JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2007; **19**:203–204
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr.352

Editorial

Special Issue on Source Code Analysis and Manipulation (SCAM 2006)

This special issue features extended versions of selected papers from the 6th International Workshop on Source Code Analysis and Manipulation (SCAM 2006) that took place in Philadelphia, PA, U.S.A. on 27–29 September 2006, co-located with the 22nd IEEE International Conference on Software Maintenance (ICSM 2006).

The aim of SCAM is to bring together researchers and practitioners working on theory, techniques and applications, concerning analysis and/or manipulation of the source code of computer systems. While much attention in the wider software engineering community is properly directed toward other aspects of systems' development and evolution, such as specification, design and requirements engineering, it is the source code that contains the only precise description of the behavior of the system. The analysis and manipulation of source code thus remain pressing concerns. Whereas several conferences and workshops address the applications of source code analysis and manipulation, the aim of SCAM is to focus on the algorithms and tools themselves—what they can achieve; and how they can be improved, refined and combined.

Held for the first time with ICSM 2001 in Florence, SCAM has been a very successful event for the last six years, with a continuously increasing attendance and number of submissions; such that it has been transformed, starting from 2007, to a working conference. Over the years, SCAM has grown its unique format, consisting of a two-day event filled with technical sessions that have plenty of time allocated to discussions. Each technical session is structured around three short presentations (15 min) followed by 45 min open discussion that is initiated by controversial questions and issues raised by the authors of the papers presented. All attendees are encouraged to write their ideas and comments on transparencies, rather than merely contributing verbally. These transparencies are then collected and scanned for publication on the SCAM Web site*.

In 2006, out of 48 submissions, 20 excellent papers were selected to be presented at the workshop, together with an inspiring keynote on 'Slicing concurrent Java programs with Indus' by John Hatcliff from Kansas State University. The accepted papers covered a broad range of topics in source code analysis and manipulation, such as slicing, refactoring, transformations, abstract interpretation, static analysis and verification. After the workshop, four outstanding papers selected by the program chairs were invited for publication in this *Journal of Software Maintenance and Evolution*: *Research and Practice* special issue. The authors of the selected papers were asked to prepare a significant extension on the workshop version of their papers. Each paper underwent a rigorous reviewing

*http://www.ieee-scam.org/

process, involving three independent reviewers and two rounds of revisions. Eventually, three papers were accepted for inclusion in this special issue, covering three different directions of source code analysis and manipulation: the study of identifier well formedness, the construction of accurate call graphs and support for automatically validating annotations in Java.

The use of well-formed variable names is crucial for code quality, particularly for supportting code comprehension processes. In particular, identifiers should be concise and consistent, in that it is possible to build a mapping from the domain of identifiers to the domain of concepts. In the paper 'An Empirical Study of Rules for Well-formed Identifiers', Lawrie, Feild and Binkley proposed an approach for verifying the well formedness of identifiers. While mappings between identifiers and concepts require domain experts, this work empirically explored whether syntactic violations are indicative of concept mapping violations. The authors report an empirical study featuring 48 million lines of code, showing that syntactic violations occur in practice, that these violations largely match with concept-based violations, and that open source systems tend to exhibit a higher percentage of violations than proprietary systems, due to the continuously increasing number of contributors for the former and the greater engineering discipline for the latter.

Call graphs are used in a wide number of software engineering tasks, such as program understanding and testing, as well as in the optimization phase of compilers. In particular, application call graphs represent calling relationships among methods, which can happen through libraries. Such call graphs are particularly useful since they provide a higher level of abstraction than whole call graphs. Construction of call graphs is relatively straightforward for procedural programming languages, while dynamic dispatch makes such a task more difficult for object-oriented languages. In the paper 'Automatic Construction of Accurate Application Call Graph with Library Call Abstraction for Java', Zhang and Ryder proposed an approach for the automatic construction of accurate application call graphs for Java. They limit the presence of spurious calls by introducing a data reachability algorithm and outline the applicability of the approach to program slicing and dataflow testing.

In the paper 'AVal: an Extensible Attribute-oriented Programming Validator for Java', Noguera and Pawlak proposed a framework for defining and applying constraints on (collections of) annotations in a Java program. In particular, the authors presented a generic method for specifying annotations and rules over them using a consistent higher-level notation. This allows the user to express certain domain-specific aspects of the program at a higher level of abstraction which can automatically be enforced. In addition, by hiding the implementation details of these annotations and their constraints from the program code, the program's complexity is reduced, resulting in simpler, more readable programs that are easier to evolve.

We hope that readers will enjoy this special issue and gain useful insights from the three papers presented. We would like to thank all the authors who submitted papers to the workshop and to this special issue, as well as the SCAM 2006 program committee and the external reviewers who helped in making this special issue possible. Finally, we would like to thank the editorial board of the *Journal of Software Maintenance and Evolution*: *Research and Practice* and the publisher Wiley for providing us with the opportunity to devote an issue of this journal to SCAM 2006.

MASSIMILIANO DI PENTA
RCOST—University of Sannio, *Benevento*, *Italy*

LEON MOONEN
Delft University of Technology, *Delft*, *The Netherlands*