book reviews

ELIZABETH ZWICKY, MING CHOW, AND RIK FARROW

CODE CRAFT: THE PRACTICE OF WRITING EXCELLENT CODE

Pete Goodliffe

No Starch Press, 2007. 558 pages.
ISBN 1-59327-119-0

It doesn't take that much programming to discover that programming is harder than it seems. It's not really that hard to turn out code that works, for some definition of "works." For instance, I once worked for a programmer who insisted that her task was done when the code compiled without error, and anything further was debugging, which could be undertaken by somebody else—an attitude I didn't discover until she declared something done when it did not in fact run at all, even a little bit. In a happy twist of fate, she later ended up working for a friend's least favorite boss, and in our more twisted and bitter moments we take joy in imagining their working relationship.

If you want to be a good programmer and not end up on the wrong side of stories like this, you need to worry about a lot more than getting the code to basically fulfill its intended function. You have to be able to pass it on to another programmer. It has to survive rounds of modification. It has to grow and change. You have to work with other programmers. *Code Craft* is meant to help you achieve all those extra goals, and from my point of view, it does a pretty good job. It also has a monkey cartoon in every chapter, just to sweeten the pot (and although the topics are dense, the writing is clear and vivid enough to keep you going without the monkeys).

This is a great book for somebody who has basic technical competence but wants to learn how to make that fit into the bigger picture. I don't agree with every detail, but I agree with all of the author's main points, and I think a programmer who absorbed them would be much more fun to work with than one who disagreed with them.

The book does not mention installation and administration issues, which are close to the top of my "Ways programmers become loathed" list, but it's primarily concerned with keeping other programmers from loathing you. (It doesn't suggest firmly that unless you are unusually talented or a GUI designer moonlighting as a programmer, your user interfaces will be horrible unless you get help, either, and that's something lots of programmers need to hear.) It does mention that security is important and requires thought, but it omits the all-important warning "Don't design your own crypto." Perhaps what it's really missing is a big list headed "Things you might think you know something about but almost certainly don't."

HEAD FIRST OBJECT-ORIENTED ANALYSIS AND DESIGN

Brett D. McLaughlin, Gary Pollice, and David West

O'Reilly, 2007. 589 pages.
ISBN 0-596-00867-8

One of the blurbs for *Code Craft* is from a four-year-old (with a last name suspiciously like the author's), who likes the monkeys best. Let me tell you, she'd love the Head First series. In fact, I had to hide while reading it, because my daughter is currently passionate about dogs and speech bubbles (or thought balloons), and there are a lot of examples that combine the two. It turns out that she keeps asking you to read what people are thinking even after it turns out to be "Look at all those Strings! That's terrible! Shouldn't we use objects or constants instead?" or even less comprehensible stuff, over and over again. She was entranced. My husband, however, was instantly suspicious. Dogs and thought balloons are not a combination that makes him think "Serious programming book." (I haven't enquired about monkey cartoons. I think it is entirely likely that he finds them more suitable. The relationship between monkeys and programmers is all too evident to anyone who deals with programmers regularly. Don't throw things! I program for a living at the moment.)

Supposedly, all this hullabaloo that delights the small child and makes the adults wary is scientifically supported as a way to keep you from turning off your brain and doing the sort of "smile and nod" routine that gets you through cocktail parties but leaves you at the end of a technical book without actually having learned anything. I'm not sure, but I didn't find it unfortunately distracting, and I was occasionally moved to actually do parts of the exercises, and I am the world's worst person at doing exercises. Threats and pleading on the part of the author almost never move me to pull out a pencil—normally only the hope that

I will discover some terrible error will move me to even read exercises. But apparently in the war between "Don't do homework" and "Do puzzles," doing puzzles does sometimes win.

Along the way, I did absorb some of the principles and terminology of object oriented design and analysis; I probably would have gotten more if I'd stuck with the exercises and I weren't currently programming in an in-house language whose eccentricities preclude the use of many lovely design principles. This would make a particularly nice introduction for somebody who doesn't see the point. It does a good job of showing why design matters and how design issues relate to real-world programming.

ESSENTIAL CVS, 2ND EDITION: VERSION CONTROL AND SOURCE-CODE MANAGEMENT

Jennifer Vesperman

O'Reilly, 2007. 395 pages.
ISBN 0-596-52703-9

I admit to a certain sense of relief that this is a very straightforward book. No monkeys. No dogs. Either you have a burning need for it (e.g., you are trying to figure out which end is up while using CVS) or you don't. If you need to know more about CVS, this is a good, explanatory reference that will keep you from the sort of hazy hand-waving incantations that go on all the time at my place of work. (I blush to admit that I'd been using the contradictory option combination "-d -P" for quite some time because, well, that's what somebody told me to do and nothing bad happened. Also it was keeping the tigers away from my desk, apparently.) The book does a nice job of explaining differences between releases (which is important in a heterogeneous environment, where my CVS has all the latest flashy features, the servers have almost all of them, and alas, half my colleagues are in the dark ages).

If you aren't deeply interested in CVS, go read something else.

THE ART OF SOFTWARE SECURITY TESTING: IDENTIFYING SOFTWARE SECURITY FLAWS

Chris Wysopal, Lucas Nelson, Dino Dai Zovi, and Elfriede Dustin

Symantec Press, 2007. 250 pages.
ISBN 0-321-30486-1

It's nice to see a book that talks about software security testing in a sane and sober way. This book is intended for people testing their own software, and it discusses attack tools in the testing context, with a brief discussion of how software security and its testing fit into the development model. Oddly, it doesn't talk much about what I've always found to be the hardest part, convincing developers to care, although it does address it indirectly by talking about the cost of fixing security flaws at various points in the process (e.g., it's a lot cheaper to notice you need encryption before you ship a million copies and end up on the front page of a newspaper).

On the whole, I found the book unsatisfying. It's got a bunch of good information, but I'm not sure who the audience is. It seems to be meant for testers who don't have a security background, but there's really not enough information about security to let them test things effectively and know what's an important vulnerability and what's not.

There's a completely gratuitous ad for a Symantec product thrown in; I'm sure that Symantec's vulnerability database is a lovely thing, but what exactly has it got to do with patch management for your product, where you're trying to release patches, not figure out what patches you need to install?

HOW TO BREAK WEB SOFTWARE: FUNCTIONAL AND SECURITY TESTING OF WEB APPLICATIONS AND WEB SERVICES

Mike Andrews and James A. Whittaker

Addison-Wesley Professional, 2006.
240 pages. ISBN 0321369440

REVIEWED BY MING CHOW

As the number of Web applications increases, so do the risks of exposing the most critical business and personal data. Despite the growing acceptance of application security, security testing in the QA process is still largely lax. Glancing at this title, one would have the impression that it resembles one of the "Hacking Exposed" books. However, Andrews and Whittaker did a very good job in not doing exactly that.

This book covers all the most current attacks and issues pertaining to Web applications and services. The attacks are well organized into specific categories: client-based, state-based, user-specific input, language-based, server-based, and authentication. Attacks such as SQL injection, cross-site scripting, buffer-overflows, and even fake cryptography are discussed. For each attack, the "when to apply attack," "how to conduct attack," and "how to protect against attack" are described with very good illustrations using code or screenshots. The book is largely independent of language choice. However, the book delves into using a plethora of tools (e.g., Witko, NitkoParos, SSLDigger) for security testing. Finally, the book concludes with a discussion of privacy issues and threats in Web services.

The primary focus of this book is on testing. This is not a book on

how to exploit servers or applications in gory detail. This is also not a book on how to write secure code, or how to design secure software. However, it gives a concise overview of all the current problems in Web applications and services. This book is handy for anyone working in software QA. It also serves as a good introduction to Web application security for new developers.

INSIDE THE MACHINE: AN ILLUSTRATED INTRODUCTION TO MICROPROCESSORS AND COMPUTER ARCHITECTURE

Jon Stokes

No Starch Press, 2006. 320 pages.
ISBN 1-59327-104-2

REVIEWED BY RIK FARROW

Have you ever wondered just what Intel has been doing to make its processors run faster and cooler? Stokes provides an illustrated view into key areas of processor technology that explains the big issues in processor design. I had a good idea of what pipelining and superscalar meant before I started reading this book; I had actually learned about pipelining in a hardware design course in the late 1970s. But Stokes makes it easy to understand what these terms mean, and what effect they are supposed to have on processor performance.

Much of this book is based on articles published previously by Stokes at arstechnica.com, and if you have been following his articles, some of this material will be familiar. Chapters have been added to the beginning to fill in some basics and terminology, for example, as well as providing some explanation of the purpose of cache. The writing is clear, and the four color diagrams make a difference in coming to grips with a complex topic.

Stokes focuses on just two processor families, Intel x86 and IBM Power RISC (used in earlier Apple Macs). AMD barely gets mentioned during a discussion of 64-bit features. But that explanation did make it clear why you must have a 64-bit operating system to use these features and that you can still run 32-bit programs, that is, you are not locked into 64-bit-wide datapaths, in x86-64.

By the end of the book, I understood a primary difference between Core Duo and Core 2 Duo (the latter have wider internal data paths for doubles, which were missing in previous Intel desktop-targeted processors), as well as what MMX, SSE, and SSE2 are supposed to do. Reading this book should help you choose the right processor, as manufacturers, such as Intel, target their designs for particular workloads, and buying the right processor, with the right amount of cache, can really make a difference.

As Stokes himself writes, this is *not* a college textbook on computer architecture and related software (compilers and APIs). For that, you want Patterson and Hennessy's *Computer Organization and Design: Third Edition* (Morgan-Kaufman, 2004).