Why Programs Fail: A Guide to Systematic Debugging

Andreas Zeller, Morgan Kaufmann, 2005, \$54.95,
ISBN: 1558608664

Sadly, debugging is an often-neglected topic in software engineering education. Although preventing errors in the first place is certainly desirable, to err is human, to rectify is wise, and to learn from mistakes is a must.

Why Programs Fail attempts to provide a systematic approach to finding, reproducing, and fixing programming errors, with a strong focus on the automation of many debugging techniques. Before delving into the details of debugging techniques, author Andreas Zeller covers some prerequisites every software developer should be acquainted with—in particular, problem-tracking systems and testing techniques.

Once the stage is set, Zeller explores the debugging process in depth. After tracking the problem in the database, the first task in debugging is reproducing the failure as stated in the original problem report. It provides the foundation upon which the debugging process is built. Once you have a reproducible problem, you can proceed and simplify it with delta debugging, an automated method to simplify test cases. Debugging then consists of obtaining a theory that explains the bug. Zeller advocates the application of the scientific method and suggests the use of a debugging logbook to make debugging explicit.

The core of the book deals with techniques for creating hypotheses and determining the failure causes of observed problems. Deductive techniques cover the whole gamut of static analysis tools, from program slicing to specialized code smell detectors. Most debugging techniques, however, are dynamic by nature. Observational techniques include logging, dynamic slicing, using debuggers, and introducing assertions to detect contract infringements automatically and to discard infection sites. Inductive techniques help make debugging more efficient by focusing on aspects of a failing run that differ from normal passing runs. Efficiency is also the focus of experimental techniques that try to automate the search for failure causes and failure-inducing program states.

This is an outstanding practical book that explores many interesting ideas. In the near future, we may see many of them integrated within our development envi-

ronments. If you would like to improve your detective skills as a programmer, this is certainly the book to read.

—Fernando Berzal

Model Driven Architecture and Ontology Development

Dragan Gasevic, Dragan Djuric, Vladan Devedzic,
Springer, 2006, \$59.95, ISBN: 3540321802

Ontologies—descriptive models of the objects and relationships that characterize a specific problem domain—seem to be a hot topic in computing these days. The Semantic Web is one of the major driving forces, but as ontologies and ontological tools become more prevalent, the potential for using them in other contexts has become evident.

The difficulties of using ontological methods, however, have also become more evident. While the basic ideas behind building an ontological model are simple enough, the reality is far different. Complications arise when you try to write down a model using some sort of formalism.

The stated goal of this book is to show how to use model-driven architecture techniques to build ontologies. To this end, the first section covers the basics of knowledge representation, defines ontologies in general and the Semantic Web in particular, and finishes with a discussion of model-driven architecture and “modeling spaces”—the authors’ framework for building models using a variety of approaches. The second section introduces some tools and ontology description systems. A final section discusses tools for ontological modeling and presents some limited and relatively high-level examples of ontologies.

The first part of the book manages to present a number of ideas in a concise and digestible way. The second half is harder to understand, and difficult notions are not always given the attention they deserve. Another difficulty arises from the authors’ presentation of using UML to model ontologies. They need to describe ways to work around the limitations of UML rather than using a more focused ontology development tool, such as Protégé.

This is a decent book on a difficult topic, but it is probably not the definitive reference work and would not be a good place to start for students in the field.

—Jeffrey Putnam

Reprinted from *Computing Reviews*, © 2007 ACM, <http://www.reviews.com>