Postprint of article in *Journal of Systems and Software* 140 (2018) 109–110

Editorial

Introduction to the special issue on program debugging

Software finds itself in virtually every system or infrastructure today. In particular, it is extensively applied to critical systems in such major areas as aeronautics, medicine, engineering, transportation, power supply, and business and finance. This trend leads to much more demanding standards in software quality because even a small software fault may result in huge losses. For example, the National Institute of Standards and Technology estimated that software failures cost the United States 0.6 percent of its GDP[1], or $112 billion according to the latest World Bank statistics[2].

Owing to the escalating demands for software functions, the development process tends to have much tighter schedules and releases have to be delivered within short intervals. As a consequence, debugging, which is one of the most important tasks in the software lifecycle, is facing great challenges. On one hand, tighter schedules are more likely to introduce bugs. On the other hand, they require much higher efficiency of debugging.

As manual debugging is expensive and inefficient, a large number of techniques have been developed for less human involvement. These techniques range from test assistance through fault localization to automatic bug repairing. Even though this area has received much attention and produced a large body of studies, there are still many unresolved challenges. For instance, the accuracy of lightweight automatic debugging techniques still has plenty of room for improvement. There may be ambiguities in comparing executions in the presence of multiple causative faults, or difficulties in reliably recording and replaying failed executions. These uncertainties may lead to bug fixes that introduce even more faults into the software. In addition, the applications of debugging techniques to various emerging areas are worth studying.

The theme of this special issue is “Program Debugging”. We solicited submissions from both academia and industry to present innovative ideas to address the latest challenges, make breakthroughs, and discuss future trends.

The call for papers attracted 21 submissions covering diverse relevant topics. Each submitted article was carefully evaluated by at least two experts in the field. After a rigorous peer review process, nine high-quality research papers have been selected for the issue. We can classify the articles into five categories as follows:

1. Overview: Know your enemy

How severe could software failures be in IT history? What were the main causes of these failures and what lessons did people learn from disasters? Overall, how urgent and important should it be for people to have a deep understanding of software faults or defects and know how to cope with them? Paper 1, titled “Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures” by W. Eric Wong et al., presents a comprehensive review to answer the above questions. Specifically, it reviews the

[1] http://www.nist.gov/director/planning/upload/report02-3.pdf
[2] https://data.worldbank.org/indicator/NY.GDP.MKTP.CD

roles that program faults played in notable software failures, provides a detailed analysis of the causes of the failures, and summarizes the losses and the lessons learned. As a systematic review, this article provides guidelines for both researchers and engineers for better software debugging and quality assurance.

2. Oracle problem

A test oracle should serve as a precise benchmark for deciding whether a test execution completely satisfies the software specification. However, such an oracle is generally difficult to obtain owing to the ambiguity of software specifications usually described in a natural language. Paper 2, titled "Automated generation of (F)LTL oracles for testing and debugging" by Ingo Pill and Franz Wotawa, solves the problem by proposing to construct a test oracle directly and automatically from formal specifications in Linear Temporal Logic (LTL). Specifically, given an LTL specification and an execution trace, SAT encoding is generated to decide whether the trace satisfies the given LTL property. The authors have demonstrated that the appealing performance of the method through empirical studies.

3. Test optimization

Software testing that provides rich information such as execution traces, internal states, and execution results usually serves as a guide for program debugging. Hence, optimizing software testing activities for better debugging performance is worth investigating.

Paper 3, titled "A multi-level feedback approach for the class integration and test order problem" by Miao Zhang et al., presents a multi-level feedback strategy to resolve the "class integration and test order" problem. The strategy calculates the test profit and assesses the performance for each class in the source code class using a reward and punishment mechanism. The order of test integration is then determined accordingly. Not only have the authors applied the strategy to commonly adopted benchmarks, but they have also demonstrated its applicability to eight industrial programs. Compared with traditional graph-based and search-based approaches, the proposed strategy does not break cycles but search for optima in integrated methods, hence significantly reducing execution time.

The execution order of test cases is deemed as one of the key factors that influence fault detection effectiveness during debugging. Paper 4, titled "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering" by Jinfu Chen et al., presents a method based on Adaptive Random Sequences (ARSs) for prioritizing test executions. The method constructs ARSs whose neighboring test cases are as diverse as possible by means of K-medoid clustering based on black-box information. This method performs better than previous techniques in terms of earlier detection of program faults.

4. Fault localization

As a core component of program debugging, fault localization has been extensively studied, but with the fast developments in the software industry, there are still many emerging problems to be solved.

Paper 5, titled "The Bayesian network based program dependence graph and its application to fault localization" by Xiao Yu et al., presents a fault localization technique founded on a new probabilistic graphical model called the Bayesian Network based Program Dependence Graph (BNPDG). Compared with traditional models such as probabilistic pro-

gram dependence graph, the BNPDG-based method does not have the limitation in reasoning across nonadjacent nodes. Moreover, by taking the output nodes as the common conditions to compute the conditional probability of each non-output node, the BNPDG-based method can produce more exact localization results.

Paper 6, titled “Fault localisation for WS-BPEL programs based on predicate switching and program slicing” by Chang-ai Sun et al., focuses on the popular service-oriented architecture. With the help of Business Process Execution Language for Web Services (WS-BPEL), coordination among multiple independent Web services becomes easy and flexible by hiding many implementation details. However, the hidden information brings difficulties to software fault localization. This article presents a solution to address the issue. The proposed technique first narrows down the range of risky software blocks through predicate switching and then performs an in-depth exploration within the suspicious blocks for more exact localization results.

Multiple-fault localization has attracted the interests of more and more researchers because real-life programs seldom contain only one single fault. Paper 7, titled “Localizing multiple software faults based on evolution algorithm” by Yan Zheng et al., reduces the task of simultaneously localizing multiple faults to a search problem. With the help of genetic and simulated annealing algorithms, the authors propose a Fast Software Multi-Fault Localization (FSMFL) framework and implement a prototype. A fitness function is used to assign higher scores to candidature program entities covering more failed and less passed test cases. The experimental results are positive.

5. System traces

As mentioned in Section 3, execution traces that reveal system behavior can provide rich information for program analysis and debugging. However, an extremely large number of traces and system states severely hinder the application of such information in practice.

Paper 8, titled “R-SHT: a state history tree with R-tree properties for analysis and visualization of highly parallel system traces” by Loïc Prieur-Drevon et al., focuses on distributed computer systems with many threads and resources, for which the number of traces is generally too large to analyze manually. The authors adopt R-Tree techniques and propose a configurable build algorithm that reorganizes data in the sub-trees to optimize system performance. They demonstrate that the proposed method can preserve optimal disk and memory usage but boosts the query performance in highly parallel traces by several orders of magnitude.

Paper 9, titled “Omniscient debugging for executable DSLs” by Erwan Bousse et al., aims at trace management and analysis of model programs. It presents a generic solution for omniscient debugging, which supports a wide range of executable Domain Specific Languages (DSLs). The solution is based on a pattern that defines runtime services independently of metaprogramming approaches with a view to providing efficient generic trace management facilities. Compared to traditional solutions, although the proposed method slightly slows down the average system execution time, it does not need to copy the model at each step and hence is much more efficient in terms of both the time and space for trace analysis.

Acknowledgments

We would like to thank all the authors for their high-quality contributions to the special issue. In addition, our appreciation is due to all the reviewers for their great effort and

constructive comments. We are also grateful to the editor-in-chief, the special issue managing guest editor, and the journal manager of JSS for their support throughout the process of preparing the special issue.

Guest Editors

Xiaoyuan Xie *
School of Computer Science, Wuhan University,
Wuhan, Hubei 430072, China

Markus Stumptner
University of South Australia, Adelaide, Australia

T.H. Tse
The University of Hong Kong, Pokfulam, Hong Kong

* Corresponding author.
E-mail address: xxie@whu.edu.cn