Software Qual J (2011) 19:237–238
DOI 10.1007/s11219-011-9134-y

SCAM 2009

Guest editor's introduction to the special section on source code analysis and manipulation

Sibylle Schupp · Andrew Walenstein

Published online: 5 March 2011

Source code plays a central part of software-based systems, and means of analyzing and manipulating it play important roles in assuring the quality of software. Three invited papers from the recent IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2009) conference highlight the diversity and importance of this role, from program development, to analysis of concurrency, and even to analysis and evolution of language definitions.

The source code of a software-based system typically contains the only precise definition of system behavior. For this reason, the theories, tools, and techniques for analyzing and manipulating source code are important for software engineering in general, and for assessing and ensuring software quality in particular.

This special section features three extended papers from the Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2009). The conference took place on September 21–22, 2009, in Edmonton, AB, Canada, and was co-located with the Twenty-fifth International Conference on Software Maintenance (ICSM 2009). Seventeen papers were accepted to the conference; their topics included as follows: software clone detection and evolution, dependency analysis, change tracking, program slicing and chopping, decompiling, model checking, and software metrics. The three extended papers selected for this special issue collectively represent a fine example of the source code focus of the conference, its mix of theoretical and practical content, and the importance of source analysis and manipulation for assuring software quality.

The paper by Dennis Giffhorn, "Advanced Chopping of Sequential and Concurrent Programs" addresses program chopping in the context of concurrent programs, a problem that has not been studied previously. Program chopping is useful in many engineering problems, including analyzing vulnerabilities and for input validation. Chops are specific

S. Schupp (✉)
Technische Universität Hamburg-Harburg, Schwarzenbergstraße 95 (E),
D-21073 Hamburg,
Germany
e-mail: schupp@tuhh.de

A. Walenstein
University of Louisiana at Lafayette, P.O. Box 41771, Lafayette, LA 70504-177, USA

subsets of a program relating to both a source and target location in a program. In the paper, Giffhorn illustrates that precise chopping can be costly enough without considering concurrency due to context dependency, and taking into account concurrency adds the extra problem of time dependency. Giffhorn presents six different chopping algorithms for concurrent programs; they vary according to how they trade-off context- and time-sensitivity and thus, trade-off precision and runtime efficiency. The algorithms were evaluated in a study using Java programs; the study shows that the time-sensitive versions, while powerful, have a worst-case time complexity that may limit their utility in many circumstances.

The paper by Rebecca Tiarks, Rainer Koschke, and Raimar Falke, "An Extended Assessment of Type-3 Clones as Detected by State-of-the-Art Tools" tackles several problems relating to code clones, specifically so-called Type-3 clones, which are segments of code that are similar to other segments, but are still different due to some modifications. To date, Type-3 clones have been difficult to analyze due in part to lack of knowledge of the distribution and properties of real-world Type-3 clones and whether cues can be found in them that separate actual clones from false positives. The paper by Tiarks et al. considers the problem comprehensively. They report on an empirical study in which they summarize observed syntactic differences in the Type-3 clone candidates as reported by state-of-the-art tools and generate a catalogue of abstractions to relate them. They also probed the patterns of differences in candidate clones to understand whether they can be used to help separate true clones from false positives and thereby, increase the precision of future clone detectors. A follow-on study shows that decision tree-based classifiers could be trained to separate the true from false positives with some success.

The paper by Ralf Lämmel and Vadim Zaytsev, "Recovering Grammar Relationships for the Java Language Specification" considers source code analysis and manipulation from another level: that of the language specifications themselves. Language specifications, i.e., grammars, are the ultimate authority for many questions of tool or software design, but are not always as correct as one might wish for. Grammars might exist in different formats—BNF, XML schema, as parser descriptions or software models—written independently or obtained from another; they also evolve over time. Recovering intended or accidental differences between different incarnations is actually not easy, and proving that two grammars are equivalent is theoretically undecidable. Based on a transformational method from grammar engineering called grammar convergence, Lämmel and Zaytsev describe an automated approach for detecting differences and ensuring consistency between different versions of a grammar. The method is evaluated using 3 different versions of the Java Language Specification (JLS), containing 2 grammars each. Aside from discovering bugs in JLS, the approach presents a reproducible and precise way of comparing grammars.