

# ***UML Distilled: From Difficulties to Assets***

**Stephen Mellor**

*UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed., by Martin Fowler, Addison-Wesley Professional, 2003, ISBN 0-321-19368-7, 208 pp., US\$34.99.*

It's difficult to have a problem with anything Martin Fowler writes. He's obviously a skilled designer, he practices what he preaches, and he has a forceful command of the English language without being stuffy. He writes from his head and his heart, so it's always an informative, easy, and often amusing read. Moreover, I was a reviewer for this edition of *UML Distilled*, and there's an appropriately complimentary quote from me on the cover page. For all that, I found myself having difficulties with this book. But these difficulties aren't a reason to put the book back on the shelf; rather, they're precisely the reason you should read it.

While I was rereading the book, reviewing it for *IEEE Software* became an increasingly uncomfortable prospect. Was it Martin's use of the Unified Modeling Language for sketching instead of building executable models that bothered me? Or was it his use of UML to capture software structure instead of conceptual domain knowledge? Perhaps it was the way he mixed application and architecture instead of treating the two separately with a set of mapping rules to create an implementation? No, I concluded, it was none of these issues. After all, I've spent six (long) years working with the UML standard, and I'm inured to these perfectly valid, indeed common, uses of the language.

Maybe it was the many descriptions of Martin's preferred approach, irrespective of the standard's dictates? One of these struck me halfway through the book, on page 72: "Although it was dropped from UML 2.0, {frozen} is a very useful concept, so I would

continue to use it." It was hardly the first time a sentence like this had appeared. Many hints and tips are included about how to use UML well and bend it to your will, including sidebars on patterns—which aren't strictly a part of the language—and the thoroughly unUML CRC (class-responsibility-collaboration) cards. Then, finally, I understood my difficulty: the book isn't about UML 2.0 per se, but how best to use UML to build good designs. UML 2.0 is merely the carrier of those ideas, much as escargots are a fine excuse to chuck down garlic and butter.

To add support to this view, the book sports an approachable bibliography containing primarily method, design, and process books that everyone should read along with *UML Distilled*. These books aren't specifically about UML either, but how to use it.

Notwithstanding my earlier difficulties, I was interested to see what changes had been made to various parts of UML 2.0 outside my particular area of expertise. (Few of us have time to keep up, which is a major advantage of a distillery such as this.) The inside jackets usefully summarize the notation, and page 11 has a table of the various diagram types and their provenance. Though sometimes very short, the chapters in the book's last third survey these diagrams. This isn't quite enough information for the fascinated reader, but it's sufficient to decide which UML specification areas to explore further.

So this book's real value centers on the difficulty I was having with it: *UML Distilled* is mostly a book on how to sketch a good design

using UML 2.0. If you insist on building systems using his approach, as Martin is a skilled designer who speaks from experience, you would find this difficulty to be an asset.

**Stephen Mellor** is a chief scientist in the Embedded Systems Division of Mentor Graphics. Contact him at Stephen\_Mellor@Mentor.com.

## Three Perspectives on Software Testing

### Fernando Berzal

*Testing Computer Software, 2nd ed., by Cem Kaner, Jack Falk, and Hung Quoc Nguyen, Wiley, 1999, ISBN 0-471-35846-0, 496 pp., US\$50.00.*

**Test-Driven Development: By Example** by Kent Beck, Addison-Wesley, 2003, ISBN 0-321-14653-0, 240 pp., US\$34.99.

*Lessons Learned in Software Testing* by Cem Kaner, James Bach, and Bret Pettichord, Wiley, 2002, ISBN 0-471-08112-4, 352 pp., US\$39.99.

Testing is the process of searching for errors, according to Cem Kaner, Jack Falk, and Hung Quoc Nguyen's *Testing Computer Software*. Basically, that search intends to discover errors before they make it to the system's final user.

You can approach the search for errors from different angles depending on your role in an organization. These three books—*Testing Computer Software*, *Test-Driven Development: By Example*, and *Lessons Learned in Software Testing*—focus on testing computer software from different perspectives.

### For testers

The best-selling *Testing Computer Software* is aimed at people who want to begin their software development career as testers (a common entry-level position in many companies). As the authors state in the preface—and as anybody in the software field knows—

“we don't expect to meet many CS graduates who learned anything useful about testing at a university.” This common lack of preparation is what makes *Testing Computer Software* an ideal resource for future testers. Moreover, although some programming knowledge is always useful to track bugs, the book's focus on black-box testing means that the reader doesn't need to be a programmer to become a good tester.

The authors divide the book into three sections. The first section covers black-box testing fundamentals, including common terminology and the design of test cases that look for boundary conditions. It also emphasizes taking the right “destructive” attitude toward the program you're testing (to make the best possible use of your testing time) and writing reproducible bug reports (the tester's primary work product). As a complement to the software error categories discussion, an appendix lists more than 400 specific types of errors, an invaluable resource for ideas on what to look for when testing your software.

The book's second section delves into details a tester should master, from test case design to test planning, from analyzing problem-tracking systems to specific testing tasks such as configuration, localization, and documentation testing. Although the examples here are somewhat dated, the book uniquely covers the political issues surrounding the often unpleasant work of testers who must bring defects to the spotlight.

Finally, the third section covers management topics. Although those earnest to get to work might find this part boring, it puts the testers' role in perspective, showing how different testing techniques apply depending on your software development model and organization type. It also includes an outstanding chapter that discusses legal aspects related to US software development, a must-read for anyone interested in selling computer software. Although it might seem irrelevant for testers to be aware of these aspects, it's indispensable for their companies (see Laurianne McLaughlin, “Buggy Software: Can New Liability Rules Help Quality?” *IEEE Software*, Sept./Oct. 2003).

### For programmers

Programmers might think that the topics *Testing Computer Software* covers don't concern them, but they are important. In fact, any professional programmer who strives to deserve that qualification should be conscious of the role testers play and understand their apparently destructive stake in the software development process. *Testing Computer Software* is therefore a good resource for programmers, although they might be more interested in Kent Beck's *Test-Driven Development: By Example*.

This relatively short book describes a technique whose goal is obtaining “clean code that works” (a quote attributed to Ron Jeffries). This tech-

## ONLINE REVIEWS

- “A Simple Explanation of C++ for Beginners” by Art Sedighi  
A review of *C++ Without Fear: A Beginner's Guide That Makes You Feel Smart* by Brian Overland.
- “Does Web Site Development Require Project Management?” by Mariá Bielíková  
A review of *Real Web Project Management: Case Studies and Best Practices from the Trenches* by Thomas J. Shelford and Gregory A. Remillard.

[www.computer.org/software/bookshelf](http://www.computer.org/software/bookshelf)

nique consists of writing automated tests that show how your code should work, coding only when a test fails, and eliminating duplication through refactorings (changes in the code's internal structure that don't affect its external behavior). Test-driven development, also known as test-first programming, isn't another testing technique but "a technique for structuring all the activities of development." In fact, TDD reduces to the following short cycle: write a test, make it run (write the code you need to pass the test), and make it right (refactor to eliminate duplication).

Kent Beck clearly explains TDD with two complete examples. Afterward, he introduces the xUnit family of testing frameworks. He concludes the book by describing what tests to write and how to use xUnit to write them. This section also includes some information about design patterns and refactoring—two essential ingredients for successful TDD. Although it's concise and even sketchy, *Test-Driven Development: By Example* provides a nice introduction to TDD for those interested in approaching software development from an unconventional perspective.

You should note, however, that TDD doesn't cover all of a software development project's testing. It just provides a useful way to organize coding. Its iterations, with their frequent test executions, inspire confidence for programmers who can try refactorings they wouldn't dare without the safety net that automated tests provide. Moreover, although Beck doesn't emphasize it enough, test creation before implementation forces programmers to design the interface of the module they're coding before they write a single line of code. That means that interfaces drive implementation and not the other way around, which tends to cause overly complex interfaces.

### For everyone

Regardless of whether your main occupation is as a tester, programmer, or even a manager, *Lessons Learned in Software Testing* will give you insight on software testing. Its authors are

three experienced testers and testing project managers that make their points from their own hard-won perspectives. The book contains 293 short discussions on almost every testing-related topic you can imagine. Its advice reminds me of Robert Glass's *Facts and Fallacies of Software Engineering* (Addison-Wesley, 2002). As Glass does in *Facts and Fallacies*, the authors reveal common assumptions and misunderstandings that could be too costly for your own projects.

*Lessons Learned in Software Testing* isn't a textbook or a comprehensive software testing guide. The other two books could fit that role, depending on whether you're a tester or a programmer. *Lessons Learned in Software Testing* doesn't explain particular techniques in exacting detail, although it certainly discusses key issues around them.

*Lessons Learned in Software Testing* is about what the authors call *context-driven testing*, or practices that are well-suited to particular circumstances. You'll find interesting comments about the tester's role and way of thinking, using bug reports correctly, and diversity's importance to software testing techniques. In fact, "tunnel vision is the great occupational hazard of testing" (page 259). You'll also learn about the relationships between software testing and the scientific method and discover a critical perspective on automated testing. This is especially useful for TDD followers because automated tests complement but don't replace manual tests.

Like *Testing Computer Software*, *Lessons Learned in Software Testing* is full of social issues because the testing

**You might already know  
some of the lessons  
learned, but you'll  
certainly find fresh  
insights.**

process affects project team members' relationships. In fact, a context-driven school exists that, given its main focus on people instead of processes, we could view as software testing's "agile manifesto" (see [www.context-driven-testing.com](http://www.context-driven-testing.com)).

*Testing Computer Software* and *Test-Driven Development: By Example* are both good resources, although you might feel more inclined to one or the other depending on your personal interests. *Lessons Learned in Software Testing* is a must-read if you want to understand what software testing is and what it implies. You might already know some of the lessons learned, but you'll certainly find fresh insights. At the very least, you'll have food for thought.

**Fernando Berzal** is an assistant professor in the Department of Computer Science and AI at the University of Granada, Spain, and cofounder of iKor Consulting, a small IT services and consulting firm. Contact him at [berzal@acm.org](mailto:berzal@acm.org).

## Making up the Numbers

### Christof Ebert

**Five Core Metrics: The Intelligence behind Successful Software Management** by Lawrence H. Putnam and Ware Myers, Dorset House, 2003, ISBN 0-932633-55-2, 328 pp., US\$43.95.

W. Edwards Deming said, "In God we trust—all others bring data," and this simple statement holds a lot of truth. We see two trends related to the data from software projects: not having the right measurements and not using them correctly. In this context, Lawrence Putnam and Ware Myers' latest book is an eye-opener. It's their fourth book together—Putnam being the estimation expert (Remember the "Putnam formula" from estimation class?) and Myers being an author and consultant. Both are seasoned measurement experts, so I waited with curiosity to see what new information they'd tell

us. In short, they essentially repeat their previous message.

### Good explanations and insight

Clearly, the book's message is that you don't need many metrics to control your software projects. Rather, it's more relevant to use a few standardized metrics consistently and effectively. As such, the authors chose the book's title well as they only discuss five core metrics. That's less than the Software Engineering Institute's core metrics, but they target the same scope. The authors demonstrate over some 300 pages how project managers can control projects with these five core metrics—namely, time, effort, size, reliability, and process productivity. The last metric is the most interesting—who hasn't struggled with productivity measurement and improvement? Putnam and Myers carefully and exhaustively explain why conventional productivity metrics—such as lines of code or function points—are neither precise nor useful to measure productivity. Starting with insight into software process improvement and the Capability Maturity Model, they explain how process and productivity are related, which explains the “process productivity” label. Needless to say, the five core metrics all relate to each other using the updated and instrumented Putnam formula, which the authors explain well.

### Incomplete theory and poor advice

The book is easy to read—sometimes too easy because theory and explanations are vague. For instance, the authors mention the Putnam equation (or software equation, as they ambitiously call it) just out of the blue. Even the original article from the '70s is more precise in this respect because it also provided the reasoning behind the equation. They don't compare or evaluate the Putnam equation against Co-omo or other estimation models. What the authors consider competitive (after all, Putnam heads an enterprise that sells his proprietary estimation tool), they simply neglect. Although the underlying history data is exten-

sive, it comes primarily from large projects. Thus, the authors' recommendation to “hold projects in two years” looks odd in times of agile and iterative project layouts. The focus on folkloric and anecdotal elements (such as relating software project overtime to “naps at the desk,” “pizza parties,” or “activities better suited to a men's magazine”) and the easy readability too often mean that the authors omit necessary background information. For example, they introduce statistical control as a phrase but ignore the theory behind it. The authors have good reasons for putting the five core metrics in the layout as explained in the book; unfortunately for novice readers, they don't mention any goal-oriented methodologies to explain how they derived those metrics.

While the book's first half is too lightweight, the second half compensates for it, with the restrictions I've mentioned. Specifically, Appendix B is impressive, summarizing the data behind the book. Unfortunately, the timeline stops in 2000, which seems odd for a book published in 2003. It's no wonder that the data doesn't cover many recent evolutions and positive results in terms of improvements to software management (for instance, as reported in the annual CHAOS reports).

### Useful to a point

If you've never read a software metrics book, I could recommend *Five Core Metrics* as a starting point. You certainly won't get lost in too much measurement theory, and it stays focused on the five core metrics. The book explains well how to use them in different situations, such as controlling projects or suppliers. However, be careful not to read it as your only measurement book. Too many things are simply missing. Combine it with a popular measurement reference book—such as Norman Fenton and Shari Lawrence Pfleeger's *Software Metrics: A Rigorous and Practical Approach* (Course Technology, 1998)—and you'll know where to draw the lines. ☞

**Christof Ebert** is Alcatel's director of R&D processes and tools. Contact him at [christof.ebert@alcatel.com](mailto:christof.ebert@alcatel.com).

## How to Reach Us

### Writers

For detailed information on submitting articles, write for our Editorial Guidelines ([software@computer.org](mailto:software@computer.org)) or access [www.computer.org/software/author.htm](http://www.computer.org/software/author.htm).

### Letters to the Editor

Send letters to

Editor, *IEEE Software*  
10662 Los Vaqueros Circle  
Los Alamitos, CA 90720  
[software@computer.org](mailto:software@computer.org)

Please provide an email address or daytime phone number with your letter.

### On the Web

Access [www.computer.org/software](http://www.computer.org/software) for information about *IEEE Software*.

### Subscribe

Visit [www.computer.org/subscribe](http://www.computer.org/subscribe).

### Subscription Change of Address

Send change-of-address requests for magazine subscriptions to [address.change@ieee.org](mailto:address.change@ieee.org). Be sure to specify *IEEE Software*.

### Membership Change of Address

Send change-of-address requests for IEEE and Computer Society membership to [member.services@ieee.org](mailto:member.services@ieee.org).

### Missing or Damaged Copies

If you are missing an issue or you received a damaged copy, contact [help@computer.org](mailto:help@computer.org).

### Reprints of Articles

For price information or to order reprints, send email to [software@computer.org](mailto:software@computer.org) or fax +1 714 821 4010.

### Reprint Permission

To obtain permission to reprint an article, contact the Intellectual Property Rights Office at [copyrights@ieee.org](mailto:copyrights@ieee.org).