Empir Software Eng (2016) 21:1579–1585
DOI 10.1007/s10664-016-9439-3

EDITORIAL

Foreword to the special issue on empirical evidence on software product line engineering

Ebrahim Bagheri[1] · David Benavides[2] · Klaus Schmid[3] · Per Runeson[4]

Published online: 9 June 2016

The software product line engineering paradigm promotes sharing common core assets for building similar software systems. Researchers have argued that the deployment of product line techniques can lead to lower development costs as well as improved delivery time. There has also been some indication that, if systematically executed, products derived from a software product line can enjoy higher quality levels given repeatedly tested core assets are used in the development process (van der Linden et al. 2007). Researchers have explored many exciting research challenges at different levels of abstraction ranging from high-level metamodels for commonality and variability representation (Chen et al. 2009) to low-level pre-processor directives for incorporating variability in the code base (Liebig et al. 2010).

There is also a significant body of knowledge on experiences in practically transferring or deploying software product line research outcomes to industrial partners and real-world settings (van der Linden et al. 2007). However, recent systematic reviews of the literature reveals that there is space for more comprehensive and inclusive research that would provide empirical evidence on the practicality and applicability of research outcomes. For instance, Ahnassay et al. (2013) observed that within product line research published between 2006 and 2011 that had an empirical evaluation component only 34 % of the papers interacted or involved industrial practitioners. This can point to the need for more in-depth study of the practicality of the approaches that are proposed by the product line community that would involve the target audience of the research outcomes. This observation can be complemented with the fact that the majority of the papers did not provide any concrete foundation for their

✉ Per Runeson
per.runeson@cs.lth.se

1 Ryerson University, Toronto, Canada

2 University of Seville, Seville, Spain

3 University of Hildesheim, Hildesheim, Germany

4 Lund University, Lund, Sweden

problem statement. In other words, the problem definition was taken for granted and only the solution was described.

The study performed by Engström and Runeson (2011) also showed that research papers published between 2001 and 2008 with a focus on software product line testing mainly present the details of the proposed solution as opposed to reporting on the experience in deploying the solution or evaluating it. The study reported that 58 % of the papers focused on discussions of the conceptual proposal or descriptions of the solution, while 17 % of the studies were geared towards experience reports or evaluation research. The evidence from systematic literature reviews in software product lines indicate that more in-depth evidence needs to be gathered in order to allow for successful replication studies and perform knowledge and expertise transfer to industrial settings.

The main objective of this special issue is to add to the available body of knowledge consisting of systematic and in-depth studies on software product line research. The special issue consists of eight highly relevant papers, each of which went through a rigorous peer-review process. We believe that this special issue contains some exemplary work that include extensive empirical studies and can serve as a foundation for further advancing the field.

The first paper in this special issue entitled "*Assessing Software Product Line Potential*: *An Exploratory Industrial Case Study*" by Koziolek et al. provides valuable insight from the experiences of conducting a domain analysis on 20 industry-scale software systems in ABB, one of the largest corporations in Automation and Power Technologies. An interesting finding of this paper, among others, is that a lack of flexibility is often seen as a hindrance for the adoption of a product line approach and therefore its adoption is likely if long-term product stability is envisioned. This was shown in two cases where positive return on investment was only observed after 3 and 5 years, respectively. This article can be found in Issue 21:2, Pages 411–448 or http://link.springer.com/article/10.1007/s10664-014-9358-0.

In the next paper, Wang et al. provide the details of their software product line test case selection strategy in "*A Systematic Test Case Selection Methodology for Product Lines*: *Results and Insights from an Industrial Case Study*". The work is motivated by the authors' practical experience in working with Cisco's Video Conference System (Saturn). The paper clearly outlines how the different features of the seven variations of Saturn were modeled through the use of feature models and how test cases are automatically selected based on traceability between features and the existing test cases. An interesting lesson learnt from the experience was that very few people in the industrial setting of the study were familiar with the feature modeling terminology and notation; therefore, this indicates that more training and technology transfer activities could promote the use of product line techniques (DOI: 10.1007/s10664-014-9345-5).

The paper titled "*Preprocessor-Based Variability in Open-Source and Industrial Software Systems*: *An Empirical Study*" by Hunsen et al. focuses on the differences between the adoption of conditional compilation methods within open source and proprietary software applications. The authors have empirically studied whether the adoption pattern of the C preprocessor, one of the widely used conditional compilation methods, is any different between the two communities. To this end, the authors have collected the values of a set of structural metrics for 20 open-source and 7 industrial systems. The key finding of the study was that the adoption of C preprocessor is comparable in both communities and therefore, experiences already reported based on open-source software could be applicable to industrial software systems as well. This article can be found in Issue 21:2, Pages 449–482 or http://link.springer.com/article/10.1007/s10664-015-9360-1.

Myllärniemi et al. have studied purposeful performance variability in their work titled "*Performance Variability in Software Product Lines*: *Proposing Theories from a Case Study*". In their work, the authors employ grounded theory to iteratively collect and analyze data from a case study performed on Nokia's product line for the base station in the 3G radio access network. In order to be able to generalize the findings, the outcomes of the case study and the existing literature were contrasted and combined. An interesting observation reported in the paper is that performance variability does not necessarily need to be the outcome of resolving trade-offs or other forms of variability such as functional variability that are mainly solution domain causes. Other problem domain explanations such as customer requirements and the need for future upgrades can be reasons that can directly affect performance variability (DOI: DOI 10.1007/s10664-014-9359-z).

Sobernig et al. present their work on "*Quantifying Structural Attributes of System Decompositions in 28 Feature-oriented Software Product Lines – An Exploratory Study*" whereby they study a set of 28 feature-intensive applications developed in the Fuji programming language. The main objective of their work is to study whether the widely promoted idea that feature decomposition leads to increase in cohesion and decrease in coupling is empirically supported or not. The main finding of the authors is that in contrast to the general perception, feature decomposition can lead to feature units that have high coupling and at the same time they do not necessarily represent highly cohesive units of functionality (DOI: 10.1007/s10664-014-9336-6).

The paper by Asadi et al., "*The effects of visualization and interaction techniques on feature model configuration*", focuses on providing support for facilitating the feature model configuration process. The authors present and incorporate visualization as well as interaction interventions to assist application engineers in going through a staged configuration process. The interventions were empirically validated through controlled experiments with 20 graduate students. The results of the study show that visualization and interaction interventions have a statistically significant impact on application engineers' comprehension and change tasks. In addition, task time to completion also showed to be smaller when the visualization interventions were applied. The authors believe that feature model and configuration visualization techniques can improve application engineers' cognition and hence subsequently improve their performance (DOI: 10.1007/s10664-014-9353-5).

In their paper "*Coevolution of Variability Models and Related Software Artifacts*: *A Fresh Look at Evolution Patterns in the Linux Kernel*" Passos et al. focus on the co-evolution of variability models and other software artifacts to explore whether specific evolution patterns are observable that are correlated between variability models and software artifacts. The authors have explored the Linux Kernel, which consists of many configuration options to see whether the evolution of the kernel variability model has any correlation with changes on C and Makefiles. The findings are reported in the form of a coevolution pattern catalog that shows how and with what frequency certain changes impact different software artifacts (DOI: 10.1007/s10664-015-9364-x).

The final paper of this collection entitled "*Breathing Ontological Knowledge Into Feature Model Synthesis*: *An Empirical Study*" is by Bécan et al. The authors focus on developing an extractive technique that would identify and model software features from existing software repositories in the form a feature model. The challenging aspect of the work presented by the authors is to accurately identify hierarchical and group relationships between the identified features. To this end, six heuristics are presented that considered all logical, syntactical and semantic relationships between features' names. The finding is that the fully automated

Table 1 Feature-based description of the selected papers

	Problem Area	Research Method	Context	Scale
Koziolek et al.	Commonality and Variability Management	Case Study	Industrial	Large (20 industrial software systems)
Wang et al.	Testing	Survey	Industrial	Down Scaled (one industrial system)
Hunsen et al.	Code Implementation	Case Study	Industrial	Large (20 open-source and 7 industrial systems)
Myllärniemi et al.	Architecture	Case Study/Post mortem analysis	Industrial	Down Scaled (one industrial system)
Sobernig et al.	Feature Modeling	Survey/Case Study	Academic	Small (28 datasets)
Asadi et al.	Visualization	Controlled Experiment	Academic	Small (20 graduate students)
Passos et al.	Commonality and Variability Management	Case Study	Industrial	Large (Linux Kernel)
Bécan et al.	Feature Modeling	Case Study	Academic	Small (two academic sources including SPLOT)

generation of a feature model from source code can be highly inaccurate; therefore, the authors provide insight into a hybrid approach for using various collection of heuristics for improving certain performance measures such as accuracy and completeness (DOI: 10.1007/s10664-014-9357-1).

Table 1 offers a feature-based description of the selected papers in this collection. Each paper is described using four features adopted from (Alvin et al. 2013). The first column describes the specific aspect of software product lines that is addressed by the paper. The second and third columns specify the research method that was used in the paper and whether the study was performed in an academic or industrial setting. The last column shows the scale of the subjects/objects that were employed during experimentation.

Finally, we would like to graciously acknowledge the rigor and dedication of the reviewers of this special issue, without whom this valuable collection would not be completed. We are also thankful to the Empirical Software Engineering Editors in Chief, Lionel Briand and Thomas Zimmermann, for their support, help and insight throughout the process of preparing this special issue.

References

Ahnassay A, Bagheri E, Gasevic D (2013) Empirical evaluation in software product line engineering, Technical report, Laboratory for systems, Software and Semantics, Ryerson University, TR-LS3-130084R4T. http://ls3.rnet.ryerson.ca/wp-content/uploads/2013/08/TR-LS3-130084R4T.pdf

Chen L, Ali Babar M, Ali N (2009) Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference* (SPLC '09). Carnegie Mellon University, Pittsburgh, p 81–90

Engström E, Runeson P (2011) Software product line testing - A systematic mapping study. Inf Softw Technol 53(1):2–13

Liebig J, Apel S, Lengauer C, Kästner C, Schulze M (2010) An analysis of the variability in forty preprocessor-based software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1* (ICSE '10), Vol. 1. ACM, New York, p 105–114

van der Linden FJ, Schmid K, Rommes E (2007) Software product lines in action: the best industrial practice in product line engineering. Springer-Verlag New York, Inc., Secaucus

Ebrahim Bagheri is a Canada Research Chair in Software and Semantic Computing, Associate Professor, and the Director of the Laboratory for Systems, Software and Semantics (LS3) at Ryerson University. He has been active in the areas of semantic web and software engineering, and, for the past several years, Dr. Bagheri's research has focused on devising empirically tested methods that enhance the systematic large-scale reuse of software assets with a focus on the software product line engineering paradigm.

David Benavides is an Associate Professor at the University of Seville (Spain), in which he obtained his Ph.D. in 2007. His main areas of expertise are Software Product Lines and Feature Model Analysis. He is also interested in open source and social applications of software systems. He has visited different research centers and has co-authored papers with more than 40 external authors along the world and has special connections with South America.

Klaus Schmid is Professor of Software Engineering at the University of Hildesheim (Germany), where he leads the software systems engineering group (SSE). His main research interests are in software product lines, adaptive systems and requirements engineering. He authored about 100 publications in these and related areas.

He previously worked with Fraunhofer and was a research fellow with the Software Engineering Center (SEC), SK and a visiting researcher at the Software Engineering Institute (SEI), USA. He also consulted for various large (and small) companies. He served in numerous roles in the scientific community and is currently Chair of the steering committee of the International Software Product Line Conference (SPLC). He received a diploma and a Ph.D. from the University of Kaiserslautern, Germany.

Per Runeson is a Professor of Software Engineering at Lund University (Sweden), head of the Department of Computer Science, and the leader of its Software Engineering Research Group (SERG) and the Industrial Excellence Center on Embedded Applications Software Engineering (EASE). His research interests include empirical research on software development and management methods, in particular for verification and validation. He is the principal author of "Case study research in software engineering", has coauthored "Experimentation in software engineering", serves on the editorial board of Empirical Software Engineering and Software Testing, Verification and Reliability, and is a member of several program committees.