2015 IEEE International Conference on Software Quality, Reliability and Security

Multi-Perspective Regression Test Prioritization for Time-constrained Environments

Dusica Marijan
Certus Software V&V Centre
Simula Research Laboratory
Fornebu, Norway
Email: dusica@simula.no

***Abstract*—Test case prioritization techniques are widely used to enable reaching certain performance goals during regression testing faster. A commonly used goal is high fault detection rate, where test cases are ordered in a way that enables detecting faults faster. However, for optimal regression testing, there is a need to take into account multiple performance indicators, as considered by different project stakeholders. In this paper, we introduce a new optimal multi-perspective approach for regression test case prioritization. The approach is designed to optimize regression testing for faster fault detection integrating three different perspectives: business perspective, performance perspective, and technical perspective. The approach has been validated in regression testing of industrial mobile device systems developed in continuous integration. The results show that our proposed framework efficiently prioritizes test cases for faster and more efficient regression fault detection, maximizing the number of executed test cases with high failure frequency, high failure impact, and cross-functional coverage, compared to manual practice.**

***Keywords*—software testing; regression testing; test case prioritization**

I. Introduction

Regression testing is a widely used technique for uncovering regression faults in software that has undergone changes. The goal is to use the appropriate minimum set of test cases that cover a particular change, in order to check if previously fixed faults have re-emerged. The common problems relate to selecting the appropriate minimum set of test cases. Generally, these problems are solved by test prioritization. However, the efficiency of regression test prioritization changes with different test performance indicators, as defined by different project stakeholders. For example, in time-constrained development environments, which are common for agile projects, teams work in short development cycles, continuously integrating code to the mainline and testing the code every time a new change has been proposed to the codebase. In such environments, an important regression-test performance indicator is the agility of the overall process. To enable efficient test prioritization for continuous integration development environment, a key aspect is a short feedback loop from code commits to test execution reports. This entails a trade-off between (a) selecting test cases that have high fault detection ability and (b) maximizing the number of test cases that can be executed in available time. Similarly, seen from a business perspective, the key aspect in regression testing is ensuring that test cases detect failures that would impact customers the most, and thus would have negative consequences for the business.

To accommodate multiple perspectives on regression test performance, in this paper, we propose a framework for optimal multi-perspective regression test case prioritization for time-constrained environments. The framework optimizes regression testing for faster fault detection integrating three different perspectives: business perspective, performance perspective, and test design perspective. As one of the advantages, the proposed framework does not require source code. It uses the following five factors extracted from different test artifacts: (i) test suite, (ii) test failure frequency and severity (iii) test execution time, (iv) total testing budget (time), and (v) test coverage. The prioritization algorithm computes a weight for each test case based on the distance of its failures from the current execution, its failure frequency and severity, coverage granularity, and total test time. The framework allows tuning the importance of factors, depending on the specifics of particular industrial environment.

We have validated the approach in regression testing of industrial mobile device software developed in continuous integration. The results show that our proposed framework efficiently prioritizes test cases for faster and more efficient regression fault detection, in limited time intervals, compared to manual practice. This means that in cases where time required to execute complete test suite exceeds available testing time, the multi-perspective framework will ensure that the most cost-efficient test cases are executed.

The paper is organized as follows. In section II, we provide background, give motivation for the work, and review related work on regression testing. In section III, we describe regression test prioritization for continuous integration and our proposed framework. In Section IV, we describe an empirical study of validating the framework in regression testing of industrial mobile device software, and present the results of the study. In section V, we draw conclusion and give directions of future work.

II. Background and Related Work

If we consider a program P, a modified version of P, P', and a test suite for P, T, then the regression testing aims to validate the correctness of P' by reusing T. However, using the whole set T is expensive and usually not necessary, and various approaches have been proposed for regression test selection, prioritization, and reduction. These approaches aim to define a test suite $T' \in T$ that will lead to improved cost-effectiveness of regression-testing, either by consuming less time, or by meeting test objectives faster. While regression test

978-1-4673-7989-2/15 $31.00 © 2015 IEEE
DOI 10.1109/QRS.2015.31

reduction discards redundant test cases (redundant with respect to some defined objective), regression test case prioritization is considered a safer technique that changes the order of test cases to achieve better test performance.

Test case prioritization has been extensively studied in literature. One class of the proposed prioritization techniques uses source code analysis, for example, applying information retrieval [1], or code-coverage information [2], [3]. Although sometimes considered more accurate, such techniques are in some cases inapplicable in practice. In some industry projects (and the case study reported in this paper), program source or binary code are not available to testers. Moreover, coverage-based prioritization techniques often assume equal severity of defects. Our goal is to account for multiple quality attributes with different severity, as considered by different users. Noguchi proposed a black-box approach for test prioritization [4], motivated by the fact that in software outsourced projects, testers rarely have access to source code. Still, his approach does not apply to time-constrained projects. Sherriff presented the approach to prioritize tests by analysing effects of changes through singular value decomposition [5]. Henard described a similarity-based solution for prioritizing sets of product configurations, specialized for large software product lines [6]. Jiang introduced adaptive-randomized test case prioritization that achieves better fault detection than code-coverage-based greedy or search-based prioritization techniques [7]. However, none of these approaches consider time constraints common for regression testing in continuous integration environment. Bryce presented a metric for cost-based prioritization [9], using combinatorial interaction coverage as objective. Srikanth proposed a technique that prioritizes tests based on the combination of fault detection rate over time and minimal test setup time, when switching configurations in execution [8]. This approach however does not consider time taken to execute tests.

Few approaches have analyzed time constraints in the context of test prioritization. Do presented a cost model for test prioritization and showed how different time constraints affect prioritization: time to setup testing, time to identify and repair obsolete tests, time to inspect results [10], [11]. Later, he conducted a series of experiments showing that time constraints can significantly affect the cost-effectiveness of prioritization [12]. Our work aims at prioritization with time constraints, but only in test execution. Walcott proposed a technique to reorder test suits in the presence of time constraints [13]. Still, this techniques considers that all tests have the same execution time. Zhang [14] and Alspaugh [15] proposed similar approaches, however, none focuses on continuous integration environment. Some research work has been done related to test prioritization in continuous environment [16] and [17], but none of these approaches consider multiple test prioritization perspectives.

III. Test Prioritization Framework

The goal of regression test case prioritization is to find the execution order for the given set of test cases that optimizes a given objective function. The objective function in our case consists of multiple perspectives: (1) selecting test cases with highest consecutive failure rate for the given number of executions, (2) maximizing the number of executed test cases while satisfying given testing time constraints, (3) selecting test cases with the highest failure impact, and (4) increasing overall test coverage.

A. Prioritization Factors

In or proposed framework, we consider four factors for test case prioritization, grouped in three perspectives.

1) Business Perspective: From a business perspective, we consider *Failure Impact* as an important factor affecting test effectiveness. It is a user-driven measure of the severity of defects of a test case. High-impact failures, when happen, degrade user experience and satisfaction. Therefore, the goal of test prioritization from a business perspective is finding such failures faster. *Failure Impact* factor is calculated based on historical user feedback reports collected from previous versions of system. Given a test case TC, its maximum failure severity, Max_Fi, and a sum of all n failure indexes that a test case has detected, F_k, a failure impact of the test case, $F_i(TC)$, is calculated as follows:

$$F_i(TC) = \frac{\sum_{k=1}^{n} F_k}{Max\_Fi}$$

2) Performance Perspective: From a performance perspective, *Test Execution Time* is an obvious determinant of test effectiveness. This factor is even more critical for time-constrained environments, such as continuous integration. In such environments, an objective of effective test prioritization is selecting test cases that execute quickly, in order to shorten the overall commit-verify feedback loop. Table I gives an example; there are five test cases (T1 to T5) shown in the table, with the distribution of regression faults in five consecutive test executions (E1 to E5). For the sake of simplicity, a test case is considered to reveal the same fault across executions. Each test case has a time unit (for example, test execution time). One possible order of test cases that maximizes the coverage of regression faults from previous executions is $Po1 = [T1, T4, T3, T5, T2]$. Consider the case where the upper bound for executing tests is 7 time units. In this case only $T1$ will be executed, checking against only one fault. Now, if we order test cases with respect to historical fault data and execution time, e.g., $Po2 = [T5, T3, T4, T2, T1]$, four test cases can execute in given 7 time units, $T5$, $T3$, $T2$ and $T4$, checking against four faults. If higher priority is given to shorter test cases with the same high failure impact, the prioritization improves the fault detection effectiveness of testing.

TABLE I. Test case prioritization for continuous regression testing.

Test case	E1	E2	E3	E4	E5	Exe time
T1	×	×			×	5
T2			×			2
T3		×			×	1
T4			×	×		3
T5	×	×				1

3) Technical Perspective: From a technical perspective, we consider two factors important for the effectiveness of regression testing. *Failure Frequency* is a measure of how often a test cases detect failures, and it is extracted from test

management system, or historical test execution reports, for a given time window. Given the hypothesis that high-frequency failing test cases often reoccur, the goal of effective regression testing is thus prioritizing such test cases. If we consider a test time window of n executions, and l the number of times a test TC has failed during n executions, we define a failure frequency of the test case, $F_f(TC)$, as:

$$F_f(TC) = \frac{l}{n}$$

Another factor considered important from a technical perspective is *Cross-functionality*, as a measure of how much of the functionality of the system under test is covered by a test case. For example, given a simple system A that consists of three components A_1, A_2, and A_3, under assumption that there are no constraints among the components, tests for A can be specified to cover A_1, A_2, and A_3 in isolation, or combining A_1 and A_2, or A_1 and A_3, or A_2 and A_3, or all three A_1, A_2, and A_3. Therefore, tests that cover wider system functionality should be prioritized for higher efficiency of testing, since this leads to better overall test coverage.

B. Approach

In this context, we define our objective function g as follows. For the given set of n test cases $S = \{S_1, S_2, ..., S_n\}$, failure status for each test case in S over the last m successive executions (if the test case has been executed before), $FS = \{\{fs_{1,1}, fs_{2,1}, ..., fs_{m,1}\}, \{fs_{1,2}, fs_{2,2}, ..., fs_{m,2}\}, ..., \{fs_{1,n}, fs_{2,n}, ..., fs_{m,n}\}\}$, first we calculate a cumulative failure frequency index for each test case $F = \{f_{S_1}, f_{S_2}, ..., f_{S_n}\}$. Next, given the F, the failure impact for each test case in S, $FI = \{fi_1, fi_2, ..., fi_n\}$, the test coverage for each test case in S, $TC = \{tc_1, tc_2, ..., tc_n\}$, and the test case execution time $Te = \{Te_1, Te_2, ..., Te_n\}$, we define the objective function as follows:

$$g = (maximize(f), maximize(fi), maximize(tc), minimize(Te))$$

where f is cumulative failure frequency of a test case, f_i is failure impact of a test case, t_c is a test coverage of a test case, and T_e is a test execution time of a test case. We define the problem of regression test case prioritization as finding the order of test cases from S, such that

$$(\forall S_i)(i = 1..n) g(S_i) \geq (g(S_{i+1})$$

One idea behind our objective function is that given a short test execution time during regression testing, an effective prioritization criterion should select test cases with highest failing frequency in the previous test executions, and having the highest impact for users. At the same time, prioritization becomes more effective when the selected set achieves higher test coverage. With respect to test failure frequency, the reasoning is as follows: the highest failure weight corresponds to the failure exposed in the $(current - 1)$ execution and the failure in every precedent execution is weighted lower than the failure in its successive execution. While re-executing the test cases that failed in precedent intermediate execution is mandatory in regression testing, very often faults detected in the second or third last execution and corrected reoccur later, due to "quick bug fixes" or masking effects that prevent fault detection.

Similarly, a successful precedent intermediate execution for a test case lowers its failure impact (i.e., priority). Given a short test execution time during regression testing, in addition to selecting test cases that showed high failure frequency, an effective prioritization criterion should select test cases (i) that execute quickly, to increase the number of executed test cases, as every test case can potentially detect faults. If a test case takes a lot of time to execute, even if it revealed failure in the precedent intermediate execution, executing this test case will prevent other failing test cases with the same failure impact (weight), but with shorter execution time, to execute; (ii) that revealed the most critical failures, and (iii) that are cross-functional, testing a larger part of software system.

IV. Evaluation

In this section, we present an experimental study to validate our proposed test prioritization framework. We have designed the experiments to answer the following research questions:

- RQ1: Does the framework outperform the manual prioritization practice with respect to test failure impact, failure frequency, and cross-functionality?
- RQ2: Can the framework increase the number of detected failures compared to manual practice of ordering test case, in case of partial test suite execution (with respect to time)?

A. Software of Analysis

We evaluated the framework using three software systems ranging from 2985 to 3327 lines of code developed for Android mobile OS. Each system has a test suite specified manually, consisting of from 501 to 700 test cases. Test suits vary in the level of feature interaction coverage from 1-wise to 7-wise feature interactions. Table II shows summary information about the used experiment subjects.

The three systems under study differ in terms of functionality, architecture, age, and the number of features. More complex functionality and architecture require more complex test suits, with higher level of test inter-dependency. Tests often cross-cut multiple functionality, which makes test prioritization more complex task. The age of the system is related to the number of failures, since mature software systems tend to have fewer bugs. Depending on a test suite design, the number of features can determine the size of a test suite. If most tests are specified to check features in isolation, then systems with large number of features will normally have large number of tests.

The systems are developed in continuous integration practice, where the developers' code copies are merged to the mainline several times a day. Continuous integration is supported by Jenkins/Hunsdon tools. On every code merge, an adequate set of tests is run to ensure a working build. The test team in charge of testing these three mobile systems consists of five test engineers who specify tests manually. The engineers are highly experienced in the domain, having from 10 to 15 years of experience. The main challenge for test engineers is using an adequate test suite that will maximize detecting failures and minimize the time taken to obtain test execution feedback.

TABLE II. EXPERIMENT SUBJECTS.

	A1	A2	A3
Size, LOC	2985	3250	3327
Test suite size	501	653	700
Min feature interaction coverage	1-wise	1-wise	2-wise
Max feature interaction coverage	7-wise	6-wise	7-wise

B. Methodology

The goal of the study is to evaluate the effectiveness of the proposed prioritization approach in detecting failures with higher failure impact, higher failure frequency, and tighter cross-functionality earlier, compared to manual test ordering. The reason that we compare with manual prioritization is because the domain-specialized testing framework employed in the projects under study requires tailored test prioritization solution, and we could not find any automated fitting approach readily available.

We conducted the experimental study as follows. First, we collected test data during five consecutive test case executions ($E1$ to $E5$). The data include: a test-suite, failing test cases for each execution, test failure impact, test coverage data and execution time for each test case. The data was collected from a test management system, and informal test specification documents. Second, we analyzed the data and built a matrix that shows how are failing test cases and their properties, in terms of failure impact and test coverage, distributed over consecutive regression test executions. For example, the test case T1 from Table I failed in test execution $E1$ and $E2$. Afterwards, the fault was fixed and T1 passed in execution $E3$ and $E4$, but due to regressions T1 failed again in execution $E5$. Based on the failure information and test coverage data from the previous five test executions, we ordered test cases using the proposed framework. Third, the team of five test experts were asked to manually order existing test cases for the three software systems under study. In the first phase, they worked individually, to avoid an anchoring effect during decision making. In the second phase, they worked in a team towards a common opinion about test case order. Finally, we executed test sets ordered manually and automatically, and compared the results. To answer RQ1, we measured and compared failure impact, failure frequency, and cross-functionality of these two test sets. To answer RQ2, we compared the number of failures detected by these two test sets, in time slices.

Furthermore, we compared prioritization effectiveness of our framework with manual prioritization using the Cost-cognizant weighted Average Percentage of Faults Detected measure (APFDC) [18]. APFDC rewards test case orders proportionally to their rate of "units-of-fault-severity-detected-per-unit-test-cost". We calculated APFDC and compared values for these two prioritization approaches.

C. Results

For the two prioritization approaches, we analyzed the following aspects: (i) how many test cases will execute in a limited testing time, (ii) what is the cumulative failure impact for all test cases executed in time increments of 20%, (iii) how many of high-frequency test cases will execute in a limited test time interval, and (iv) how many cross-functional tests will execute in a limited test time interval. The results are presented in Figure 1. Graphs in the figure show for each of these aspects, for each experiment subject, how automatic and manual prioritization techniques perform. The graphs in the same column represent comparison results for one experiment subject, A1, A2, and A3. Each row corresponds to one of the four factors: the number of failures detected, the cumulative failure impact, failure frequency, and test cross-functional coverage, respectively. Each graph shows values at time increments of 20% of the total test budget.

Let TA_A1, TA_A2, and TA_A3 be automatically prioritized test suits, and TM_A1, TM_A2, and TM_A3 be manually ordered test suits for A1, A2, and A3 subjects. For all three experiment subjects, after 20% of all test suites has been executed, the proposed approach outperformed the manual approach in terms of all four analyzed aspects. After 40% of test suites has been executed, the proposed approach outperformed the manual approach for all subjects, except for A1, where TM_A1 revealed the same number of faults as TA_A1. With 60% of test suites executed for A1 and A3, the proposed approach outperformed the manual approach for all aspects, except for failure frequency, where the two approaches performed equally. With 60% of test suites executed for A2, the proposed approach outperformed the manual approach for all aspects except for failure impact, where the two approaches gave the same result. After 80% of test suites has been executed for all subjects, performance of the two approaches started to be resemble. Finally, after complete test suits were executed, the two prioritization approaches gave the same results, since both approaches order tests suites without discarding any test cases.

In addition, we used APFDC measure to compare the effectiveness of test prioritization of the two approaches. In particular, $APFDC_{MULTI} = 19.59$, $APFDC_{manual} = 16.17$. These results show that our proposed prioritization framework has higher rate of regression fault detection per unit of test case cost.

In summary, the results of the case study show that multi-perspective framework efficiently prioritizes test cases for faster and more efficient regression fault detection, maximizing the number of executed test cases with high failure frequency, failure impact, and cross-functional coverage, in a limited period of time. This means that in cases where time required to execute a complete test suite exceeds available testing time, the multi-perspective framework will ensure that the most cost-efficient test cases will be executed.

D. Threats to Validity

External validity: We applied the proposed approach on three Android software applications. A threat may be that the results cannot be generalized for other domain. However, we selected the three most diverse applications in terms of functionality, age, architecture, and number of features. The age of the applications is directly correlated with the number of failures, the number of features is correlated with the size of a test suite, and functionality and architecture are correlated with the level of cross-functional coverage of tests. Although the proposed framework would need to be evaluated for different contexts, we believe that due to the varying characteristics of our software subjects, the results would be similar.

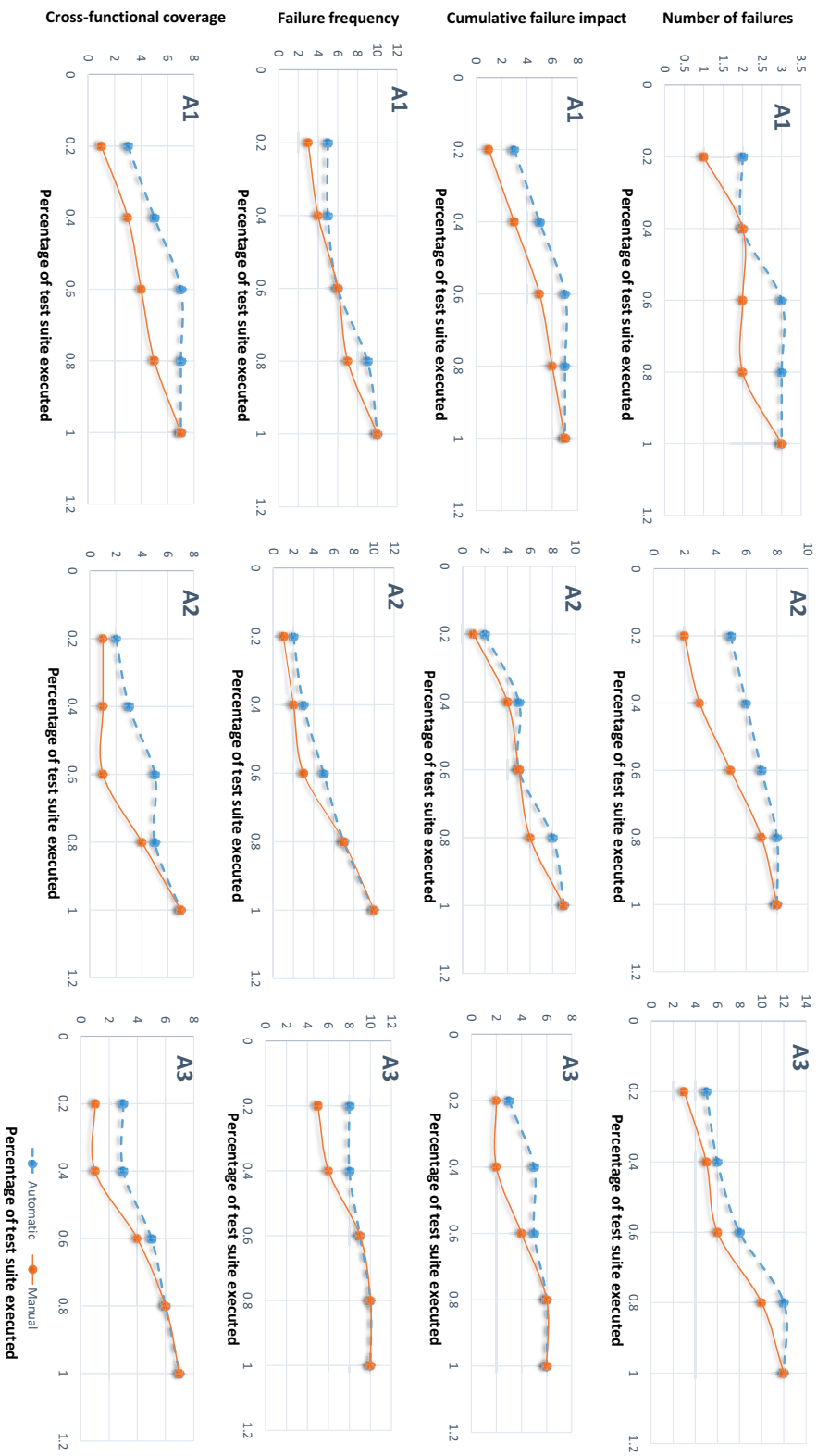

Fig. 1. Columns represent experiment subjects A1, A2, and A3. Rows represent four prioritization factors: the number of failures detected, the cumulative failure impact, failure frequency, and test cross-functional coverage.

Internal validity: In comparison with manual prioritization, we used test suites specified manually by the test team of five experts. A threat may be the choice of experts. While it may be that other testers order test cases differently, the team of five experts who ordered the tests that we used in the experiments consisted of senior experts who worked with mobile application development and testing from 10 to 15 years. Furthermore, as manual prioritization was made in two steps, first individually and later in a group, we analyzed the congruence of their suggested test orders for the same test suits. The results showed that their opinion coincided in most of the cases.

Construct validity: A threat to construct validity may be the selection of experimental setting for evaluation. However, we started our research from the existing case study, trying to improve current practice. Therefore, the choice of a case study is obvious. Another threat may be the choice of four specific prioritization factors. However, the factors were carefully chosen based on the interviews with test experts who have long experience in software development and testing. Besides, the proposed approach allows tuning the importance of factors, depending on the specifics of particular industrial environment. We believe that the general concept of history-based multi-perspective test prioritization, potentially with small alterations in some domain-specific parameters, can be beneficial for other continuous integration testing contexts. In future work, we may consider investigating other factors, for example, resource availability.

V. Conclusion

In this paper, we have presented an approach for multi-perspective regression test prioritization. The approach is designed to optimize regression testing for faster fault detection integrating three different perspectives: business perspective, performance perspective, and technical perspective. The approach has been validated in regression testing of three mobile device industrial applications, of approximately 3000 LOC. The results show that multi-perspective framework efficiently prioritizes test cases for faster and more efficient regression fault detection, maximizing the number of executed test cases with high failure frequency, failure impact, and cross-functional coverage, in a limited period of time. This means that in cases where time required to execute complete test suite exceeds available testing time, the multi-perspective framework will ensure that the most cost-efficient test cases will execute. In future work, we will extend the framework to support more prioritization factors.

Acknowledgment

This work is supported by the Research Council of Norway, through Certus SFI project.

References

[1] J.-H. Kwon, I.-Y. Ko, G. Rothermel, and M. Staats, "Test case prioritization based on information retrieval concepts," in *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, vol. 1, Dec 2014, pp. 19–26.

[2] H. A. Sulaiman, M. A. Othman, M. F. I. Othman, Y. A. Rahim, and N. C. Pee, "An improved history-based test prioritization technique technique using code coverage," *Advanced Computer and Communication Engineering Technology*, vol. 315, pp. 437–448, 2015.

[3] R. Beena and S. Sarala, "Code coverage based test case selection and prioritization," *CoRR*, vol. abs/1312.2083, 2013. [Online]. Available: http://arxiv.org/abs/1312.2083

[4] T. Noguchi, H. Washizaki, Y. Fukazawa, A. Sato, and K. Ota, "History-based test case prioritization for black box testing using ant colony optimization," in *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, April 2015, pp. 1–2.

[5] M. Sherriff, M. Lake, and L. Williams, "Prioritization of regression tests using singular value decomposition with empirical change records," in *Proc. of the The 18th IEEE Int. Symposium on Software Reliability*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 81–90. [Online]. Available: http://dx.doi.org/10.1109/ISSRE.2007.20

[6] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines," *Software Engineering, IEEE Transactions on*, vol. 40, no. 7, pp. 650–670, July 2014.

[7] B. Jiang and W. Chan, "Input-based adaptive randomized test case prioritization: A local beam search approach," *Journal of Systems and Software*, vol. 105, no. 0, pp. 91 – 106, 2015.

[8] H. Srikanth, M. B. Cohen, and X. Qu, "Reducing field failures in system configurable software: cost-based prioritization," in *ISRE*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 61–70.

[9] R. C. Bryce, S. Sampath, J. B. Pedersen, and S. Manchester, "Test suite prioritization by cost-based combinatorial interaction coverage," *Int. Journal Systems Assurance Eng. and Management*, pp. 126–134, 2011.

[10] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "An empirical study of the effect of time constraints on the cost-benefits of regression testing," in *Proc. of the 16th Int. Symp. on Foundations of software eng.* New York, NY, USA: ACM, 2008, pp. 71–82. [Online]. Available: http://doi.acm.org/10.1145/1453101.1453113

[11] H. Do and G. Rothermel, "Using sensitivity analysis to create simplified economic models for regression testing," in *ISTTA*. New York, NY, USA: ACM, 2008, pp. 51–62. [Online]. Available: http://doi.acm.org/10.1145/1390630.1390639

[12] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *Software Engineering, IEEE Transactions on*, vol. 36, no. 5, pp. 593–617, Sept 2010.

[13] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Timeaware test suite prioritization," in *ISSTA*. New York, NY, USA: ACM, 2006, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1146238.1146240

[14] L. Zhang, S.-S. Hou, C. Guo, T. Xie, and H. Mei, "Time-aware test-case prioritization using integer linear programming," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ser. ISSTA '09. New York, NY, USA: ACM, 2009, pp. 213–224.

[15] S. Alspaugh, K. R. Walcott, M. Belanich, G. M. Kapfhammer, and M. L. Soffa, "Efficient time-aware prioritization with knapsack solvers," in *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, ser. WEASELTech '07. New York, NY, USA: ACM, 2007, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/1353673.1353676

[16] 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?" *Inf. Softw. Technol.*, vol. 54, no. 7, pp. 739–758, Jul. 2012.

[17] D. Saff and M. D. Ernst, "An experimental evaluation of continuous testing during development," in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '04. New York, NY, USA: ACM, 2004, pp. 76–85.

[18] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proceedings of the 23rd Int. Conference on Software Engineering*, 2001, pp. 329–338.