Guest Editor's Introduction: 10th Working Conference on Reverse Engineering

Arie van Deursen, *Member*, *IEEE Computer Society*, and Eleni Stroulia, *Member*, *IEEE*

1 Introduction

This special issue marks the 10th anniversary of WCRE, the Working Conference on Reverse Engineering. Given a readily available representation of a software system, reverse engineering aims at obtaining an abstract representation of the system. For example, starting with binaries, source code, or execution traces, a reverse-engineering method may produce as output a higher-level representation of the same system, such as source code, architectural views, or use cases, correspondingly.

Reverse-engineering methods and technologies play an important role in many software engineering tasks, such as bug fixing, software adaptation and maintenance, and system reengineering or migration. All these activities require, to some extent, an understanding of the system that transcends a limited locality and a single point of view. For example, to fix a particular bug in an object-oriented system, a developer may have to understand how the undesirable runtime behavior of a particular object may be caused by behavior defined in the object's ancestor classes. In such a case, reverse engineering a class diagram from the system code and a set of relevant sequence diagrams based on the traces of its erroneous execution instances becomes indispensable. Alternatively, the task may be the translation of the system from an imperative to an object-oriented language because the compiler of its original programming language is no longer supported. Then, clustering of the system procedures around the data elements on which they depend is essential for obtaining an object-oriented conceptualization of the system. Finally, when a developer considers a particular refactoring, clone detection could simplify his task by pinpointing the various code fragments that are candidates for modification.

Since 1990, when Chikofsky and Cross [1] presented their taxonomy of reverse-engineering tasks, a substantial body of work has been conducted in the area with important results. A range of techniques have been developed for system-dependency analysis, clone detection, architecture reconstruction, data reverse engineering, legacy-system reengineering, and user-interface migration.

Today, the research agenda of the reverse-engineering community is evolving. Emerging software platforms and architectures (service-oriented architectures on Web services standards) necessitate that we work on supporting the migration of current systems towards these new platforms. Furthermore, as software development becomes increasingly evolutionary, especially with the advent of agile methods, the reverse engineering of trends and events in the evolution of systems based on their revision histories is becoming more and more important. Finally, new programming abstractions, such as aspects for example, motivate a suite of new methods aimed at recognizing these abstractions in software.

2 This Issue

The papers in this special issue provide an interesting cross section of the field of reverse engineering. They cover some traditional as well as some more modern topics in the area, including binary reverse engineering, up to the analysis of source code repositories in order to reconstruct the historic origins of certain pieces of code.

2.1 Deoptimization

There are several good reasons why one may want to reverse engineer binaries, such as malware detection, patent violation discovery, or interoperability analysis. However, compilers typically carry out a number of low-level optimizations, such as instruction scheduling, predication, and speculation, which restructure the low-level code of programs. Consequently, it is much harder to reconstruct the original logic of the code. The paper by Snavely et al. addresses this problem, by proposing techniques by which the effects of various optimizations on the code structure can be (partially) undone.

2.2 Application Retargeting

Advances in hardware can provide new opportunities to software applications, justifying a migration to the new hardware platform of the application in question. The paper by Baumstark and Wills deals with such migrations for image-processing applications. A large body of such applications exists, which are mostly written in C. Therefore, they are not able to make appropriate use of new data-parallel hardware developments, which would enable these applications to run on a new generation of

- *A. van Deursen is with CWI and Delft University of Technology, PO Box 94079, 1090 GB Amsterdam, The Netherlands. E-mail: Arie.van.Deursen@cwi.nl.*
- *E. Stroulia is with the Department of Computing Science, University of Alberta, 221 Athabasca Hall, Edmonton, AB, T6G 2E8, Canada. E-mail: stroulia@cs.ualberta.ca.*

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

portable video products. The authors present an approach for automatically extracting a data-parallel program model from sequential image processing code and apply their technique to a set of production programs.

2.3 Software Querying

Many reverse engineering approaches build up a representation of a software system in terms of graphs or relations. Calculating with such graphs or relations can be used for the detection of design patterns or patterns of problematic design and the computation of design metrics. Such applications require an expressive query language and an efficient implementation especially for queries on large graphs. Beyer et al. propose RML, the Relation Manipulation Language, and its implementation CrocoPat. Key characteristics of CrocoPat are its capability to deal with relations of arbitrary arity and its efficient implementation based on binary decision diagrams.

2.4 Software Clustering

Clustering is among the earliest research problems in the area of software reverse engineering. The general objective is to analyze the various interdependencies among the system entities so that they can be organized in cohesive clusters that provide an insight to the system architecture. Andritsos and Tzerpos discuss a novel approach for integrating different types of attributes to assess inter-component distance based on an information-loss minimization metric.

2.5 Source Code Histories

One of the aims of reverse engineering is to support program comprehension. Godfrey and Zou provide a technique that helps a developer to gain a better understanding of the original context of an apparent design change. Their "origin analysis" technique compares different versions of a system, stored in a version management system. More specifically, this technique can be used to detect how files and functions are merged or split between different versions, thus helping developers answer such questions as "why was this new function added?" or "where did the XYZ functionality go?"

3 Paper Selection Procedure

The papers in this special issue were selected from the full set of papers and presentations given at the 10th Working Conference on Reverse Engineering (WCRE) held in Victoria, Canada, from November 13-16, 2003, the proceedings of which were published by the IEEE Computer Society. Based on WCRE review reports and discussions during the conference, an initial list of nine papers was selected. These were subsequently subjected to two regular rounds of full reviewing, resulting in the five papers presented in this special issue.

References

[1] E.J. Chikofsky and J.H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, pp. 13-17, Jan. 1990.

Acknowledgments

We would like to thank all authors (including those whose papers were eventually not selected) and all reviewers for the effort they made for this special issue. A special word of thanks goes to the staff of the *IEEE Transactions on Software Engineering* for their support throughout the reviewing process.

Arie van Deursen (http://www.cwi.nl/~arie/) received the MSc degree (1990) from the Vrije Universiteit Amsterdam and the PhD degree (1994) from the University of Amsterdam. He is a professor of software engineering at Delft University of Technology, and research leader at CWI, the Dutch National Research Institute in Mathematics and Computer Science. His research interests include aspect-oriented software development, program analysis, software testing, and program comprehension. He is a member of the IEEE Computer Society, the ACM, and the ACM Special Interest Group on Software Engineering. He was WCRE program chair in 2002 and 2003.

Eleni Stroulia (http://www.cs.ualberta.ca/~stroulia) received the BSc degree (1989) degree from the University of Patras in Greece and the MSc (1991) and PhD degrees (1994) from the College of Computing, Georgia Institute of Technology. She is an associate professor with the Department of Computing Science at the University of Alberta. The research of her group focuses on developing and applying artificial intelligence and machine-learning algorithms for software analysis and reengineering, migration to service-oriented architectures, and tools for mentoring software-development teams. She is a member of the ACM, IEEE, and AAAI. She was WCRE program chair in 2003 and 2004.