Empir Software Eng (2016) 21:749–752
DOI 10.1007/s10664-016-9433-9

EDITORIAL

Guest editorial: special section on software reverse engineering

Romain Robbes[1] · Rocco Oliveto[2] · Massimiliano Di Penta[3]

Published online: 26 April 2016

As defined in a seminal taxonomy by Chikofsky and Cross (Chikofsky and Cross II 1990) software reverse engineering is:

> *"the process of analyzing a subject system to (i) identify the systems components and their inter-relationships and (ii) create representations of the system in another form or at a higher level of abstraction."*

Such an analysis process pertains to artifacts that can be at an extremely low-level, e.g., binary files or system execution traces, or at intermediate-level, e.g., such as source code, patches, and defects, and the history and discussions associated to these artifacts.

Software reverse engineering is a mature research field with high practical relevance: often, the only way to get an understanding of a large and complex software system is through these lower-level artifacts, especially when higher-level artifacts are absent or

✉ Romain Robbes
rrobbes@dcc.uchile.cl

Rocco Oliveto
rocco.oliveto@unimol.it

Massimiliano Di Penta
dipenta@unisannio.it

[1] Department of Computer Science (DCC), University of Chile, Santiago, Chile

[2] Department of Bioscience and Territory, University of Molise, Pesche, IS, Italy

[3] Department of Engineering, University of Sannio, Benevento, Italy

outdated. In general, almost every time one has to evolve an existing software system—and especially when source code or binary is the only reliable source of documentation—reverse engineering becomes a key element of software evolution (Canfora and Di Penta 2007). Indeed, in such cases reverse engineering is highly recommended by the IEEE Std. 1219 (IEEE 1999) for software maintenance.

This special section features six papers on the topic of reverse engineering, using an array of techniques and data sources: some use very low-level artifacts, while some use higher-level ones and historical information.

The first article from this special section, *"On the detection of custom memory allocators in C binaries"* (Chen et al. 2016), focuses on binary analysis. The article describes a technique to detect custom memory allocators and deallocators, which is vital to properly detect and track data structures in performance-critical applications. The tool implementing the technique, MemBrush, has been evaluated on a large number of real-world applications, and was found to have a high accuracy. MemBrush can then transfer that data to existing reverse engineering tools.

The article *"Scalable data structure detection and classification for C/C++ binaries"* (Haller et al. 2016) also deals with binary analysis. The tool presented in the article, *MemPick*, analyzes the links between the objects in memory in order to detect higher-level data structures. *MemPick* can detect several commonly used data structures, including several types of linked lists, trees, or graphs. The tool was evaluated on 30 different systems.

In *"Inferring extended finite state machine models from software executions"* (Walkinshaw et al. 2016), the focus shifts to execution traces. More specifically, the goal of the article is to infer Extented Finite State Machines (EFSMs) from the excution traces. EFSMs can blend the control and data aspects of the software system under study. The technique presented to infer EFSM is based on machine learning, and is evaluated quantitatively and qualitatively on three software systems.

The remainder of the special section, starting with *"Mining architectural violations from version history"* (Maffort et al. 2016), exploits higher-level information. The article combines static and historical analysis in the context of checking the architectural conformance of a software system. Since detecting architectural violations in this way is a challenging problem, the article also proposes an iterative process for experts to verify the conformance semi-automatically.

In the article *"Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults"* (Jaafar et al. 2016), the focus is on classes depending on design pattern (and anti-pattern) elements, under the assumption that the good and/or bad characteritics of patterns and anti-patterns may propagate to their dependencies. The article puts that assumption to the test by analysing the fault-proneness and change-proneness of dependencies of six design patterns and ten anti-patterns, in the sequence of releases of three software systems.

Finally, this special section concludes with *"Investigating technical and non-technical factors influencing modern code review"* (Baysal et al. 2016), in which the code review process of two large open-source projects is studied, in order to better understand the factors that might result in whether the code submissions are evaluated in a timely manner or not. Reverse engineering techniques are applied in order to reconstruct the patch submission process from the information in the issue tracking and code review systems.

These six articles exemplify the breath, depth and quality of the research performed in the field of software reverse engineering.

References

Baysal O, Kononenko O, Holmes R, Godfrey M (2016) Investigating technical and non-technical factors influencing modern code review. Empir Softw Eng. doi:10.1007/s10664-015-9366-8

Canfora G, Di Penta M (2007) New frontiers of reverse engineering. In: International Conference on Software Engineering, ICSE 2007, Workshop on the Future of Software Engineering, FOSE 2007, pp 326–341

Chen X, Slowinska A, Bos H (2016) On the detection of custom memory allocators in C binaries. Empir Softw Eng. doi:10.1007/s10664-015-9362-z

Chikofsky EJ, Cross II JH (1990) Reverse engineering and design recovery: a taxonomy. IEEE Softw 7(1):13–17

Haller I, Slowinska A, Bos H (2016) Scalable data structure detection and classification for C/C++ binaries. Empir Softw Eng. doi:10.1007/s10664-015-9363-y

IEEE (1999) IEEE Standard for Software Maintenance, IEEE Std 1219-1998, vol 2. IEEE Press

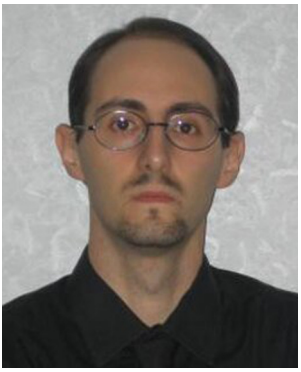
Jaafar F, Guéhéneuc YG, Hamel S, Khomh F, Zulkernine M (2016) Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. Empir Softw Eng. doi:10.1007/s10664-015-9361-0

Maffort C, Valente M, Terra R, Bigonha M, Anquetil N, Hora A (2016) Mining architectural violations from version history. Empir Softw Eng. doi:10.1007/s10664-014-9348-2

Walkinshaw N, Taylor R, Derrick J (2016) Inferring extended finite state machine models from software executions. Empir Softw Eng. doi:10.1007/s10664-015-9367-7

Romain Robbes is an Associate Professor at the University of Chile (Computer Science Department), in the PLEIAD research lab, where he works since January 2010. He earned his PhD in 2008 from the University of Lugano, Switzerland and received his Master's degree from the University of Caen, France. His research interests lie in Empirical Software Engineering, including, but not limited to, Mining Software Repositories. He authored more than 60 papers on these topics, including top software engineering and programming languages venues such as ICSE, FSE, ASE, EMSE, ECOOP, or OOPSLA, received best paper awards at WCRE 2009 and MSR 2011, and was the recipient of a Microsoft SEIF award 2011. He has served in the organizing and program committees of many software engineering conferences (ICSE, MSR, WCRE, ICSME, OOPSLA, ECOOP, IWPSE, and others) and serves on the Editorial Board of EMSE.

Rocco Oliveto is an associate professor in the Department of Bioscience and Territory at University of Molise, Italy. He is the chair of the Computer Science Program and the director of the Laboratory of Computer Science and Scientific Computation of the University of Molise. He received the PhD in Computer Science from University of Salerno, Italy, in 2008. His research interests include traceability management, information retrieval, software maintenance and evolution, search-based software engineering, and empirical software engineering. He is an author of about 100 papers appeared in international journals, conferences and workshops. He serves and has served as organizing and program committee member of international conferences in the field of software engineering. He is a member of the IEEE Computer Society, IEEE, and ACM.



Massimiliano Di Penta is an associate professor at the University of Sannio, Italy. His research interests include software maintenance and evolution, mining software repositories, empirical software engineering, search-based software engineering, and service-centric software engineering. He is an author of more than 190 papers appeared in international journals, conferences and workshops. He serves and has served in the organizing and program committees of more than 100 conferences such as ICSE, FSE, ASE, ICSM, ICPC, GECCO, MSR WCRE, and others. He has been a general co-chair of various events, including the 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM 2010), the second International Symposium on Search-Based Software Engineering (SSBSE 2010), and the 15th Working Conference on Reverse Engineering (WCRE 2008). Also, he has been program chair of events such as the 28th IEEE International Conference on Software Maintenance (ICSM 2012), the 21st IEEE International Conference on Program Comprehension (ICPC 2013), the ninth and 10th Working Conference on Mining Software Repository (MSR 2013 and 2012), the 13th and 14th Working Conference on Reverse Engineering (WCRE 2006 and 2007), the first International Symposium on Search-Based Software Engineering (SSBSE 2009), and other workshops. He is currently member of the steering committee of ICSME, MSR, SSBSE, and PROMISE. Previously, he has been a steering committee member of other conferences, including ICPC, SCAM, and WCRE. He is in the editorial board of the IEEE Transactions on Software Engineering, the Empirical Software Engineering Journal edited by Springer, and of the Journal of Software: Evolution and Processes edited by Wiley. He is a member of the IEEE.