The Journal of Systems and Software 85 (2012) 2193–2194

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

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

Editorial

Guest editors' introduction to the special issue on automated software evolution

1. Introduction

Research in software evolution and evolvability has been thriving in the past few years, with a constant stream of new formalisms, tools, techniques, and development methodologies: on the one hand, the objective was to facilitate the way long-lived successful software systems can be changed in order to cope with demands from users and the increasing complexity and volatility of the contexts in which such systems operate; on the other hand, the research aim was to understand and if possible control the processes by which demand for these changes come about.

The theme of this special issue was "Automation in the context of software evolution", and papers were invited to present concepts, techniques or methodologies that are automated or amenable to automation. The theme of automation in software evolution is established in various fields: some software communities already benefit from the automation of a subset of central tasks: for example, programming teams are relying more and more on automated testing, when using test-driven development, especially regression testing, when a well-developed test suite of testing scripts is composed. Other communities have benefited for several years of measures to automatically detect quality of a requirements specification document, in order to write the right requirements from the early start of a project. Software maintenance and evolution present similar challenges: companies, governments, and Open Source communities spend a great deal of resources on a continual basis to fix, adapt, and enhance their software systems. The ability to evolve software rapidly and reliably; and the ability to automatically propose possible paths of evolution and enhancements, or discover maintenance activities, represent major challenges for software engineering.

We expected four types of submissions: articles dealing with the automatic parsing of data for the evolution and maintenance of software projects; articles describing state-of-the-art methods, models, and tools, supporting or improving the automation of software evolution and maintenance; empirical studies in the field, addressing one or many human, technical, social, and economic issues of how to automate software evolution through qualitative and/or quantitative analyses; finally, industrial experiences, including good practices and lessons learned on automating the activities of software evolution or maintenance in specific contexts or domains.

The call for papers of this special issue attracted 36 submissions from nearly 20 different countries covering many topics related to software evolution. Each paper was carefully evaluated by at least three experts in the field. After a rigorous peer review process, only 9 high quality research papers were selected for this issue.

2. Contributions

In the paper entitled "Using pig as a data preparation language for large-scale mining software repositories studies: an experience report", W. Shang, B. Adams and A.E. Hassan report on their experience in using a web platform to support mining software repositories (MSR) studies. They particularly focus on the automation of the data preparation phase, for which scalability constitutes an important issue. The paper presents three case studies that, while identifying some limitations, demonstrate the scalability and exibility of pig for extracting, transforming and loading software repository data. These results will certainly have a positive, concrete impact on the entire MSR research community, that develops novel techniques and tools to analyze large software (eco)systems, their development and their evolution.

The paper entitled "Towards automated traceability maintenance", by P. Mäder, O. Gotel, concerns the maintenance of traceability relations during the evolution of software systems. The authors present an approach for supporting the semi-automated update of traceability relations between requirements, analysis and design models of software systems expressed in UML. This approach is based on the analysis of change events captured during the use of a modeling tool and is supported by a prototype tool. An empirical study conducted with 16 students has shown that the proposed approach can save tedious and error-prone work.

The paper titled "Dependency solving: a separate concern in component evolution management" by S. Zacchiroli, P. Abate, R. Di Cosmo and R. Treinen, focuses on the issue of dependencies between large-grained components, and how to automatically remove the dependency links between them, when building and continuously updating the development of large software systems. In large Open Source collections of components, the interdependencies and the automatic solving of discrepancies between components face several challenges: first, because the packages are actually independent projects, developed and maintained by different developers and asynchronously combined into a single, tested and issue-free collection only at defined dates; second, because solving discrepancies among these individual projects (or components) has to take into consideration each component's version control repositories, and the component versions.

In their paper entitled "Identification and application of extract class refactorings in object-oriented systems", M. Fokaefs, N. Tsantalis, E. Stroulia and A. Chatzigeorgiou propose the use of a clustering algorithm to identify extract class refactoring opportunities in object-oriented systems. This algorithm identifies and ranks alternatives to split a class into different, more cohesive classes. The result of the refactoring is measured with entity placement

0164-1212/$ – see front matter
http://dx.doi.org/10.1016/j.jss.2012.05.074

metrics to evaluate the design improvement/degradation. An empirical study conducted with software developers and software quality professionals on three software systems indicates that the proposed approach is able to identify meaningful refactoring opportunities and improve the design quality. This work represents a worthy contribution for the refactoring field since in the literature, there is still a lack of studies where the authors actually check whether the refactorings suggested by the experimented techniques satisfy the software engineers' expectation.

"Model-driven support for product line evolution on feature level", by A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer and S. Kowalewski, focuses on product lines and how their evolution should be automatically supported. The paper presents an approach, called EvoFM based on two main features: the concept of "fragment", a group of related features in a feature model, and that are likely to change together; and the ability to describe the operations that change the fragments with dedicated model transformations, as "evolution operations". The approach is completed with a tool support based on an Eclipse plugin, and the validation is also performed based on the study of several versions of the Eclipse IDE itself.

The paper titled "Automated, highly-accurate, bug assignment using machine learning and tossing graphs" by P. Bhattacharya, I. Neamtiu and C.R. Shelton is a comprehensive study of the issue of bug triaging among software developers: who is responsible for a bug assignment? Can she toss such assignment to someone else? Can this process be automatically performed? The paper investigates various machine learners and techniques to provide a timely contribution in the field of bug triaging.

"On the relationship between comment update practices and software bugs", by W.M. Ibrahim, N. Bettenburg, B. Adams and A.E. Hassan, analyzes the link between the way source code comments are updated over time and the probability of bugs. A comment can indeed be consistently or inconsistently updated with respect to the associated source code. The authors study the comment update practices in three large open source software projects. Their results show that these practices, when they are themselves inconsistent over time, can better explain and predict the introduction of bugs than other indicators previously considered in the literature.

In order to help reduce the tedious and time consuming task of detecting and fixing bugs, in their paper entitled "Towards automated debugging in software evolution: evaluating delta debugging on real regression bugs from developers' perspectives", K. Yu, M. Lin, J. Chen and X. Zhang evaluate delta debugging in practical settings using real regressions taken from mid-sized open source systems. Delta debugging is an automated approach for isolating changes that introduced failures using the divide-and-conquer strategy. The authors conducted an empirical study to evaluate the costs and benefits of the application of delta debugging. They showed that two thirds of isolated changes in the systems analyzed provide direct or indirect clues in locating regressions.

"Preserving knowledge in software projects", by O.F. Alam, B. Adams and A.E. Hassan, focuses on the issue of knowledge preservation: with software projects growing in size and complexity, it becomes essential not only to decide which elements or subsystems are more important and central to preserve the overall knowledge of the projects, but also to prioritize the preservation of the changes on these elements, in case they have accumulated a long history of such changes. The authors define the concept of "Foundationality" of both subsystems and periods to identify, first, which subsystems are more central to other subsystems, and second the temporal periods when such foundational subsystems accumulated the most relevant changes. Albeit it becomes clear that the foundational subsystems are those that are accessed more frequently by others, open questions remain on how to derive a threshold to foundational periods, or how many of these periods should be kept in the knowledge preservation scheme of a software project.

3. Acknowledgments

We would like to thank all the authors for their high-quality submissions. We thank the anonymous reviewers for their constructive feedback that significantly helped the authors in reaching this level of quality. We warmly thank Hans van Vliet, editor-in-chief of JSS, for his trust and for his support all along the editorial process. We sincerely hope you will enjoy reading this special issue as much as we enjoyed guest editing it.

Andrea Capiluppi *
DISC, Brunel University, London, UK

Anthony Cleve
PReCISE Research Center, University of Namur, Namur, Belgium

Naouel Moha
Dèpartement d'informatique, Universitè du Quèbec à Montréal, Montréal, Canada

* Corresponding author.
E-mail addresses: andrea.capiluppi@brunel.ac.uk (A. Capiluppi), acl@info.fundp.ac.be (A. Cleve), moha.naouel@uqam.ca (N. Moha)

25 May 2012
Available online 9 June 2012