Empirical Software Engineering (2019) 24:329–331
https://doi.org/10.1007/s10664-018-9678-6

Introduction to the special issue on software analysis, evolution, and reengineering

Gabriele Bavota[1] · Andrian Marcus[2]

Published online: 11 January 2019

It is a pleasure to introduce the papers in this Special Issue, featuring extended papers from the 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2017). SANER is the result of the merger of two conference series: the Working Conference on Reverse Engineering (WCRE) series; and the European Conference on Software Maintenance and Reengineering (CSMR) series.

The main track of the conferences included eight sessions organized by topics, which reflect current research trends in the field: mining software repositories and open source software development; feature and knowledge management; mobile applications and energy consumptions; software and models analysis; source code and data transformation; program comprehension; software development support; code smells and refactoring; and testing and fault localization.

After the conference, held in Klagenfurt, Austria, we selected six of the 34 accepted papers and invited their authors to submit an extended version to the Special Issue of the Springer Journal of *Empirical Software Engineering*. Paper selection was based on the reviews, feedback collected during the online program committee discussion, and the presentation at the conference. To be included in this special issue, we asked the authors to substantially enhance and revise their conference papers. Five of the papers went through a thorough reviewing process, undergoing multiple review rounds, and are included in this special issue:

1 "Redundancy-Free Analysis of Multi-Revision Software Artifacts" By Carol V. Alexandru, Sebastiano Panichella, Sebastian Proksch, and Harald C. Gall

The increasing interest in the Mining Software Repositories (MSR) field has pushed the software engineering research community to propose techniques and tools aimed at supporting

✉ Gabriele Bavota
gabriele.bavota@usi.ch

[1] Università della Svizzera italiana (USI), Lugano, Switzerland

[2] The University of Texas at Dallas, Richardson, TX, USA

large-scale MSR studies. In this context, Alexandru et al. present a framework named LISA (*Lean Language-Independent Software Analyzer*) aimed at efficiently representing and analyzing multi-revisioned software artifacts, such as software artifacts evolving in a versioning system. The authors observe that subsequent revisions of a software system typically exhibit a substantial overlap. Thus, the proper handling of this redundancy can help in substantially facilitating large-scale studies performed on multi-revisioned artifacts.

2 "An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems" By Alexandre Decan, Tom Mens, and Philippe Grosjean

The authors study the evolution of package dependency networks in seven software ecosystems composed by packages developed for a specific programming language (e.g., CRAN for R). The presented study compares how the dependency networks evolved over time across these seven ecosystems, presenting metrics aimed at characterizing the networks from different perspectives (e.g., changeability, fragility). The authors reveal substantial differences across the studied ecosystems, and provide initial evidence of laws of software ecosystems evolution.

3 "Shorter Identifier Names Take Longer to Comprehend" By Johannes C. Hofmeister, Janet Siegmund, and Daniel V. Holt

The paper focuses on the role played by identifier names during program comprehension. The authors investigate how different styles of identifiers (single letters, abbreviations, and words) impact on program comprehension by running a study involving 72 professional developers asked to localize defects in code snippets. The reported findings highlight the importance of using complete words in the naming of identifiers: When full words are used the developers experienced a 19% reduction of the time needed to find a defect.

4 "Effective Fault Localization of Automotive Simulink Models: Achieving the Trade-Off Between Test Oracle Effort and Fault Localization Accuracy" By Bing Liu, Shiva Nejati, Lucia, and Lionel C. Briand

In the context of model-based development, the authors of this paper present an approach for improving fault localization accuracy for Simulink models. The authors highlight that, while one way of improving the accuracy of fault-localization is to simply add more test cases, this operation comes with a cost related to the manual definition of the test oracles. Thus, they propose an approach combining two different components: (i) a search-based test generation algorithm to increase test suite diversity; and (ii) a predictor aimed at predicting whether additional test cases are likely to have a positive impact on the fault localization accuracy. The authors show the benefits of their approach when working with small test suites.

5 "Querying Distilled Code Changes to Extract Executable Transformations" By Reinout Stevens, Tim Molderez, and Coen De Roover

Many MSR studies rely on change distilling algorithms to extract AST-level change operations that transform a specific version of a code component into a target one. This is useful, for example, when analyzing the code changes implemented in a commit. The authors face the challenge of mining the output of the change distilling algorithms to automatically identifying minimal and executable change sequences that implement a specific evolution pattern (e.g., a rename method refactoring). This can be used, for example, to automatically detect refactoring operations performed in the history of a software system.

We hope that the readers will appreciate these five articles, which are representative of the main topics covered at the conference and are also exemplars of high quality research presented at the conference, year after year.

Acknowledgements We would like to thank the reviewers for their rigor and dedication while reviewing the submissions for this special issue, as well as the authors for fulfilling all the requests and providing such an excellent work. We are also grateful for the continuous support by the editorial staff of the Journal of Empirical Software Engineering and by the Editors-in-Chief Robert Feldt and Tom Zimmermann.