Introduction to the special issue on source code analysis and manipulation

Gabriele Bavota¹ · Michaela Greiler²

Published online: 20 March 2018

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Research in the area of source code analysis is concerned with the main and most essential artifact and building block in the software engineering process. Those building blocks include any fully executable description of the software system starting from very low level expressions such as machine code to high level descriptions of the system using high level languages or even graphical representations. It is the source code where we can find answers to many questions one might have about the software system. And it is also the source code that represents the essential truth about the behavior and execution of a system. Therefore, it is an important and crucial requisite to analyze, manipulate and learn from it. In this special issue, we have selected four excellent papers that look at different aspects of source code analysis.

Those articles have been selected from the accepted submissions to 16th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2016). SCAM brings together researchers and practitioners to enhance theory, techniques, and applications that focus on the analysis and manipulation of source code. To be included in this special issue the papers have been substantially enhanced and revised and went through another thorough reviewing process undergoing multiple review rounds. The selected studies tackle topic such as change impact analysis, information retrieval in software engineering, comparison of mutation testing and evaluation of similarity analysis techniques and tools. A more detailed description of each of the four articles included can be found below.

- 1) What are the effects of history length and age on mining software change impact? By Leon Moonen, Thomas Rolfsnes, Dave Binkley, and Stefano Di Alesio

Change impact analysis helps developers to understand which parts of the system are affected by a given change, and therefore helps to understand the impact of this change on the system. Traditionally, static or dynamic analysis techniques have been used to perform change

✉ Gabriele Bavota
gabriele.bavota@usi.ch

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

² Microsoft Corporation, Redmond, WA 98052, USA

impact analysis, but because static analysis tends to overestimate the impact set and dynamic analysis is very cost intensive, alternative approaches have been introduced that use evolutionary coupling. Evolutionary coupling occurs through software comments, commit messages, bug reports and many more traces left by the developer. A positive aspect of using evolutionary coupling for change impact analysis is also the language independence. The quality of the change impact analysis highly depends on the historical evolutionary couplings that are identified. In this article, the authors investigate the impact the length of the history (i.e., the number of transactions in the history), and the history age (i.e., the number of transactions since the patterns were mined) have on the performance of change impact analysis. The authors show that the history length impacts the performance, and that more transactions yield better results, even though the effects plateau at a certain number of commits. At the same time, history age significantly changes the performance of the change impact analysis and even little older patterns drastically decrease the quality of the outcome. Finally, the authors provide a prediction model to derive the optimal history length when using evolutionary coupling for change impact analysis.

- 2) The Need for Software Specific natural language techniques by David Binkley, Dawn Lawrie, Chris Morrell

Tools for information retrieval have been recently applied in the area of software engineering, e.g., to find code that implements some desired functionality within a software system. While the application of those tools showed some promising results, questions arose whether those IR tools which are originally designed for another domain, should be adjusted to the software engineering domain to obtain better results. The work in this article firstly presents a study that empirically investigates this question and shows that the information needs differ between those two domains. In particular, in the area of software engineering topic models are beneficial for information retrieval, whereby increased query length does not increase retrieval success. Those findings motivate the need for software specific information retrieval techniques and tools that are customized for the software engineering domain.

- 3) How effective are mutation testing tools? An empirical analysis of Java Mutation testing tools with manual analysis and real faults by Marinos Kintis, Mike Papadakis, Andreas Papadopoulos, Evangelos Valvis, Nicos Malevris, Yves Le Traon

In this article, the authors evaluate the strength and weaknesses of four Java based mutation testing tools (i.e., MAJOR, PIT, PITRV, MUJAVA). An important point of consideration for using mutation testing tools is obviously their performance (i.e., their fault finding capabilities). This is one of the aspects the authors investigate. In particular, the authors look at strength of the tools' fault detection capabilities by running them to detect a set of known faults as well as by manually investigating the mutants produced by the different tools. The achieved results show that the research version PIT_{RV} of the mutation testing tool PIT outperforms the three other tools in terms of fault finding capability (found 97%) as well as in the manual inspection of the produced mutants.

- 4) A comparison of code similarity analysers by Chaiyong Ragkhitwetsagul, Jens Krinke, David Clark

Often, blocks of code are reused several times in different places within a software system. Regularly, these blocks are not one to one copies but include modifications and alterations of the code. It is a common quality goal to remove those code duplications, as they are known to cause performance issues and introduce errors and faults. This article evaluates the performance of 30 similarity detection tools and techniques for finding code blocks that have undergone alterations. In particular, the authors look at the performance of detecting code blocks that have undergone either local or global modifications. The study shows that similarity analyzers specifically developed for source code outperform general tools, that the performance of the tools highly depends on their configuration, and that compilation of code is a good normalization technique.

These four articles represent a wide range of topics concerned with source code analysis which are appealing for both researchers and practitioners and are also a great example of the wide range of applications these techniques have.

Gabriele Bavota and Michaela Greiler, December 2017.

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 requests and providing such excellent work. We are also grateful for the continuous support by the Editorial board for the Journal of Empirical Software Engineering and by the Editors-in-Chief Robert Feldt and Tom Zimmermann.