FOCUS: GUEST EDITORS’ INTRODUCTION

REFACTORING CHANGES a program’s source code without changing its external behavior, typically to improve the software’s design. Optimization is similar but has different goals. We’ve always considered refactoring akin to prime numbers; that is, any sufficiently advanced culture will discover the concept. The notion that restructuring code can improve its design can be traced back to computing’s early days; Lisp practitioners and some of the Forth literature attest to that fact. However, it wasn’t until 1990 that the term “refactoring” appeared in print in a research paper by William Opdyke and Ralph Johnson that described the process and identified common refactorings.

Since then, the processes have been formalized (somewhat), common refactorings have received names, some operations have been automated and incorporated into development tools, and the entire concept has emerged as an academic-research area. Refactoring has been adopted as an agile practice, has been applied to help reengineer and evolve legacy code, and has been applied in other software development contexts. So, many programmers are looking for better support for these operations in their daily work.

In May 2014—nearly a quarter century since refactoring research’s early days—41 people attended the Future of Refactoring seminar in Dagstuhl, Germany. Attendees came from across the spectrum: from industry to academics and from originators of the field to relative newcomers. We spent a week collaborating to deepen our understanding of what was being accomplished in refactoring and to determine the directions in which research and industry should go. One important
outcome was the idea to produce this special issue of IEEE Software. Another outcome was that attendees noticed that practitioners often relax the behavior-preservation aspect of refactoring. The article by Munawar Hafiz and Jeffery Overby (see the next section) examines this issue.

In This Issue
The articles we selected range from historical, exploring refactoring research’s origins, to practical, exploring software developers’ experiences with refactoring, to theoretical, exploring new refactoring techniques that haven’t yet appeared in the wild.


“Refactoring Myths,” by Munawar Hafiz and Jeffrey Overby, examines popular misconceptions about tool-based refactorings.

“Challenges to and Solutions for Refactoring Adoption: An Industrial Perspective,” by Tushar Sharma and his colleagues, is an experience report about adopting refactoring techniques at Siemens Corporate Development Center India.

“Refactoring for Asynchronous Execution on Mobile Devices,” by Danny Dig, describes a particular common situation: applying refactorings to convert a long-running, synchronous process into an asynchronous one.

“Refactoring—a Shot in the Dark?,” by Marko Leppänen and his colleagues, analyzes interviews of 12 seasoned software architects to determine how refactoring is used in the real world.

“Database Refactoring: Lessons from the Trenches,” by Gregory Vial, is an experience report of applying refactorings to restructuring a relational database in a small-to-medium enterprise.

In the Point/Counterpoint department, John Brant and Friedrich Steimann debate whether current tools are trustworthy and how much that really matters.

Whatever your interest in refactoring, there’s something in this issue that will deepen your understanding of refactoring and enrich your design experience. We hope you enjoy the issue.

Reference

See www.computer.org/software-multimedia for multimedia content related to this article.