Empir Software Eng (2015) 20:410–412
DOI 10.1007/s10664-015-9382-8

EDITORIAL

# Guest editorial: special section on software maintenance and evolution

**Massimiliano Di Penta[1] · Jonathan I. Maletic[2]**

Published online: 29 March 2015
© Springer Science+Business Media New York 2015

As described in the IEEE Standard Glossary of Software Engineering Terminology (1994), software maintenance is *"The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment."*

Since the first studies of software maintenance and evolution, (Lehman 1980; Lehman and Belady 1985), research in the area has covered a wide variety of topics. Topics range from reverse engineering, program comprehension, software migration, refactoring, and regression testing. As such the five articles in this special issue cover differ aspects of software maintenance and evolution, namely traceability link recovery and usage, understanding of issue reports, and identification/refactoring of code smells specifically related to spreadsheets.

In the paper *"Do Developers Benefit From Requirements Traceability When Evolving and Maintaining a Software System?"* the authors conduct a controlled experiment with 71 subjects to investigate whether the availability of traceability links helps developers to achieve higher productivity and correctness during maintenance tasks. Results of their study indicate that, when traceability links are available, developers perform their tasks 24% faster and with 50 % more correct solutions. This paper shows that traceability links can indeed be useful, however in practice these links are often unavailable or outdated. This fact led to the development of techniques, often based on Information Retrieval (IR) techniques, to help recover links (Antoniol et al. 2002; Maletic and Marcus 2001; Asuncion et al. 2010).

✉ Massimiliano Di Penta
dipenta@unisannio.it

Jonathan I. Maletic
jmaletic@kent.edu

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

[2] Department of Computer Science, Kent State University, Kent, OH, USA

This leads to the next paper *"An Empirical Study on the Importance of Source Code Entities for Requirements Traceability*. It investigates, using eye-tracking technology, what types of code elements developers visually examine when verifying traceability links recovered using IR-based techniques. They use this empirically collected information as a basis to fine tune the weights (TF/IDF) in the underlying IR technique. The results show that this helps to improve the accuracy of traceability link recovery.

Among approaches to recover traceability links, there has been recent work that uses topic models (Asuncion et al. 2010) and techniques such as Latent Dirichlet Allocation (LDA) (Blei et al. 2003). In the paper *"Do Topics Make Sense to Managers and Developers?"* the authors make use of topic models and investigate how professional and open source developers consider the relevancy of topics extracted from issue reports and how such topics reflect activities mined from version control systems.

In the paper *"Modeling the 'Hurried' Bug Report Reading Process to Summarize Bug Reports"* the authors propose an unsupervised approach for bug report summarization. The approach attempts to select sentences that deal with frequently discussed topics, while at the same time are close to the bug report title and description. The results show the superiority of the proposed approach with respect to those previously proposed. The authors also give evidence that the generated summaries are considered useful by developers.

One factor that often impacts software maintainability is the presence of code smells, i.e., symptoms of poor design or implementation choices (Fowler 1999). In their paper *"Detecting and Refactoring Code Smells in Spreadsheet Formulas"* the authors define rules to identify smells in spreadsheets and propose visualization and refactoring techniques for such smells. They evaluate the approach over three case studies and also collect feedback from the spreadsheets' owners.

## References

Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E (2002) Recovering traceability links between code and documentation. IEEE Trans Softw Eng 28(10):970–983

Asuncion HU, Asuncion AU, Taylor RN (2010) Software traceability with topic modeling. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), pp. 95–104, ACM, Cape Town, South Africa, May 2-8, 2010

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022. doi:10.1162/jmlr.2003.3.4-5.993

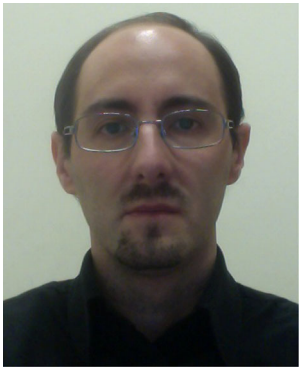
Fowler M (1999) Refactoring: improving the design of existing code. Addison-Wesley, MA

IEEE Standard Glossary of Software Engineering Terminology (1994) IEEE Std 610.12-1990 (1991 Corrected Edition), The Institute of Electrical and Electronics Engineers, Inc.

Lehman MM (1980) Programs, life cycles and laws of software evolution. Proc. IEEE 68(9):1060–1076

Lehman MM, Belady LA (1985) Software evolution - processes of software change. Academic Press London, p 538

Maletic JI, Marcus A (2001) Supporting program comprehension using semantic and structural information. In: Proceedings of 23rd International Conference on Software Engineering. IEEE CS Press, Toronto, Ontario, Canada, pp 103–112



**Massimiliano Di Penta** is 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 testing. He is author of over 200 papers appeared in international journals, conferences and workshops. He serves and has served in the organizing and program committees of over 100 conferences such as ICSE, FSE, ASE, ICSM, ICPC, GECCO, MSR WCRE, and others. He is currently member of the steering committee of ICSME, MSR, SSBSE, and PROMISE. Previously, he has been steering committee member of other conferences, including ICPC, SCAM, and WCRE. He is in the editorial board of 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.

**Jonathan I. Maletic** is a Professor in the Department of Computer Science at Kent State University in Ohio, USA. His research interests are centered on software evolution and he has authored over 100 refereed publications in the areas of analysis, manipulation, transformation, comprehension, traceability, and visualization of software. The National Science Foundation has awarded Prof. Maletics several research grants, including an award to support the srcML project. He received the PhD and MS in Computer Science from Wayne State University and the BS in Computer Science from The University of MichiganFlint.