

Introduction to the Special Section on the ACM SIGSOFT Foundations of Software Engineering Conference

Prem Devanbu and Michal Young, *Member, IEEE Computer Society*

THE 14th ACM SIGSOFT Foundations of Software Engineering Conference (SIGSOFT FSE-14) was held in Portland, Oregon, during November 2006. Continuing the tradition of FSE, this conference was a single-track conference, providing an intimate, lively, and intensive venue for exchange of the best current academic research ideas in software engineering. SIGSOFT FSE-14 was a highly competitive and prestigious conference, with 25 accepted papers drawn from a pool of 125 high-quality submissions after a rigorous peer review process. Each paper received at least three reviews. Papers passing a quality threshold in the reviews were presented by reviewers and discussed at a face-to-face meeting of the full technical program committee and final decisions for acceptance were made by the committee as a whole.

During the review process, program committee members were asked to identify particularly innovative and well-presented papers for inclusion in a special section of the *IEEE Transactions on Software Engineering*. Conference chairs solicited arguments in favor and against the nominated papers and, after careful consideration, invited the authors of a small handful of papers to submit revised and extended versions of their FSE-14 papers for the special section. These submissions were then rigorously rereviewed prior to acceptance for this issue.

The first paper in this issue is concerned with the critical and complex task of program understanding. A significant chunk of the software maintainers' burden is the need to understand the code that s/he is asked to change. Anyone who has suffered through the task of understanding someone else's code will appreciate the careful study described by Sillito, Murphy, and De Volder in "Asking and Answering Questions during a Programming Change Task." The authors conducted a qualitative study of programmers (both graduate students and industrial programmers) engaged in program comprehension subtasks while maintaining software. They categorize these subtasks hierarchically, describing how programmers begin with

islands of perception within the code and expand to increasingly higher levels of conceptual structure. This study is comprehensive, describing 44 different types of questions. We expect that this careful and detailed analysis will influence both further empirical studies of program understanding and the design of software development environments.

An important thread of software testing research is devising ways to exploit a variety of design representations in selecting and assessing test suites. In the second paper, Rutherford, Carzaniga, and Wolf expand the class of representations that serve as "models" or "specifications" for model-based or specification-based testing. They propose that, for distributed systems, discrete event simulation models serve as executable specifications that are particularly suitable for selecting test suites. Among their advantages are that simulation models of distributed systems are widely used and maintained in practice and they include a variety of environmental characteristics that system designers have considered potentially significant. These authors show that simulation models can be used not only to define test adequacy criteria (in a fashion analogous to other specification-based and model-based testing techniques), but also to estimate their effectiveness in detecting actual software faults by measuring their effectiveness in detecting seeded faults in the simulation models. We expect "Evaluating Test Suites and Adequacy Criteria Using Simulation-Based Models of Distributed Systems" to have a near-term impact on the practice of testing distributed systems and a longer-term influence on software testing research.

Prem Devanbu received the BTech degree in electrical engineering from the Indian Institute of Technology, Chennai, and the PhD degree from Rutgers University in computer science. He is a professor of computer science at the University of California, Davis. He is also an adjunct associate professor of software engineering at Chiang Mai University, Thailand. He is a member of ACM SIGSOFT and the Explorit Children's Science Museum in Davis, California. His research interests include software tools and empirical software engineering.

Michal Young received the BA degree in computer and information science from the University of Oregon and the MS and PhD degrees in information and computer science from the University of California, Irvine. He is an associate professor of computer science at the University of Oregon. He is a member of ACM SIGSOFT and the IEEE Computer Society. His research interests include concurrency and software testing and analysis, and revolve around the relation between the structure of software systems and the structure of correctness arguments.

• P. Devanbu is with the Department of Computer Science, Kemper Hall, University of California, Davis, Davis, CA 95616.

E-mail: devanbu@cs.ucdavis.edu.

• M. Young is with the Department of Computer and Information Science, 120 Deschutes Hall, University of Oregon, Eugene, OR 97403-1202.

E-mail: michal@cs.uoregon.edu.

For information on obtaining reprints of this article, please send e-mail to: tse@computer.org.