2005

Guest Editorial: Special Issue on Software, Maintenance and Evolution

Panos K. Linos
Butler University, linos@butler.edu

Guest Editorial: Special Issue on Software Maintenance and Evolution

Mark Harman, Bogdan Korel, and Panagiotis K. Linos

1 Software Maintenance

Software maintenance and evolution continues to play a vital role in the development of software systems. It is widely acknowledged that the majority of development effort, and thereby expenditure, is allocated to postinitial release activity. This activity, which takes place after the software has seen its first release, is known as software maintenance (or software evolution).

For many years there has been an international effort among researchers and practitioners to control the costs associated with software maintenance and evolution and to improve the quality of the processes and products associated with this vital activity. The problems in this area are particularly challenging because they require experience and expertise that spans the entire spectrum of work on both software engineering and computer science and beyond.

Software maintenance encompasses human elements of software engineering, which require psychological, statistical, and empirical evaluation. It involves algorithms, methods, and techniques which require a deep understanding of all activities in the software development life cycle. The algorithms and tools that support software maintenance often manipulate software to support its evolution, with the consequence that correctness is a paramount concern. The systems that form the subjects for this work are often the largest and most complex currently known, raising the additional challenges of scalability and robustness.

2 The 20th IEEE International Conference on Software Maintenance

In 2004, The IEEE International Conference on Software Maintenance (ICSM) celebrated its 20th anniversary. Since its inception in 1983, ICSM has evolved into an international forum for researchers, practitioners, educators, technology transfer experts, project managers, tool developers, and users who explore current issues facing the computing community. Every year ICSM attracts people from academia, government, nonprofit organizations, and the software industry.

For the first time since 1998, ICSM 2004 was colocated with the 10th IEEE International Symposium on Software Metrics (Metrics 2004). In addition to the colocation with Metrics 2004, ICSM 2004 was also colocated with a number of workshops and colocated events which have sprung up around the main conference and which provide a focus for particular activities associated with software naintenance and evolution. In 2004, these colocated workshops included the Ninth IEEE Workshop on Empirical Studies of Software Maintenance (WESS '04), the Sixth International Workshop on Web Site Evolution, the Fourth International Workshop on Source Code Analysis and Manipulation (SCAM '04), and Software Technology and Engineering Practice (STEP).

3 In this Special Issue

In 2004, the ICSM program chairs, Mark Harman and Bogdan Korel, selected 38 papers for inclusion in the proceedings from 122 submissions. All papers were refereed by at least three referees. Of the 38 papers submitted, eight were selected for consideration for this special issue. These papers were extended from their conference version and reviewed according to the *IEEE Transactions on Software Engineering* reviewing process. Seven of the papers successfully completed the review process and are contained in this special issue. The rest of this editorial provides a brief overview of these seven papers.

The term "crosscutting concern" refers to functionality that is scattered among several software modules. When such a concern (or functionality) needs to be modified, software engineers need to detect the code that implements it, which is not a trivial task. The term Aspect-Oriented Software Development (AOSD) has been introduced for addressing the problem of crosscutting concerns and various aspect-oriented languages have been created that implement the concept of an aspect, which facilitates the process of capturing crosscutting concerns in a localized manner. Two of the papers in this special issue concern AOP.

The paper entitled "On the Use of Clone Detection for Identifying Crosscutting Concern Code" considers clone-detection as a potential technique for identifying crosscutting concerns. In this paper, the authors claim that clone detection techniques appear to be a promising approach for

- *M. Harman is with the Software Engineering Group, Department of Computer Science, King's College London, Stran, London WC2R 2LS, UK. E-mail: mark@dcs.kcl.ac.uk.*
- *B. Korel is with the Computer Science Department, Illinois Institute of Technology, 10 West 31st Street Street, Chicago, IL 60616. E-mail: korel@iit.edu.*
- *P.K. Linos is with Butler University, Fairbanks Center for Communications and Technology, Department of Computer Science and Software Engineering, 4600 Sunset Avenue, Indianapolis, IN 46208-3485. E-mail: linos@butler.edu.*

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

identifying certain types of crosscutting concerns. In order to support their hypothesis, they present the results of a case study performed on the source base of a lithography system of approximately 10 million lines of C code. More specifically, their research findings support the hypothesis that some crosscutting concerns are implemented by similar pieces of code (i.e., clones), which are spread throughout the system. In fact, their study shows that such code pieces can comprise up to 25 percent of the code size. This paper provides some groundwork for future research toward the automation of aspect mining activities based on clone detection.

In the paper entitled "Refactoring the Aspectizable Interfaces: an Empirical Assessment," the authors focus on a specific kind of crosscutting concerns, referred to as aspectizable interfaces. Aspectizable interfaces are interfaces that crosscut the principal decomposition. The paper reports on an experimental assessment of the effects of migrating the implementation of the aspectizable interfaces to aspects. The results of this study suggest that the migration of the aspectizable interfaces produces code that is easier to understand during the execution of maintenance tasks.

As with software engineering in general, some aspects of software maintenance have an inherently empirical character because it is important to attempt to provide empirical evidence to support the claims made for the methods, approaches, and techniques which software maintainers may adopt to improve software maintenance. In "An Experimental Investigation of Formality in UML-Based Development," the authors present the results of two experiments on the impact of using a formal constraint language (the Object Constraint Language, OCL) with Unified Model Language (UML) models, contrasting this with the use of natural language to express constraints. The results suggest that, while adequate training is necessary, the use of OCL offers significant benefits.

In software maintenance, it is important to be able to automate and semi-automate the process of analysis and characterization of the evolution of software systems. Methods, algorithms, and approaches which provide automated support for human analysis of the evolutionary process postsoftware delivery are therefore highly important. Two papers in this special issue concern techniques for supporting the process of human analysis of software evolution.

The paper entitled "Analyzing the Evolutionary History of the Logical Design of Object-Oriented Software" presents a method for analyzing the evolution of object-oriented systems from the point of view of their logical design. The method is based on a heuristic domain-specific structure differencing algorithm that recognizes the design-level changes. The paper reports on a case study that uses the presented method to determine the evolution profiles for classes of a large Java system.

Program spectra characterize a program's behavior inside the black box. In the paper entitled "Checking Inside the Black Box: Regression Testing by Comparing Program Spectra," the authors present a new class of program spectra referred to as value spectra. Value spectra differences between program versions are used to expose internal behavioral differences during regression testing. The paper reports on an experimental study that evaluates the presented value spectra.

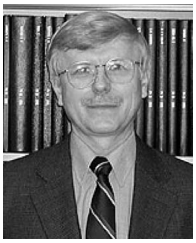
Fault detection is a vital aspect of software engineering and software maintenance and assessment of approaches that may predict the presence of faults remains a pressing concern. Two of the papers in this special issue address the problem of finding faults for software as it evolves over a series of releases.

The paper entitled "Studying the Fault-Detection Effectiveness of GUI Test Cases for Rapidly Evolving Software" addresses some shortcomings of modern smoke regression techniques, such as the inability to automatically retest Graphical User Interfaces (GUIs). To this end, the authors present various empirical results and discuss solutions to this problem. More specifically, the contributions of this paper include the identification of requirements for GUI smoke testing and the formalization of a GUI smoke test as a sequence of events. In addition, the authors discuss and use DART (Daily Automated Regression Tester), which is an automated regression testing process. Finally, the results of their empirical studies demonstrate the feasibility of the overall smoke testing process in terms of execution time and storage space. Some other equally important results of this research indicate that smoke tests cannot cover certain parts of the code and that having comprehensive test oracles may balance off the lack of large smoke test suites.

In "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," the authors present an empirical investigation into the bug-prediction capability of the widely studied object-oriented metrics of Chidamber and Kemerer for open source software. The paper uses regression analysis, decision trees, and neural nets to investigate the metrics' bug-predicting ability and considers the evolution of Mozilla over seven releases.

Mark Harman
Bogdan Korel
Panagiotis K. Linos
Guest Editors

Mark Harman is a professor of software engineering and the head of the Software Engineering Group in the Department of Computer Science, King's College, London. He has published extensively on program slicing, transformation, and testing. More recently, he was instrumental in founding the field of search-based software engineering, which concerns the application of meta-heuristic search algorithms to problems across the spectrum of software engineering activity. He is a member of the editorial board of seven journals in the software engineering area, including the *IEEE Transactions on Software Engineering*.



Bogdan Korel is an associate professor in the Computer Science Department at the Illinois Institute of Technology, Chicago. His research interests are in software maintenance, software testing, and automated software analysis. His major research contributions are in software slicing and automated test generation.

Panagiotis (Panos) K. Linos is a professor of computer science and software engineering at Butler University. Before joining Butler, he was the chairperson of the Computer Science Department at Tennessee Technological University. He is the founder of the CeASER (Center for Applied Software Engineering Research) and EPICS (Engineering Projects in Community Service) program at Butler University. His research interests and activities focus on software maintenance and evolution, reengineering and reuse, program comprehension, software measurements, and metrics.