (i.e., Tomcat and Camel) to evaluate our approach. The results of the empirical study show that our approach provides failure information to testers and developers more effectively than other prioritization techniques, and each prioritizing method of our approach improves the ability to obtain failure information.**

Keywords- regression test; test case prioritization; history data; failure information; continuous integration environments

I. Introduction

In continuous integration environments, Developers continuously merge commits to the source code repository, and managers or automated systems control critical situations (e.g., breaking builds or damaged crucial functionalities) as fast as they can. In critical situations, the source code repository must be frozen and all integration processes stop. A regression test essentially performs a whole test suite for each change. Regression tests are so costly that they need to be optimized in order to reduce the expense with respect to time and resources. Researchers have proposed various optimization techniques to solve this problem. Yoo and Harman's survey classified optimization techniques into three categories: selection, minimization, and prioritization [1]. A prioritization, which is studied in this paper, is used to determine the execution order of test cases based on certain criteria.

Developers need to know the existence of regression faults early. Furthermore, they practically need to acquire information about the faults from the regression tests. Failure information can help deal with the faults revealed by failed test cases. But existing prioritization techniques typically find the first test case to detect each fault and ignore the rest of the failed test cases, despite the fact that these could provide useful information. Existing prioritization techniques to improve the fault detection rate are based on various criteria such as coverage, distribution, model, and requirements [1]. History-based approaches are one of these prioritization techniques and utilize information from history data.

This paper presents an approach based on history data to effectively obtain failure information. Our approach is composed of two stages of prioritizing that use different types of analysis. In the first stage, we assign weights to each test case based upon statistical analysis of the failure history; we order test cases by these weighted priority values. In the second stage, when a test case fails during the regression test, we reorder the test cases based upon the correlation data. The correlation data are collected from the result status and failure history of previous consecutive test sessions.

The main contributions of this paper are as follows:

1. We propose a test case prioritization approach, referred as to AFSAC, which is based on history data. This is used to quickly obtain more failure information. AFSAC stands for prioritization by Accumulation of Failures with weights from Statistical Analysis and Correlation data.
2. We conduct empirical study with two open-source projects in continuous integration environments to evaluate the effectiveness of our approach. We show the practical results of our experiments including the prioritization approaches proposed by other researches.

The rest of this paper is organized as follows: In section II, we introduce the background and related work of our study. Section III describes our prioritization approach. Section IV presents the empirical study including details of the experiments, results, and analysis. In section V, we present our conclusions and future work.

II. Background and Related Work

A. History-based Prioritization Techniques

History-based prioritization techniques use history data derived from the records of previous tests. History data contain various types of information such as what test cases failed in which test session, what files are changed, and the amount of time each test case took to be executed. Collecting coverage data for coverage-based prioritization techniques is time-consuming. And as software evolves and increases in complexity, it becomes much more costly. Alternatively, history-based prioritization techniques do not use source code analysis; therefore, they require less time and effort to acquire the information needed. Kim and Porter utilized the execution history and fault detection history to select and prioritize test cases [2]. Marijan et al. [3] introduced ROCKET to prioritize test cases for continuous regression testing using the failure history and execution time of the test cases. Elbaum et al. [4] introduced the selection and prioritization approach to utilize the history data of failure and execution and the execution time.

B. *Continuous Integration Environments*

In continuous integration environments, if the detected faults are severe, they are fixed immediately with frozen source code repository. To investigate the causes and determine the criticality of a situation, the failure information can be useful. For regression tests in continuous integration environments, researchers have proposed optimization techniques in attempts to increase the cost efficiency. Elbaum et al. [4] mentioned that their approach could report failures more quickly through the use of test case prioritization; however, they did not consider the amount of failure information could be provided to support later works. In this paper, we focus on acquiring failure information.

C. *Correlation of Test Cases*

Recently, many software products have become increasingly complex and consist of a large number of modules. Thus, it can be difficult to predict the ramifications of one small incorrect change. Developers usually depend on their practical experience and knowledge when they run test cases. Ekelund et al. [5] proposed an approach to find test cases correlated to the changed package for efficient regression testing. The correlation data can include both the visible correlation from code analysis and the invisible correlation due to the complexities of the software. This implies that the correlation data can be used instead of having to rely on the practical experience of the developers and code analysis.

III. Approach

In this section, we describe our approach, AFSAC, to effectively obtain failure information, especially for continuous integration environments.

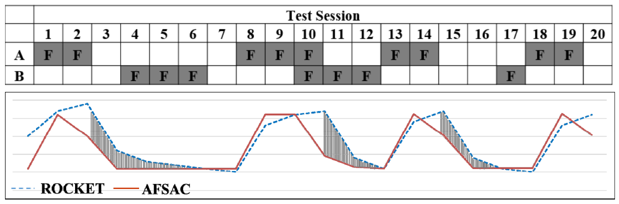
A. *Overview*

The approach proposed in this paper is composed of two stages that prioritize the test cases. In the first stage, the prioritizing method applies weights, which are determined by statistical analysis. This determines the execution order of test cases for early fault detection. The basis of ordering in this stage is that more error-prone test cases should be located in the fore positions of the execution order. The second prioritizing method reorders the test cases to obtain more failure information by the correlation data derived from the results of previous tests. If a test case fails during a regression test, the remaining test cases that have yet to be executed are immediately reordered. The reordering is completed by assigning high priority to test cases correlated to the failed test cases. We describe the details of each prioritizing method in the following subsections.

B. *Prioritizing by Statistical Analysis of Failure History*

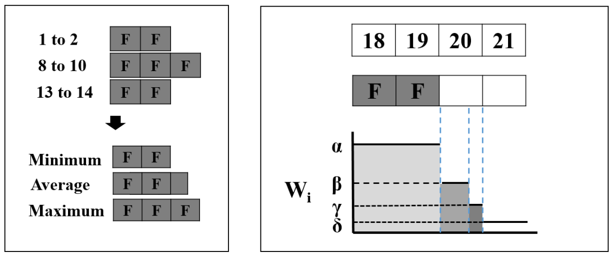
We adopt the ROCKET approach of [3] as a prioritizing algorithm except the part considers the execution time. We conjecture that the probability or pattern of reoccurring failures is represented by the characteristics of each test case and project. Thus, we develop statistical analysis method for failure history that can reflect the characteristics of the target project and each test case. We use the analyzed data to determine the weight values for prioritization. The weighting method is designed to alleviate the drawback that can occur with ROCKET. Fig. 1(a) shows how the priorities of ROCKET and AFSAC are changed in each test session. The priority of ROCKET is still large after the results of test cases change from "failure" to "pass" in test sessions 3, 11, and 15. Additionally, the values become significantly smaller if the test case remains as "pass". Thus, the other test cases can lose their opportunity to be executed early. In this aspect, using a fixed range to assign large weights can hinder the prioritization performance. Thus, we do not assign fixed large weights to the test cases but instead assign weights by utilizing the statistical data. This causes the priority to decrease smoothly in advance and alleviates the aforementioned drawback.

The statistically analyzed data include minimum, average, and maximum range values. The weights are determined according to the position of the target test session. Fig. 1(b) and (1) show the criteria used to determine the weights, W_k. From the analyzed data, we know the minimum, average, and maximum ranges of failures, which are denoted as Fr in (1). The result values of the test case, Tr, of test session i are set by using (2). Finally, we calculate the priority values of test session i, Pv_i, using (3) for each test case. In (3), n denotes latest passed test session and m denotes target test



	Test Session																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	F	F						F	F	F			F	F				F	F	
B				F	F	F				F	F	F					F			

(a) Changing trends in the priority of test case A



Failure statistics of test case A   Weight by failure statistics

(b) Weighting by the failure statistics

Figure 1. Statistical analysis of failure history

session. After calculating the priority values for all test cases, we prioritize the test cases in ascending order of priority value. Note that a test case with a lower priority value has higher priority.

$$W_k = \begin{cases} \alpha, & \text{if } k < Fr_{\min} \\ \beta, & \text{if } Fr_{\min} \le k < Fr_{avg} \\ \gamma, & \text{if } Fr_{avg} \le k < Fr_{\max} \\ \delta, & \text{if } Fr_{\max} \le k \end{cases} \quad (1)$$
$$(\alpha > \beta > \gamma > \delta > 0)$$

$$Tr_i = \begin{cases} -1, & \text{if failed in test session } i \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

$$Pv_i = \sum_{i=1}^{n} Tr_i \times \omega + \sum_{j=n+1}^{m-1} Tr_j \times W_{m-j} \quad (3)$$

C. Prioritizing by Analyzing the Correlation Data

Analysis of the correlation among test cases is based on flipping the results of the test cases by one commit. A flipped test case means that the result of a test case is changed to the opposite of result status of the previous test session. In the research by Ekelund et al. [5], flipping is described as changing the verdict between two consecutive test runs. Generally, in continuous integration environments, commits are forced to be written atomically. Managers of source code repositories and code reviewers recommend that commits not be combined with unrelated and inconsistent changes. Thus, we can say that two test cases are correlated when their results are changed to the opposite status by one commit in two consecutive test sessions. After every regression test is terminated, we form a correlation matrix in order to use the correlation data derived from the result of previous tests for the next test session. The correlation matrix is composed of values reflecting the accumulated number of flipped results.

IV. Empirical Study

In this section, we describe the details of our experiments and present the experimental results. We discuss the effectiveness of AFSAC by answering the following research questions:

- RQ1: Can AFSAC improve the ability of regression tests to obtain failure information? Is AFSAC more effective than other techniques?
- RQ2: Can the two prioritizing methods improve the ability to obtain failure information respectively?

A. Experiment Environment

We use two open-source Apache software projects (i.e., Tomcat and Camel) in our experiments. Using these projects, which includes real faults, helps practical evaluation.

Every commit integrated into the master branch of each source code repository from January 1, 2015 to December 31, 2015 is included in the target versions for both projects. We compared AFSAC to several existing prioritizing techniques that use history-based approaches in terms of obtaining failure information. We used different sizes of history data for the prioritizing technique described by [4], as denoted by FW-n. Due to space limitations, we use abbreviations to denote each technique. Untreated order of test cases is denoted by Unt. And random order and ROCKET [3] are denoted by Rnd and RKT. We also divided each prioritizing method of AFSAC into AF, SA, and C for evaluation. AF denotes the using calculation of accumulated failures, which means calculating *Pv* without W_k in (3). SA denotes using the weighting method with the data obtained from the statistical analysis. C denotes using the prioritizing method to reorder with the correlation data.

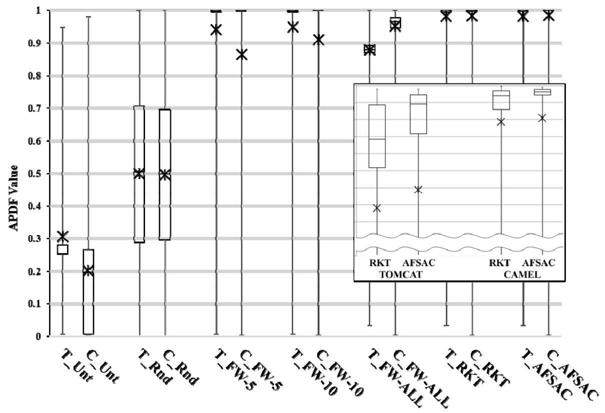
$$APDF = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{nm} + \frac{1}{2n} \quad (4)$$

We adopt the APFD defined by Rothermel et al. [6] to measure the effectiveness of our approach, which aims to obtain failure information more quickly. In (4), the test suite has *n* test cases and detects *m* number of faults. And TF_i is rank of the first test case which detect fault *i*. We use the same equation, only we change the notation of TF_i to the rank of the failed test case *i*. We refer to this modified metric as the average percentage of detected failures (APDF). If every failed test case reveals all different faults, the APFD value is same as the APDF value.

B. Results and Analysis

We discuss the results by answering the two research questions mentioned earlier.

1) RQ1. Effectiveness of AFSAC and Comparing the results of the different techniques: Fig. 2(a) compares the results of the prioritization techniques with respect to their effectiveness. AFSAC performs better than the other techniques in terms of the mean, minimum and each propotional indicator (with the exception of the maximum), including no prioritization. The maximum values of APDF are the same in several cases. This was caused by many cases that continuously have few failures with simple patterns. In these cases, prioritization techniques that consider the failure history are sufficient for maximizing the effectiveness. Furthermore, the differences in the APDF values between the techniques are not significantly different. The two projects have so many test cases that the possible number of permutations is very large. Additionally, we also normalized the value to [0,1]. For these reasons, we present the results in another way by showing the pair-wise comparison (except for cases that have equivalent execution orders). The box in Fig. 2(a) shows the results of the pair-wise comparison between RKT and AFSAC, indicating that AFSAC is more effective than RKT.


(a) APDF boxplots for Tomcat (T) and Camel (C)

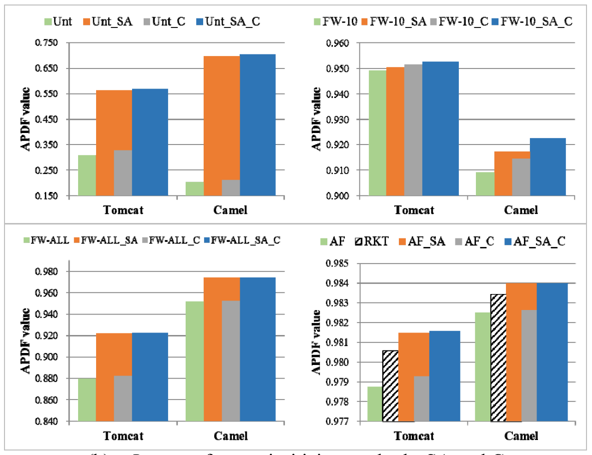

(b) Impact of two prioritizing methods, SA and C

Figure 2. Effects of AFSAC

2) *RQ2. Effectiveness of each prioritizing method:* We conducted experiments in order to investigate the impact of our two prioritizing methods. We applied the prioritizing method based on the SA and C to the other prioritization techniques; Unt, FW-n, and AF. Fig. 2(b) shows the results of the techniques that utilized SA and C respectively. The experimental results show that the four techniques with the applied SA and C are improved with respect to APDF in both subjects. The SA is proposed to alleviate the drawback of RKT. The last graph in Fig. 2(b) shows the effect of applying SA to AF and RKT. The APDF value of AF is lower than that of ROCKET, but the value of AF_SA (which applies SA on AF) is higher. This implies that the statistical data are useful and beneficial for obtaining failure information, and the failure history of each test case is patterned. However, it does not prove that the patterned failure history reflect the characteristics of test cases. We will investigate the correlation between the history data and the characteristics of test cases in our future work.

C. *Threats to Validity*

We used two open-source projects as subjects to evaluate our approach. Although the number of subject projects is small, they are large-scale and famous open-source projects. The size of the test suites and the number of target versions are very large. Hence, the experimental results of our empirical study are sufficient to confirm the effectiveness of our proposed approach.

V. CONCLUSION

We propose an approach (i.e., AFSAC) that is a test case prioritization technique based on history data to effectively provide failure information to testers or developers in continuous integration environments. Our approach uses two prioritizing methods; each method utilizes the statistical data of failures and the correlation data. We conduct an empirical study to evaluate the effectiveness of AFSAC with two open-source projects (i.e., Tomcat and Camel) and we compare our approach to existing prioritization techniques. The AFSAC outperforms other prioritizing techniques in terms of the APDF. The results of our experiments also show that the two prioritizing methods of our approach improve the ability to obtain failure information. In our future work, we will conduct additional experiments to evaluate our approach with other projects and compare our results with the techniques of other researchers.

ACKNOWLEDGMENT

This research was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(No. 2015045358), and the MISP(Ministry of Science, ICT & Future Planning), Korea, under the National Program for Excellence in SW)(R2215-16-1005) supervised by the IITP(Institute for Information & communications Technology Promotion).

REFERENCES

[1] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability, 22(2), 67-120.

[2] Kim, J. M., & Porter, A. (2002, May). A history-based test prioritization technique for regression testing in resource constrained environments. In Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on (pp. 119-129). IEEE.

[3] Marijan, D., Gotlieb, A., & Sen, S. (2013, September). Test case prioritization for continuous regression testing: An industrial case study. In Software Maintenance (ICSM), 2013 29th IEEE International Conference on (pp. 540-543). IEEE.

[4] Elbaum, S., Rothermel, G., & Penix, J. (2014, November). Techniques for improving regression testing in continuous integration development environments. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 235-245). ACM.

[5] Ekelund, E. D., & Engström, E. (2015, September). Efficient regression testing based on test history: An industrial evaluation. In Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on (pp. 449-457). IEEE.

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