Information Sciences 422 (2018) 318–335

Contents lists available at ScienceDirect

Information Sciences

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

A cost-effective software testing strategy employing online feedback information☆

Zhi Quan Zhou[a,*], Arnaldo Sinaga[c], Willy Susilo[a], Lei Zhao[b], Kai-Yuan Cai[b]

[a] Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia
[b] Department of Automatic Control, Beijing University of Aeronautics and Astronautics, Beijing 100191, China
[c] Del Institute of Technology, Indonesia

ARTICLE INFO

Article history:
Received 26 September 2009
Accepted 28 November 2015
Available online 1 September 2017

Keywords:
Software testing
Random testing
Partition testing
Dynamic partitioning
Online feedback
Early fault detection
Software cybernetics

ABSTRACT

An online partitioning strategy is presented, in which test cases are selected based on feedback information collected during the testing process. The strategy differs from conventional approaches because the partitioning is performed online rather than off-line and because the partitioning is not based on program code or specifications. It can, therefore, be implemented in the absence of the source code or specification of the program under test. The cost-effectiveness of the proposed strategy has been empirically investigated with a set of subject programs, namely, SPACE, SED, GREP, and the Siemens Suite of Programs. The results demonstrate that the proposed strategy constantly achieves large savings in terms of the total number of test case executions needed to detect all faults.

1. Introduction

Testing is a very expensive part of software production and maintenance. It is estimated that, in general, testing consumes between 30% and 50% of the software development time [27]. Random testing (RT) [23] is a basic testing method, and often serves as a benchmark against which the effectiveness of other testing techniques are measured. In RT, test cases are selected randomly from the input domain. It can, therefore, avoid the overhead incurred in code- or specification-based partitioning strategies. RT is often employed to test real-world products and incorporated into test case generation tools [1,4,17,18,24,34].

Malaiya introduced an *antirandom testing* technique [26] to enable quicker detection of the first failure. In other words, antirandom testing attempts to reduce the number of test cases needed to be run before the first failure can be revealed. Compared with RT, which requires each test case to be selected randomly regardless of the previously executed test cases, antirandom testing only selects the first test case randomly, and each subsequent test case is selected by choosing the one with the maximum total distance to all the previously executed test cases. Antirandom testing requires that the total number of test cases be known in advance. Its randomness is also limited since only the first test case is chosen randomly and the sequence of subsequent test cases is deterministic.

☆ A preliminary version of this paper was presented at the International Conference on Quality Software (QSIC'09) [36].

* Corresponding author.
E-mail addresses: zhiquan@uow.edu.au (Z.Q. Zhou), ams939@uow.edu.au (A. Sinaga), wsusilo@uow.edu.au (W. Susilo), zhaolei@asee.buaa.edu.cn (L. Zhao), kycai@buaa.edu.cn (K.-Y. Cai).

http://dx.doi.org/10.1016/j.ins.2017.08.088
0020-0255/

In order to detect the first failure quicker without the limitations of antirandom testing, an *adaptive random testing* (ART) strategy has been proposed [2,8–10,13,25]. ART is based on the intuition that when failure-causing inputs are clustered, selecting an input close to previously executed non-failure-causing test cases will be less likely to detect a failure. ART, therefore, proposes to have test cases evenly spread over the entire input domain. It differs from antirandom testing in that it preserves the randomness since all test cases in ART are randomly selected and ART does not require the predetermination of the total number of test cases to be run.

A related technique is known as *adaptive testing*, developed by Cai et al. [6]. Both the objective and the approach of adaptive testing is different from ART. ART was developed as an enhancement to RT with the objective of using fewer test cases to detect the first failure. Adaptive testing, on the other hand, adjusts the selections of test actions online following the idea of adaptive control, to achieve an optimization objective, such as minimizing the total cost of detecting and removing multiple faults. Adaptive testing involves off-line partitioning of the input domain. It has been shown that partitioning strategies can have a significant impact on testing effectiveness [6]. While various partitioning techniques have been proposed [12,14,29], most of these techniques partition the input domain off-line and they incur non-trivial overheads.

In order to find a way of achieving effective partitioning without heavy overheads, Cai et al. conducted a pilot study [7] following the idea of *software cybernetics*, which explores the interplay between software and control [5]. Cai et al. [7] adopted a new testing paradigm. First, a very large test suite is given. Second, test cases are selectively executed through dynamically partitioning the test suite online. Their strategy is based on the intuition that a test case that has detected a failure previously should have a higher capability of detecting a failure again during the evolution of the software under test. In their approach, there are two partitions: partition 0 and partition 1. Initially, all test cases are stored in partition 1, and partition 0 is empty. A test case t is randomly selected from partition 1 to test the program. If no failure is detected, then t is considered not powerful and moved to partition 0. Otherwise, the program under test may be modified to remove a fault. In practice, such an attempt to debug the program does not always correctly remove the fault, and may sometimes introduce new faults [22]. In any case, the modified program needs to be retested on t. If a failure is detected again, then the above modify-and-retest process will be repeated until no more failure can be detected. Then t will be returned to partition 1, from which the next test case will be selected randomly. The testing process will stop when partition 1 becomes empty or when the given stopping criterion is met. Cai et al. experimentally compared this dynamic partitioning strategy with two other random testing strategies and found that the dynamic partitioning strategy outperformed the other two [7].

We note that the above dynamic partitioning algorithm is quite simple, and that all the proposed test case selection strategies used sampling with replacement [7]. In real-world software testing, it is more practical to use sampling without replacement to save cost. This paper follows the direction shown in Cai et al. [7] to further investigate the usefulness of online feedback information.

The motivation of this research is the fact that in real-world software development, the software will undergo many changes/versions. The test pool is very large and consists of both already-applied and not-yet-applied test cases. Every time a change is made, the software needs to be retested. Considering the total cost of testing across multiple versions, how can we select test cases in the most cost-effective manner so as to use the minimum number of test case executions to detect and remove the maximum number of defects? This situation is more complex than conventional regression testing and we want a practical method that is easy to implement without the need for sophisticated tool support (such as those for change tracking and analysis) or the need for any kind of test case coverage information or assumptions of the availability of the source code of the program under test. In short, we want a method that only depends on information that can be readily collected from test case executions (for example, pass/fail information) – this will enable the technique to be accepted by real-world practitioners.

The contributions of this research are summarized as follows: (i) A family of dynamic partitioning algorithms are proposed to selectively execute test cases from a given large test suite for evolving software. The dynamic partitioning strategy is based on online feedback collected during test case executions without the need to refer to any kind of coverage information, change analysis, or human judgement. (ii) Empirical study results demonstrate that the proposed strategy is more cost-effective than the strategy without using feedback information. (iii) Empirical study results show that feedback information on early fault detection is very useful for further improving the cost-effectiveness of testing. In particular, one such algorithm DP1SE constantly outperforms all the others and, hence, is the best among all 10 algorithms studied in this paper. The results of this research further justify the emergence of the area of software cybernetics.

The rest of this paper is organized as follows: Section 2 proposes a family of 7 test case selection algorithms, where the first is pure random testing and serves as a benchmark for measuring the cost-effectiveness of the other 6 algorithms that employ online feedback information, which is pass/fail information collected during test case executions. To investigate the cost-effectiveness of the proposed algorithms, Section 3 presents the design and results of experiments with 3 large subject programs, namely, the SPACE, SED and GREP programs. In Section 4, we consider more feedback information (namely, information as to how early a fault can be detected) and develop 3 additional algorithms. Empirical study results of these 3 algorithms are also presented. Section 5 conducts an additional series of experiments with a set of smaller subject programs, namely, the Siemens Suite of Programs, to enhance the external validity of our study. Section 6 presents further discussion and concludes the paper.