Editorial: selected topics in ASE

Robert J. Hall

Received: 14 May 2012 / Accepted: 17 May 2012 / Published online: 31 May 2012
© Springer Science+Business Media, LLC 2012

In addition to publishing the high quality manuscripts that flow in to the Journal on a regular basis, I often become aware of really interesting and high quality research that, for any of a number of reasons, has not “come over the transom” to the Journal. In what I hope to be a regular, yet not necessarily annual, series of special issues, I plan to invite the researchers working in these areas to submit archival quality contributions describing their work. This special issue is the first in this series.

- Automation can benefit just about any activity in a software engineer’s professional life. In “The Design and Evaluation of a Tool to Support Software Designers at the Whiteboard”, Mangano and van der Hoek show how this is true even in the early and informal stages of design, when people draw lots of boxes and arrows, erase things, and wave their hands.
- Determining what code does is an important and daily activity during maintenance. Mining specification information from program structure and behavior promises to help engineers with this activity by speeding it up and lessening the tendency toward errors. In “Scenario-Based and Value-Based Specification Mining: Better Together”, Lo and Maoz show how putting together two alternative approaches they have explored previously yields better results than either approach itself.
- Determining differences between versions of code is both an important and almost daily task in software maintenance, but it is also difficult. Text-oriented differencers, like the Unix tool “diff”, take a rather shallow syntactic approach to the problem. When the artifacts represented by the source have inherently concurrent meanings, a useful differencing tool must avoid meaningless syntactic similarity, such as mere proximity in the source file, and provide the user with more semantically meaningful differences. In “Vdiff: A Program Differencing Algorithm for

R.J. Hall (✉)
AT&T Labs Research, Florham Park, NJ 07932, USA
e-mail: Bob.ASEJ@gmail.com

Verilog Hardware Description Language”, Duley, Spandikow, and M. Kim describe their tool for improved differencing of hardware descriptions. Modern hardware, of course, is inherently concurrent and of complexity comparable with concurrent software, so this interesting work should provide transferable lessons for software differencing tools in general.

- While modern programming languages like Java provide programmers with convenient tools for coding thread-level concurrency, they are not that helpful in avoiding bugs relating to concurrency in general, and memory-model specific semantics in particular. In previous work, these authors described a tool for helping find such bugs; in this work, “JRF-E: Using Model Checking to give Advice on Eliminating Memory Model-related Bugs”, K. Kim, Yahvuz-Kavecı, and Sanders extend the ideas of that paper into a tool for giving advice to the user in eliminating the bugs found.
- Program analyses increasingly need to be able to reason accurately about strings. However, the theory of strings is complex and can lead to large search spaces. In “STRSOLVE: Solving String Constraints Lazily”, Hooimeijer and Weimer show how to improve efficiency, accuracy, and likelihood of completion by avoiding the *a priori* construction of the search space and only materializing parts lazily, i.e. when they are needed during the constraint solving.

Please enjoy these contributions and feel free to email the authors with any specific questions. You are invited, as always, to email me at Bob.ASEJ@gmail.com with your thoughts on this issue, this approach to special issues, or the Journal in general.