Book Reviews

ELIZABETH ZWICKY, WITH MARK LAMOURINE AND RIK FARROW

Naked Statistics: Stripping the Dread from the Data

Charles Wheelan
W. W. Norton and Company, 2013. 260 pp.
ISBN 978-0-393-07195-5
Reviewed by Elizabeth Zwicky

I am a little bit obsessed with introductory statistics books, in my attempt to convince the rest of the world that the most minimal grasp of mathematics will in fact combine with a tiny amount of statistics to produce all sorts of impressive-looking abilities. This includes not only the ever-popular ability to figure out how the news is lying to you, but also all kinds of everyday magic in trying to design, build, debug, or maintain computer systems.

The author of *Naked Statistics* shares this opinion, and so has written a book which aims at explaining some reasonably complex and subtle statistical concepts without requiring you to do any noticeable amount of math. It does include the math, should you want to follow along, but it hides it at the end of the chapters for easier skipping by people who are not enthusiastic about numbers.

If you have been resisting statistics because it seems to be made up of complex and incomprehensible mathematical formulae with the occasional curve, this book may change your mind. You will probably also enjoy it if you vaguely remember a statistics course in college but aren't quite sure how it relates to things you might care about. It's not directly related to computers, and if you find yourself needing to calculate your own statistics, you might end up wanting a more number-heavy introduction. (Or not. It's surprising how often the numbers are less important than the ability to say, for instance, "Shouldn't we be comparing these to some population?" or "Are you sure this correlation is relevant?")

Algorithms in a Nutshell

George T. Heineman, Gary Pollice, Stanley Selkow
O'Reilly, 2009. 335 pp.
ISBN 978-0-596-51624-6
Reviewed by Elizabeth Zwicky

Algorithms in a Nutshell also attempts to bring a difficult and useful domain to non-specialists. It doesn't do as good a job at avoiding looking and sounding like a mathematics text, although it does try. I'm still looking for an unthreatening algorithms book.

On the other hand, take their word for it and mine: at some point you are going to need to write code that does searches, or sorts, or looks through a problem space. Or you are going to need to deal with somebody else's code for these things that is dying horribly. At that point, knowing the best and worst cases for different algorithms is going to be important. This is as practical and friendly an introduction as you're going to get, heavy on the examples.

It's not going to substitute for a good computer science education, but it will remind you of the classes you've forgotten or bail you out when you run up against situations where your code—or somebody else's—mystifyingly fails to obey your expectations. And they include examples of how you evaluate algorithms, too.

Don't skip the ending chapters, which include some useful general principles, several of which are more elegant versions of my favorites; don't write your own implementation of any basic algorithm if you can help it, and when faced with a problem best solved with graph theory, immediately turn it into a simpler problem that no longer involves graphs.

The Art of Readable Code

Dustin Boswell and Trevor Foucher
O'Reilly, 2011. 184 pp.
ISBN 978-0-596-80229-5
Reviewed by Elizabeth Zwicky

I am imagining my college roommate at the moment, a guy who was a professional programmer and scornful of the idea of a computer science degree (it was a rental, not a dorm). He eventually learned, at the cost of a weekend of misery, that we were not kidding about sort algorithms mattering, but he would have rolled on the floor in hysterics at the idea of reading a book that has two entire chapters on naming variables and two more on how to write comments. That statistics stuff, it makes some kind of sense, but really, nobody with any sense worries about petty things like how to name variables.

We were young then, and stupid, and many years have come and gone. In that time, I've learned a lot about readable and unreadable code, and have many of my own rules of thumb (including one the authors missed: "Everybody is still basically a small child. Avoid variable names that are suggestive or naughty-sounding."). Nonetheless, I found myself saying, "Ah, yes, that makes sense—I never thought about that" from time to time, and "Oh, that explains some intuitions I never quite figured out" quite frequently.

Take my word for it; it matters how you name your variables. You can find this out from a book, or you can spend a lot of extra time debugging. The book is cheaper. Don't worry; it's not like That Guy who insists there is one true way to name variables and one true way to indent things. It's sensible, flexible advice, with examples to show you why you care.

On the other hand, if you were hoping somebody would just tell you the answer and then you could crank out readable code, this book will disappoint you, and it has to. Readability, in anything, is difficult and requires careful thought about readers. This is good advice that will help you do that thinking, but you're still going to have to work.

Living with Complexity

Donald A. Norman
MIT Press, 2011. 265 pp.
ISBN 978-0-262-01486-1
Reviewed by Elizabeth Zwicky

Living with Complexity is a book about our attitudes toward complexity and technology, which are much maligned and yet central to our day-to-day experiences.

Norman argues that our immediate opinions are wrong. He offers some strategies for both individuals and designers to deal with complexity, and he suggests new ways of thinking about issues around complexity.

There are several Donald Norman books that I love very much. This one I think is nice enough, but sadly, not earthshaking. It makes several good points (complexity is not inherently bad; social factors drive much complexity; services are often overlooked; and how is it that we can still be this bad at power strips and projectors?).

Ultimately, though, it fails to come together into a coherent whole. I prefer *The Design of Everyday Things,* but if you can't lay hands on it and haven't encountered Donald Norman before, *Living with Complexity* would certainly be a worthwhile read. It might seem more captivating to me if I hadn't been familiar with his earlier work.

MapReduce Design Patterns

Donald Miner and Adam Shook
O'Reilly, 2012. 227 pp.
ISBN 978-1-449-32717-0
Reviewed by Elizabeth Zwicky

MapReduce is the underlying technology for most companies dealing with big data. If you want to throw around serious amounts of data—you don't just want to use the cloud, you want to be the cloud—you are going to end up using MapReduce. In many situations, you're going to use it with some sort of insulation, using a language like Pig designed to hide what's going on underneath. But there's a limit to how far that will take you, and programming for MapReduce requires some twists in how you may be used to working.

MapReduce Design Patterns is a good place to start if you need to work directly in MapReduce for some reason. If MapReduce is further below you, you may not be interested until you start pushing the envelope, at which point you need to think more carefully about what's going on. And, quite possibly, to drop into straight MapReduce.

I use Pig a lot, and pure MapReduce occasionally, so I was familiar with the basic patterns, but I still found some new and interesting ideas.

Getting Started with Raspberry Pi

Matt Richardson and Shawn Wallace
O'Reilly Media, 2013. 176 pp.
ISBN 978-1-449-34421-4
Reviewed by Mark Lamourine

When I describe a Raspberry Pi to most people, the first question I usually get is, "But what does it do?" *Getting Started with Raspberry Pi,* O'Reilly's introduction to learning computing with the Raspberry Pi, is one attempt to answer that question.

The Raspberry Pi has become the hot small hobbyist computer in the last year. It's meant to be a tool for people who want to learn and experiment with programming, network services, and robotics. The previous leader in this space was the Arduino, and people have asked me why the world needs another hobbyist computer, but there's really no comparison. The Pi is a fully outfitted general purpose computer, while the Arduino is a programmable microcontroller. This means that the Pi is suited to different kinds of projects than the Arduino. In fact, there are a number of projects that use the Pi to program the Arduino. The two are really complimentary.

The O'Reilly "Make" books are aimed at people who mean to get their hands dirty. While they are often directed at beginners, they don't hand-hold or subject readers to long lectures on fundamentals. Instead they focus on small projects with achievable goals and then leave the reader with tips for further reading.

The first chapter walks the reader through setting up their Pi. The following chapters gradually introduce Linux, Python, and then move on to hardware projects using the Raspberry Pi GPIO pins directly, or adding an Arduino. Each chapter finishes with a "Going Further" section that includes references (and in the eBook I read, links) to additional resources on the topic.

The format of the book follows O'Reilly's "Make" imprint style. This is the signature red and blue logo, big simple bold text on a crisp white background, and hand-drawn graphics. It's easy to read and very comfortable and welcoming.

This book is very well suited to the adventurous beginner. The chapters are clear and complete. It is a good idea to keep a Web browser and search engine handy. The range of topics means that there might even be something new for an experienced server administrator who might not have had a chance to play with sensors or robotics.

The low cost of the Raspberry Pi has meant that they are being purchased by (and being given as gifts to) people who have only ever had the slightest exposure to computers outside the canned Windows or Mac OS experience. Will it be the Mountains of Robotics, the Seaside of Media, the Caves of Programming, or something else entirely? *Getting Started with Raspberry Pi* is the signpost at the crossroads.

Practical Vim: Edit Text at the Speed of Thought
Drew Neil
The Pragmatic Bookshelf, 2012. 311 pp.
ISBN 978-1-93435-698-2
Reviewed by Rik Farrow

Elsewhere in the issue, Dave Josephsen refers to the argument of whether to use vi or Emacs as a religious one. For myself, choosing to use vi was more a pragmatic decision: vi was found on all of the many systems I was using at the time. I learned vi, and continued to learn new tricks, until I thought I had mastered vi. That was 25 years ago. Over the last five years, I started noticing differences in how vi worked—for example, if I happened to pass a directory instead of a file to open. I thought the new behavior was better, displaying the directory's contents and then getting to choose the file I wanted to edit. Then I noticed other things, such as previously unmapped keys were now mapped, and *strange* things, like the screen was split.

What had happened was that vi, written by Bill Joy in the early '80s, had been replaced by vim. And when I noticed this book, I decided it was time to learn about the new tool.

I got away without knowing about vim for years because it works pretty much like vi—it's just that vim does a whole lot more. *Practical Vim* is organized as a series of "Tips" within each chapter, but starts out with several chapters of very basic vim. The author encourages more advanced users (ahem) to skip around, which I immediately started doing. I learned that vim has a special register that allows me to solve simple equations and insert the results inline. I found out how to *intentionally* split the screen, and what visual block is all about.

The book works well: the instructions are clear, examples easy to follow, and everything I tried worked. My only complaint is about vim, not the book. Now I have another set of keystrokes to memorize so I can learn all the new features. *Practical Vim* does not teach you about scripting, another facet of vim, although it does use scripting in several examples. There is much to learn.

If you have been a vi user, and have noticed something different, I encourage you to learn about vim. Or buy this book, as it can make the experience less painful and more fun.