Guest editorial: special issue on selected topics in automated software engineering Specification mining and defect detection

Mats P.E. Heimdahl · Gabriele Taentzer

Received: 21 April 2011 / Accepted: 25 April 2011 / Published online: 11 May 2011
© Springer Science+Business Media, LLC 2011

The research field of automated software engineering has its genesis in the intersection of artificial intelligence and software engineering research; the conference series that led to the current Automated Software Engineering (ASE) conference started as Knowledge Based Software Assistant (KBSA) in 1986, changed name to Knowledge Based Software Engineering (KBSE) in 1991, and—recognizing the expansion of the field—changed name and broadened the scope to Automated Software Engineering (ASE) in 1997. This field is one of the largest and most vibrant in software engineering research. Significant research results in this field have been presented in the Automated Software Engineering Journal since 1994.

The papers in this issue of the ASE Journal have been selected to represent outstanding work in recent research directions in automated software engineering.

The first three papers are all related to data mining. In “Inferring Specifications for Resources from Natural Language API Documentation” Zhong, Zhang, Xie, and Mei are applying advances in data mining techniques to infer API interface specifications from API documentation written in natural language and use this information to detect inconsistencies between documented API usage and actual usage in the implementations. They illustrate their Doc2Spec on selected open source projects and show that it can detect real defects in existing projects.

In “Mining Temporal Specifications from Object Usage” Wasylkowski and Zeller are also approaching a mining problem but they use static analysis and model check-

M.P.E. Heimdahl (✉)
University of Minnesota, Minneapolis, MN, USA
e-mail: heimdahl@cs.umn.edu

G. Taentzer
Philipps-Universität Marburg, Marburg, Germany
e-mail: taentzer@informatik.uni-marburg.de

ing techniques to learn temporal logic specifications from code examples. Such specifications can then be effectively used to check code for their violation.

Static or dynamic defect-detection tools can accept programming rules as input and detect their violations in software as defects. As these rules are often not specified, approaches have been developed to mine programming rules as frequent patterns from program source code. In “Mining Alternative Patterns for Defect Detection”, Thummalapenta and Xie present a new approach, called Alattin, that mines patterns in four pattern formats—conjunctive, disjunctive, exclusive-disjunctive, as well as combinations of these patterns—and illustrate how this approach improves the accuracy of the analysis.

Deshmukh, Emerson, and Sankaranarayanan illustrate how verification techniques have made inroads in the automated software engineering field. Given that much implementation detail is hidden in object-oriented concurrent libraries, it is quite easy to cause thread safety violation by invoking methods in an unsafe manner. In “Symbolic Modular Deadlock Analysis” they address this issue through a technique for inferring interface contracts that specify permissible concurrent method calls. In this work, the focus has been on guaranteeing deadlock-free execution for the methods in the library. They have developed a prototype tool illustrating both scalability and efficacy of this approach.

Finally, Han, Wallace, and Miller return to the roots of ASE with a programmer’s assistant in “Code Completion Of Multiple Keywords From Abbreviated Input”. Conventional code completion finds one keyword at a time based on an exact character match. Abbreviation completion improves the efficiency of code-writing by supporting code completion of multiple keywords based on non-predefined abbreviated input; a technique that does not require the user to memorize abbreviations and provides incremental feedback of the most likely completions. A user study of their system found 30% reduction in time usage and 41% reduction of keystrokes over conventional code completion.

We hope that the papers in this issue will generate interest in automated software engineering and encourage your participation in the ASE Journal as well as the ASE Conference series.

Acknowledgements We would like to thank the Editor in Chief of the ASE Journal, Robert J. Hall, for encouraging us to assemble this special issue. Of course, this special issue would not have been possible without the outstanding work by the many reviewers involved in selecting the papers.