The Journal of Systems and Software 139 (2018) 14–31

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Using reliability risk analysis to prioritize test cases

Ying Wang[a], Zhiliang Zhu[a], Bo Yang[a], Fangda Guo[b], Hai Yu[a,*]

[a] Software College, Northeastern University, NO. 195, Chuangxin Road, Hunnan District, Shenyang, PR China
[b] School of Computer Science and Engineering, Northeastern University, NO. 195, Chuangxin Road, Hunnan District, Shenyang, PR China

ARTICLE INFO

Article history:
Received 31 May 2017
Revised 4 January 2018
Accepted 22 January 2018
Available online 2 February 2018

Keywords:
Regression testing
Test case prioritization
Probabilistic risk analysis
Information flow
Complex network

ABSTRACT

In this paper, we present a risk-based test case prioritization (Ri-TCP) algorithm based on the transmission of information flows among software components. Most of the existing approaches rely on the historical code changes or test case execution data, few of them effectively use the system topology information covered by test cases when scheduling the execution of test cases. From the perspective of code structure, the proposed algorithm firstly maps software into an information flow-based directed network model. Then, functional paths covered by each test case are represented by a set of barbell motifs. Finally, combining with probabilistic risk analysis (PRA) and fault tree model, we assign a priority to each test case by calculating the sum of risk indexes of all the barbells covered by it. Experimental results demonstrate that Ri-TCP technique has a higher detection rate of faults with serious risk indicators and performs stably in different systems, compared with the other state-of-the-art algorithms.

1. Introduction

Regression testing is an important guarantee for software quality, whose purpose is to ensure that the modifications in previous versions of software meet the users' requirements (Kung et al., 1996). In the life cycle of software testing, regression testing plays a significant role, which approximately accounts for 50% of the total maintenance cost due to its high frequency of being executed (Harrold, 2009). To improve the test efficiency and reduce test effort, testers might schedule test cases in an order according to some criterion to make the critical test cases be executed preferentially, this is so called "test case prioritization technique" (Elbaum et al., 2001b; Wong et al., 1997).

A test case is a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. Test cases are the cornerstones of quality assurance where they are developed to verify the quality and behavior of a product (IEEE, 2010). Test case prioritization technique aims to achieve code coverage at the fastest rate possible, increase assurance in reliability of the system at a faster rate, or improve the fault detection ability of test case suite during the testing process (Marchetto et al., 2016). A higher fault detection rate can provide earlier feedback on the system under test, enable earlier debugging, and increase the likelihood that, if the testing period is cut short, test cases that offer the greatest fault detection capacity in the available testing time will have been executed (Hao et al., 2016).

Numerous algorithms are proposed to address the test case prioritization problem. Of these, code-based prioritization techniques have drawbacks when dealing with large-scale software because of the statement and block level information is hard to manage (Ma and Zhao, 2008). The structural complexity-based prioritization strategy is to assign weights to classes based on the complexity of system topology, and then prioritize the test cases with the goal of maximizing the total or additional covered indicators. However, they ignored the information transmission relationships between classes that are covered by test cases. Thus, approaches to scheduling test cases by comprehensively analyzing the coverage information are valuable.

Risk analysis theory is successfully applied to software testing field, to improve productivity and reduce the costs of testing. The approaches Redmill (2005), Redmill (2004), Felderer and Ramler (2013), Felderer et al. (2012) and Amland (2000) addressed risk-based testing at a general level. Redmill (2005, 2004) emphasized the human and organizational factors. Employing risk as the basis for test planning did not provide a formula for perfection. Test planners must retain responsibility, but informed use of risk could provide illuminating guidance. Felderer and Ramler (2013) and Felderer et al. (2012) showed a model-based approach to risk-based testing, with the focus on product risks affecting the quality of the product itself. In Felderer and Ramler (2013), they pre-

* Corresponding author.
E-mail addresses: wangying8052@163.com (Y. Wang), zzl@mail.neu.edu.cn (Z. Zhu), yb9506@126.com (B. Yang), yuhai@mail.neu.edu.cn, yuhai@126.com (H. Yu).

https://doi.org/10.1016/j.jss.2018.01.033
0164-1212/

sented a generic risk-based testing methodology and a procedure how it can be introduced in a test process. Based on this procedure, four stages of risk-based test integration were derived, i.e., initial risk-based testing, risk-based test reporting, risk-based test planning, and optimization of risk-based testing. Risk-based testing has a high potential to improve the software development and test process as it helps to optimize the allocation of resources and provides decision support for the management (Amland, 2000). Thus, considering risk indicators when prioritizing the test cases is essential for improving test efficiency.

In this paper, we propose a strategy combining three reliability risk factors – dynamic execution probability, fault-proneness and failure consequence – to schedule test cases to be executed. By equating the functional invocations with the transmissions of information flow, the software system is mapped into a class-level directed network model. Based on the complex network theory, we decompose the functional paths into a series of barbell motifs which consists of a class node pair and an information transmission relationship contained therein. With the aid of the fault tree model, we quantitatively analyze all the state events caused by the failure of each barbell motif in the system. Then, the risk index covered by test case is treated as a basis for ordering their execution. By comparing with the other state-of-the-art techniques based on several case studies, we show that the proposed approach performs better in effectiveness and stability across different software systems. The main contributions of this approach are summarized as follows.

- A class-level directed network model based on information flow for analyzing communication relationships between modules of software.
- An evaluation scheme for quantifying the risk indexes of classes in the system using the PRA model.
- A measurement to assess the risk coverage of test cases combining fault tree analysis and barbell motifs.
- A comprehensive comparison with previous studies from the perspective of detection rate of faults with high risk index.

The remainder of this paper is organized as follows. Section 2 discusses related research and Section 3 introduces the Ri-TCP technique. In Section 4, an evaluation indicator is described. In Section 5, we provide a comparison with previous research and discuss the experiment results. Finally, we give our conclusion in Section 6.

2. Related work

2.1. Test case prioritization techniques

Considering the coverage information as a target, test case prioritization techniques produce an optimal order for maximizing the coverage rate of certain factor (e.g., branch coverage, decision coverage, or statement coverage) as early as possible (Do et al., 2010). Rothermel et al. (2001) transformed the test case prioritization problem into a solution of searching the optimal order from all possible permutations of test cases. Its formalized definition is described as follows:

Definition 1. *Test case prioritization problem.* Given a test suite T, the set PT consisting of all the permutations of test cases in T, and a function f from PT to the set of real numbers, find a $T' \in PT$ such that $(\forall T'')(T'' \neq T')[f(T') \geq f(T'')]$.

Yoo and Harman (2012) surveyed the area of prioritization technique and discussed open problems and potential directions for future research. According to their paper, we categorized the existing approaches into four types: coverage-based prioritization, interaction testing, cost-aware test case prioritization and prioritization approaches based on other criteria.

Coverage-based prioritization. By analyzing the static call graphs of JUnit test cases and the program under test, Mei et al. (2012) prioritized the test cases in the absence of coverage information operating on Java programs tested under the JUnit framework. As dynamic coverage-based techniques use actual coverage information while their approach used estimated coverage information, the former was intuitively better than the latter in terms of fault-detection effectiveness. However, by avoiding the need to instrument code and execute test cases, this approach might be more applicable than dynamic coverage-based approaches in cases where gathering coverage information was inappropriate or was not cost effective.

Jeffrey and Gupta (2006) proposed a test case prioritization technique based on the coverage requirements presented in the relevant slices of the outputs of test cases. They called this approach the "REG+OI+POI" heuristic strategy for prioritization, where REG, denotes REGular statement (branches) executed by the test case, OI denotes the Output Influencing and POI denotes the Potentially Output Influencing statements (branches) executed by the test case. The experimental results suggested that accounting for relevant slicing information, along with information about the modifications traversed by each test case, had potential when used as part of the test case prioritization process.

Interaction testing. Interaction testing is required when system under test involves multiple combinations of different components. Bryce and Memon (2007) also applied the principles of interaction coverage to the test case prioritization of event-driven software. They extended the notion to t-way interactions over sequences of events. Prioritization by interaction coverage of events improved the rate of fault detection in half of our experiments. The test suites that include the largest percentage of 2-way and 3-way interactions had the fastest rate of fault detection when prioritized by interaction coverage.

Previous studies used in tools to generate software interaction test suites have been evaluated on criteria of accuracy, execution time, consistency, and adaptability to seeding and constraints. Bryce and Colbourn (2005) prioritized interaction test cases based on user specified importance. For example, an operating system with a larger user base might be more important than one with a smaller user base. After weighting each level value for each factor, they calculated the combined benefit of a given test by adding the weights of each level value selected for the test.Computational results suggest that the greedy methods for constructing biased covering arrays could be useful when testers desire a prioritized ordering of tests.

Cost-aware test case prioritization. Yoo et al. (2009) introduced a test case prioritization technique, which can significantly reduce the required number of pair-wise comparisons by clustering test cases. The paper demonstrated that clustering without input parameters could outperform unclustered coverage-based technologies, and discussed an automated process that could be used to determine whether the application of the proposed approach would yield improvement.

Walcott et al. (2006) presented a regression test prioritization technique that used a genetic algorithm to reorder test suites in light of testing time constraints. Experiment results indicated that our prioritization approach frequently yields higher average percentage of faults detected (APFD) values, for two case study applications, when basic block level coverage was used instead of method level coverage. The experiments also revealed fundamental trade-offs in the performance of time-aware prioritization.

Prioritization approaches based on other criteria. Elbaum et al. (2001b, 2001a) performed a series of experiments to explore how the three factors-program structure, test suite composition, and change characteristics-affect the fault detection rate of test suites. Using a multiple regression model, they illustrate which metric