Guest Editors’ Introduction to the Special Issue on the International Conference on Software Maintenance and Evolution

Tibor Gyimóthy and Václav Rajlich

The International Conference on Software Maintenance and Evolution (ICSM) is the leading research conference in software maintenance and evolution. It provides a widely recognized international forum for discussions and exchange of ideas between researchers and experts.

ICSM 2005 was held in Budapest on 26-29 September 2005. In cooperation with the ICSM conference, several collocated workshops were organized. These included the Seventh IEEE International Symposium on Web Site Evolution (WSE), the Fifth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM), the Third International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), the Software Evolvability Workshop, and the 13th International Workshop on Software Technology and Engineering Practice (STEP).

For ICSM 2005, 180 research papers were submitted. From this record number of papers the program committee selected 55 full papers for inclusion in the proceedings. Of these 55 papers, eight were extended from their conference version, passed through the standard TSE review process involving three anonymous reviewers per paper, and are contained in this special issue.

The first paper, “Feature Identification: An Epidemiological Metaphor” by G. Antoniol and Y.-G. Guéhéneuc, presents an epidemiology inspired approach to feature identification. The approach relies on statistical analysis of static and dynamic data and identifies features of large, multithreaded object-oriented programs: the Firefox e-mail tool and the Mozilla Web browser. The results suggest that the epidemiological analysis overcomes the limitations of existing methods and increases the accuracy of the feature identification. Extracted microarchitectures consist of variables, structures, classes, and methods, and support the maintenance and program understanding.

The second paper, “Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Programs” by L.C. Briand, Y. Labiche, and J. Leduc, provides a methodology to reverse engineer sequence diagrams from dynamic analysis data. Because of inheritance, polymorphism, and dynamic binding, reverse engineering and understanding the behavior of an object-oriented system is difficult. Multithreading and distribution further complicate the analysis. In order to separate the instrumentation from the application code, aspect-oriented programming was used. The approach is illustrated on a distributed library management system developed in Java, using RMI as the distribution middleware.

The third paper, “Static Analysis of Object References in RMI-Based Java Software” by M. Sharp and A. Rountev, presents a points-to analysis of distributed Java programs that contain the Remote Method Invocation (RMI). Points-to information is often a prerequisite for program understanding, testing, and code optimizations. The existing points-to analysis techniques cannot be applied directly to RMI-based distributed applications. The approach presented in this paper allows us to represent the flow of remote objects, the effects of remote invocations, and the remote propagation of object graphs through serialization. The result of an experimental study suggests that the approximate version of the analysis could be a good option for a relatively precise and practical points-to analysis of RMI-based Java applications.

The fourth paper, “Incremental Maintenance of Software Artifacts” by S.P. Reiss, presents a software development tool (CLIME) that identifies potential problems and inconsistencies between software design, specification, documentation, source code, test cases, and other artifacts. The approach is based on constraints implemented as database queries that describe the association of the different software artifacts. These constraints are maintained and checked incrementally during the evolution of the system and the inconsistencies are reported to the developers. The approach is validated on several software development projects where many language and documentation problems and inconsistencies between the code and documentation have been identified.

The fifth paper, “Tool-Supported Refactoring of Object-Oriented Code into Aspects” by D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, presents refactoring Object-Oriented Programs (OOP), written in Java, into equivalent Aspect-Oriented Programs (AOP) written in AspectJ. The approach relies on six refactorings to support migration from OOP to AOP. These refactorings are combined with existing OO transformations into a tool, AOP-Migrator, which is implemented as an Eclipse plug-in. The approach is applied to several Java systems: JHotDraw,

T. Gyimóthy is with the Software Engineering Department, University of Szeged, H-6720 Szeged, Arpad ter 2, Hungary. E-mail: gyimothy@inf.u-szeged.hu.

V. Rajlich is with the Department of Computer Science, Wayne State University, Detroit, MI 48202. E-mail: rajlich@wayne.edu.

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

Published by the IEEE Computer Society
PetStore, Jaccounting, and JSpider. The results suggest that, although many transformation steps can be automated, there are some refactoring decisions which are best left to the human.

The sixth paper, “Identifying Failure Causes in Java Programs: An Application of Change Impact Analysis” by X. Ren, O.C. Chesley, and B.G. Ryder, provides a tool (Crisp) that aids programmers during regression testing. Crisp uses the results from a tool (Chianti) that decomposes a program edit into atomic changes, identifies the affected tests, and finds the parts of the edit that affect a particular test (affecting changes). Crisp constructs intermediate versions of the program that has been edited and determines the changes that resulted in a failing regression test. In an experimental study, the authors were able to find the failure-inducing changes in 67 of 68 failing tests for two Java programs.

The seventh paper, “On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques” by H. Do and G. Rothermel, presents two controlled experiments that assess prioritization techniques using mutation faults. The effectiveness of the prioritization has been examined in terms of the rate of fault detection, focusing on Java programs. The results suggest that the test case prioritization based on mutation faults may provide an effective approach when compared to the methods using hand-seeded faults.

The eighth paper, “The SEXTANT Software Exploration Tool” by T. Schäfer, M. Eichberg, M. Haupt, and M. Mezini, presents a software exploration tool (SEXTANT) that supports software comprehension. This tool supports five functional requirements derived from the analysis of the existing tools and from the studies on developer needs. The paper presents four case studies.

This issue is a result of a great effort of many people. First of all, the guest editors would like to thank to Harry M. Sneed, the general chair of ICSM 2005, for the excellent organization of the conference that provided a record number of high quality papers from which we selected the papers that appear in this special issue. Many thanks to all authors and to the reviewers for their hard work. We also appreciate the excellent support and cooperation provided by the IEEE Computer Society Staff (especially Suzanne Werner).

Tibor Gyimóthy
Václav Rajlich
Guest Editors