Information and Software Technology 56 (2014) 1322–1344

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests

Chu-Ti Lin [a,*], Kai-Wei Tang [b], Gregory M. Kapfhammer [c]

[a] Department of Computer Science and Information Engineering, National Chiayi University, Chiayi, Taiwan
[b] Cloud System Software Institute, Institute for Information Industry, Taipei, Taiwan
[c] Department of Computer Science, Allegheny College, Meadville, PA, USA

A R T I C L E   I N F O

Article history:
Received 16 August 2013
Received in revised form 10 April 2014
Accepted 14 April 2014
Available online 2 May 2014

Keywords:
Software testing
Regression testing
Test suite reduction
Code coverage
Test irreplaceability

A B S T R A C T

Context: In software development and maintenance, a software system may frequently be updated to meet rapidly changing user requirements. New test cases will be designed to ensure the correctness of new or modified functions, thus gradually increasing the test suite's size. Test suite reduction techniques aim to decrease the cost of regression testing by removing the redundant test cases from the test suite and then obtaining a representative set of test cases that still yield a high level of code coverage.
Objective: Most of the existing reduction algorithms focus on decreasing the test suite's size. Yet, the differences in execution costs among test cases are usually significant and it may take a lot of execution time to run a test suite consisting of a few long-running test cases. This paper presents and empirically evaluates cost-aware algorithms that can produce the representative sets with lower execution costs.
Method: We first use a cost-aware test case metric, called Irreplaceability, and its enhanced version, called EIrreplaceability, to evaluate the possibility that each test case can be replaced by others during test suite reduction. Furthermore, we construct a cost-aware framework that incorporates the concept of test irreplaceability into some well-known test suite reduction algorithms.
Results: The effectiveness of the cost-aware framework is evaluated via the subject programs and test suites collected from the Software-artifact Infrastructure Repository — frequently chosen benchmarks for experimentally evaluating test suite reduction methods. The empirical results reveal that the presented algorithms produce representative sets that normally incur a low cost to yield a high level of test coverage.
Conclusion: The presented techniques indeed enhance the capability of the traditional reduction algorithms to reduce the execution cost of a test suite. Especially for the additional Greedy algorithm, the presented techniques decrease the costs of the representative sets by 8.10–46.57%.

1. Introduction

The goal of software testing is to execute the software system, locate the faults that cause failures, and improve the quality of the software by removing the detected faults. Testing is the primary method that is widely adopted to ensure the quality of the software under development [1]. According to the IEEE definition [2], a test case is a set of input data and expected output results which are designed to exercise a specific software function or test requirement. During testing, the testers, or the test harnesses, will execute the underlying software system to either examine the associated program path or to determine the correctness of a software function. It is difficult for a single test case to satisfy all of the specified test requirements. Hence, a considerable number of test cases are usually generated and collected in a test suite [3].

If the requirement set covered by a test case overlaps the set covered by another test case, then the two test cases may be redundant to each other. At the beginning of software testing, a large number of test cases may be automatically generated by a test case generator without considering the redundancies of test cases [4,5]. Yet, because the resources allocated to a testing team are usually limited, it may be impractical to execute all of the generated test cases. If software developers can reduce the test suite by removing the redundant test cases, while still ensuring that all test requirements are satisfied by the reduced test suite, then testing may be more efficient and the effectiveness of testing in unlikely to be compromised.

* Corresponding author. Tel.: +886 5 2717227.
E-mail address: chutilin@mail.ncyu.edu.tw (C.-T. Lin).

http://dx.doi.org/10.1016/j.infsof.2014.04.013
0950-5849/© 2014 Elsevier B.V. All rights reserved.

The process of removing the redundant test cases is called test suite reduction or test suite minimization [6,7].

Additionally, evolutionary development, incremental delivery, and software maintenance are common in software development [8,9]. In such development processes, the functionality of a software system may be refined to meet the customer's needs or may be delivered incrementally. Each time the software developers modify the system, they may also introduce some faults. New test cases should be added to ensure the quality of new functions. The existing test cases should also be re-executed in order to detect the faults caused by imperfect debugging. Such an activity is called regression testing [2,10]. In the process of software development, more and more test cases will be included, thus often causing some test requirements to be associated with more than one test case. To reduce the cost of regression testing, test suite reduction can also be applied to remove the redundant test cases [11].

The test suite reduction problem can be defined as follows [6,7,9]:

Given:

- A set $T = \{t_1, t_2, \ldots, t_n\}$ representing the test cases in the original test suite, where n denotes the number of test cases (i.e., $n = |T|$).
- A set of test requirements $R = \{r_1, r_2, \ldots, r_m\}$ in which each test requirement must be satisfied by at least one of the test cases in T, where m denotes the number of test requirements (i.e., $m = |R|$).
- The binary relation between T and R: $S = \{(t, r) \mid t \text{ satisfies } r, t \in T \text{ and } r \in R\}$.

Objective:

- Find a subset of the test suite T, denoted by a representative set RS, to satisfy all of the test requirements satisfied by T.

Let us consider the binary relation between $T_0 = \{t_1, t_2, t_3, t_4, t_5\}$ and $R_0 = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ that is shown in Table 1 as an illustration. If the set of test cases $\{t_1, t_2\}$ are executed instead of T_0, all of the requirements in R_0 are still satisfied. Thus, the test cases $\{t_3, t_4, t_5\}$ do not need to be executed. In this example, the subset $\{t_1, t_2\}$ is the minimal representative set.

In fact, the test suite minimization problem can be reduced to the minimum set cover problem [6]. Karp proved that the set cover problem is NP-complete [12]; yet, many techniques have been proposed to obtain the near-optimal solution for the test suite reduction problem. Even though the representative sets produced by these techniques are not guaranteed to be optimal, they can significantly decrease both the size of the test suite and the cost associated with its execution. However, to the best of our knowledge, most of the existing reduction algorithms ignore the significant differences in execution costs among test cases. In response to this limitation, this paper presents a cost-aware test suite reduction technique based on the concept of test irreplaceability.

The main contributions of this paper include: (C1) presenting a cost-aware test case metrics, called Irreplaceability and EIrreplaceability, based on the concept of test irreplaceability; (C2) constructing a cost-aware framework that incorporates the concept of test irreplaceability into some well-known test suite reduction algorithms; (C3) empirically evaluating the effectiveness of the presented test suite reduction techniques and evaluating whether these methods select many test cases in common during the test suite reduction process. It is important to note that, in comparison with the preliminary version of this work (i.e., [9]), this paper provides several additional contributions. Considering (C1) as an example, although the test case metric, Irreplaceability, was presented in [9], this paper further presents the enhanced version, EIrreplaceability, which strictly dominates Irreplaceability. Considering (C2), the work in [9] only explained how to use Irreplaceability to evaluate the test cases and selected the test cases based on the concept of the additional Greedy algorithm. This paper further incorporates the presented cost-aware test case metrics into two additional well-known reduction algorithms. Considering (C3), in addition to the small subject programs used in [9], this paper further includes empirical studies with a larger subject program. Moreover, this paper evaluates the common rates of the representative sets according to the concept developed by Zhong et al. in [3].

The remainder of this paper is organized as follows. Section 2 reviews some well-known test suite reduction algorithms and some cost-aware regression testing techniques. In Section 3, we discuss the differences in execution costs among test cases and explain how the differences influence test suite reduction. Based on these discussions, we show how to use test irreplaceability to evaluate each test case. In Section 4, we propose the cost-aware algorithms by incorporating test irreplaceability into the existing reduction algorithms. Section 5 reports on the results from the empirical studies. Finally, Section 6 furnishes some concluding remarks.

2. Related work

Test suite reduction, test case prioritization, and test case selection are three primary techniques to make regression testing more efficient and effective [13]. As described in Section 1, test suite reduction techniques remove the redundant test cases and then produce a representative set of test cases that still yield a high level of code coverage. Test case prioritization techniques reorder the test cases according to some specific criteria such that the tests with better fault detection capability are executed early during the regression testing process. Test case selection techniques run a subset of the test cases that execute the modified source code in order to ensure the correctness of the functionalities of the updated program [14]. Recall that this paper aims to present and empirically evaluate the cost-aware test suite reduction algorithms. In order to introduce the state-of-the-art relevant techniques, Sections 2.1.1–2.1.3 first review three well-known test suite reduction algorithms that will be used to explain the presented technique in Section 4, and then Section 2.1.4 reviews some existing cost-aware test suite reduction techniques. To the best of our knowledge, few cost-aware test case selection techniques have been proposed in the literature, while a lot of cost-aware test case prioritization techniques have been proposed and have received considerable attention. Thus, Section 2.2 focuses on the reviews of the cost-aware regression test case prioritization techniques.

2.1. Reviews of well-known test suite reduction techniques

2.1.1. Additional Greedy algorithm

The additional Greedy algorithm [15], hereafter called Greedy for simplicity, is a well-known method for finding the near-optimal solution to the test suite reduction problem. This algorithm repeatedly moves the test which covers the most unsatisfied test

Table 1
An example of a test suite.

Test suite T_0	Requirement set R_0					
	r_1	r_2	r_3	r_4	r_5	r_6
t_1		●	●		●	
t_2	●			●		●
t_3					●	●
t_4		●	●	●		
t_5				●		●