Empir Software Eng
DOI 10.1007/s10664-011-9188-2

EDITORIAL

Introduction to the special issue on software repository mining in 2009

Michael Godfrey · Jim Whitehead

This special issue of Empirical Software Engineering consists of revised and extended versions of four selected papers originally presented at the 6th IEEE Working Conference on Mining Software Repositories (MSR 2009). The conference was held in Vancouver, Canada on May 16–17, and was co-located with the 2009 ACM/IEEE International Conference on Software Engineering (ICSE 2009). This conference brings together researchers who share an interest in advancing the science and practice of software engineering via the analysis of data stored in software repositories.

Software repository mining research emerged in the late 1990s in conjunction with the ready availability of software configuration management, mailing list, and bug tracking repositories from open source projects; the first Mining Software Repositories Workshop was held in 2004 in Edinburgh. In the past, researchers were able to access this kind of project data from only commercial projects, and with great difficulty. In contrast, open source projects typically provide free and open access to all of these artifacts via well-defined Internet-accessible interfaces. The availability of this data makes it possible for researchers to empirically explore a range of software engineering questions using software repository data as the primary source of information about activity within software projects. Some commonly explored areas include software evolution, models of the software development processes, characterization of developers and their activities, prediction of future software qualities, use of machine learning techniques on software project data, software bug prediction, analysis of software change patterns, and analysis of code clones. There has also been a stream of research on tools for software repository mining, and techniques for visualizing software repository data.

For those wishing to develop a broad understanding of the software repository mining research field, there are several resources. Tao Xie and Ahmed E. Hassan provide an overview of software repository mining research areas and methods in the tutorial notes to their Mining Software Engineering Data tutorial, given at several recent software

M. Godfrey (✉)
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
e-mail: migod@uwaterloo.ca

J. Whitehead
University of California, Santa Cruz, Santa Cruz, CA, USA

Published online: 10 December 2011

engineering conferences (notes are available at http://ase.csc.ncsu.edu/dmse/). Kagdi et al. provide a survey of software repository mining techniques in (Kagdi et al. 2007) that focuses on software evolution research, but which provides broad coverage of a wide range of research methods and repository types used in the field. Hassan provides a survey of the entire field along with future research challenges in (Hassan 2008). Together, these three sources give a strong general introduction to software repository mining research.

Due to its emphasis on research involving software repository data, papers at the Mining Software Repositories conference tend to take a quantitative empirical approach to exploring research questions. As a consequence, it is natural to select the best papers from this conference for inclusion in Empirical Software Engineering. The four papers in this special issue provide a good cross section of the topics and approaches explored in the software repository mining community.

The first two papers focus on predicting the location of defects in software projects. The first paper, "Time Variance and Defect Prediction in Software Projects," by Ekanayake et al., adds the time dimension into software defect prediction, exploring the variability of performance of a bug prediction model over time, using data from four open source projects. Within a project's history, there are periods of stability—where defect predictions work well—and instability, where they don't. A novel heat-map visualization makes this clear by providing an overview of the effectiveness of bug prediction over time. This recognition that defect prediction results are variable over time is an important one, since many defect prediction techniques have only been evaluated a single moment in a project's history. The paper also provides a "meta-model" that can be used to look forward and predict the quality of a defect prediction model over time.

In "On the use of calling structure information to improve fault prediction," Shin et al. explore the use of software call relationships in software fault prediction. This work is motivated by the observation that many software faults are interface faults, and hence adding call information to a fault prediction model would presumably increase prediction performance. The paper finds that adding call relationship information to an existing fault prediction model that does not involve this information yields a modest improvement in prediction accuracy.

In "Analyzing and Mining a Code Search Engine Usage Log", Bajracharya et al. provide an in-depth study of how users interact with an Internet source code search engine. By analyzing 12 months worth of usage data from Koders, an Internet search engine for source code, the authors were able to build usage profiles of typical users based on observed patterns of interaction. They found that the usage was "sparse" in the sense that while there were many users (over 10 million activities observed from over 3 million users), the vast majority used the tool only sparingly. They were also able to identify expert usage based on both frequency and usage patterns: those who used it most regularly had identifiable usage patterns that revealed they knew what information they were looking for and how to get it. The lessons learned from this study have been fed back into the development of the tool to aid both occasional users and experts.

In "Refining Code Ownership With Synchronous Changes", Hattori et al. seem to echo a quote from Marshal McLuhan: "We shape our tools, then our tools shape us". Through a detailed usage study, they found patterns of interaction by developers with version control systems (VCSs) that suggest that the usage of the tool is often shaped by some of the design characteristics of the tools, rather than the fundamental task at hand. By instrumenting an IDE and comparing the ongoing work to what is committed to the VCS, they found that developers will often seek to minimize the need for conflict resolution by avoiding committing changes that will require complicated merges. They examine the nature of code ownership as observed across three example systems, and discuss how it is shaped by VCSs.

We hope you enjoy the papers in this special issue.

References

Hassan A (2008) The road ahead for mining software repositories. In: *Frontiers of software maintenance*, held with the 2008 IEEE International Conference on Software Maintenance, Beijing, China, pp. 48–57.

Kagdi HH, Collard ML, Maletic JI (2007) A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J Software Mainten 19(2):77–131

Michael W. Godfrey is an associate professor in the David R. Cheriton School of Computer Science at the University of Waterloo, which he joined in 1998. Between 2000 and 2005, he held an Industrial Research Chair in telecommunications software engineering sponsored by Nortel Networks and the National Science and Engineering Research Council of Canada (NSERC). His research interests span many areas of software engineering including software evolution, empirical studies, reverse engineering, program comprehension, mining software repositories, and software clone detection and analysis.

Jim Whitehead is a professor of Computer Science at the University of California, Santa Cruz, where he directs the Software Introspection Laboratory. His research interests in software engineering focus on software bug prediction, understanding the nature of bugs, software evolution, and software design. He received his PhD in Information and Computer Science from the University of California, Irvine in 2000.