Software Qual J (2011) 19:3–4
DOI 10.1007/s11219-010-9119-2

# Guest editor's introduction to the special section on the 2009 international conference on program comprehension (ICPC 2009)

Rainer Koschke · Andrian Marcus · Gerald C. Gannod

Published online: 30 November 2010

Program comprehension is a vital blend of software engineering activities that support reuse, inspection, maintenance, evolution, migration, reverse engineering, and reengineering of existing software systems. The International Conference on Program Comprehension (ICPC) is the principal venue for work in the area of program comprehension as well as a leading venue for work in the areas of software analysis, reverse engineering, software evolution, and software visualization. ICPC 2009 took place during May 17–19, 2009, in Vancouver, British Columbia, Canada, co-located with the International Conference on Software Engineering (ICSE '09).

ICPC 2009 received a record number of technical paper submissions (74), which allowed us to assemble an excellent program that continues ICPC's tradition of providing a high-quality venue for sharing the latest advances in program comprehension. The program included 20 full research papers and 14 short papers. Of these 20 full research papers, five were extended beyond their conference versions and submitted to the standard SQJ review process involving three anonymous reviewers per paper. Three of these invited papers passed the review process and are contained in this special section.

The paper entitled "Resumption Strategies for Interrupted Programming Tasks" by Chris J. Parnin and Spencer Rugaber deals with the reality of programming where programmers are often interrupted by phone calls, colleagues, and other external events. After the sudden interruption, programmers must resume their task. This paper studies the frequency and persistence of interruptions and the various strategies programmers use in their task resumption. An exploratory analysis was performed on 10,000 recorded sessions of 86 programmers and a survey of 414 programmers. Based on the analysis, the authors propose

R. Koschke (✉)
University of Bremen, Bremen, Germany
e-mail: koschke@tzi.de

A. Marcus
Wayne State University, Detroit, MI, USA
e-mail: amarcus@wayne.edu

G. C. Gannod
Miami University, Oxford, OH, USA
e-mail: gannodg@muohio.edu

a framework for understanding resumption strategies and suggest how task resumption might be better supported in future development tools.

The paper entitled “Automatically Identifying Changes that Impact Code-to-Design Traceability during Evolution” by Maen Hammad, Michael Collard, and Jonathan Maletic presents an approach to automatically determine whether a given source code change impacts the design (i.e., UML class diagram) of the system using lightweight analysis and syntactic differencing of the source code changes. This allows code-to-design traceability to be consistently maintained as the source code evolves. The tool implementing the approach is evaluated and compared against manual inspection by human experts. The authors’ study shows that the tool performs better than manual inspection. Moreover, the study shows that most of the code changes do not impact the design. If changes do impact the design, they are generally larger in terms of number of changed files and lines of code. The results also reveal that most bug fixes do not impact design.

The paper entitled “Empirical Studies on Programming Language Stimuli” by Andreas Stefik and Ed Gellenbeck presents Sodbeans, an integrated development environment designed to output carefully chosen spoken auditory cues to supplement empirically evaluated visual stimuli. Originally designed for the blind, earlier work suggested that Sodbeans may benefit sighted programmers as well. The authors evaluate Sodbeans in two studies. In the first experiment, they compare a visual debugger, an auditory debugger, and a multimedia debugger, which includes both visual and auditory stimuli. The results indicate that while auditory debuggers on their own are significantly less effective for sighted users when compared with visual and multimedia debuggers, multimedia debuggers might benefit sighted programmers under certain circumstances. In the second pilot survey, the authors analyze individual elements in a custom programming language to garner empirical metrics on their comprehensibility. Results showed that some of the most widely used syntax and semantics choices in commercial programming languages are extraordinarily unintuitive for novices.

We thank the authors for providing these contributions, the anonymous reviewers for their detailed and constructive comments, and Rachel Harrison, the Editor in Chief of the Software Quality Journal and the editorial staff for their support.