Book Reviews

ELIZABETH ZWICKY, MARK LAMOURINE, AND MELISSA GRAY

Peopleware

Tom DeMarco and Timothy Lister
Addison Wesley, 2013. 238 pp.
ISBN 978-0-321-93411-6

Reviewed by Elizabeth Zwicky

Peopleware is an old favorite of mine, and I approached this edition with some trepidation, the way you approach anything you loved when younger that has now been updated; will it turn out to have lost its luster, either through age or through savage updating? On the whole, I was very happy. The original copyright shown is 1987, and in the intervening roughly quarter-century, a lot has changed, but the fundamentals of programming and managing people have not. The update manages to remove most of the dated references and adds a good bit of purely new material.

Peopleware is an introduction to the human side of managing technology teams. It is eminently readable—it comes in short, vivid chunks that say things programmers want to hear in terms that management can understand. If you are feeling that there is something fundamentally missing from the practice of technology management, this will fill that gap and fire you up.

I'm somewhat sad that after this long, the humanistic approach found here still feels fresh, startling, and avant-garde. Paying more attention to human issues than new technologies, like personal jet packs and hover cars, seems destined to remain the wave of the future. And yet there are signs of hope—when a phone rings audibly in my office, people are startled and displeased. It's a rare event, rare enough that the last time a repetitive noise went on for a while, one of my colleagues leapt up angrily to search out and silence the phone, only to realize that the rest of the office was laughing at him. The annoying noise was in fact a crow on the windowsill. The good news here is that our office environment is both quiet and near a window; the bad news is that the entire team was within sight and earshot of the crow and the ensuing search. So there's still work for *Peopleware* to do.

Adaptive Software Development

James A. Highsmith
Addison Wesley, 2000. 348 pp.
ISBN-0-932633-40-4

Reviewed by Elizabeth Zwicky

Somehow this crept onto a list of new releases, so I was puzzled to read through an entire book on software development practices for rapidly changing environments that never used the terms "Agile" or "Extreme" as we now know them. It's still a worthwhile book, with a detailed explanation of a practical and human-centered approach to development in high-change environments. It is quite kind to the waterfall model, suggesting places it is appropriate and ways to gently move people away from it. And it is heavily influenced by *Peopleware* while being much more traditional in tone and format.

This would be a great bridge book for somebody who wants or needs to move to a more flexible style of managing projects, but would like to do so without overt, radical breaks with tradition.

The Practice of Network Security Monitoring

Richard Bejtlich
No Starch Press, 2013. 334 pp.
ISBN 978-1-59327-509-9

Reviewed by Elizabeth Zwicky

This book will tell you how to install Security Onion and its add-ons, how to work with those tools (including tricks, traps, and subtleties involved), and it provides significant discussion of how to place monitoring taps. Bejtlich provides some advice on how you keep track of what's going on when you don't know what you're dealing with. These are all significant challenges for new network security administrators.

I feel convinced that this book would help me set up a network security monitoring system based on open source systems and use it to improve the security of pretty much any network. On the other hand, this is a task where I don't really need all that much help—I know a lot about how networks work and about the practicalities of securing them. I'm less convinced that somebody without all that background would find it sufficient.

What it doesn't talk about, except in vague and abstract terms, is the actual practice of network security monitoring—what alerts are important? which ones are not? There's a lot of good information here, but it doesn't quite jell into a clear problem statement and answer, and it isn't quite enough for a security novice.

Graph Databases

Ian Robinson, Jim Webber, and Emil Eifrem
O'Reilly Media, 2013. 200 pp.
ISBN 978-1-449-35626-2

Reviewed by Elizabeth Zwicky

In a world where we all used graph databases, you could just write a book about how to use them. But most of us don't, so this is a relatively broad introduction, covering what graph databases are, why you might want one, and how you would design, query, and optimize one.

Simplifying somewhat further than is advisable, a graph database allows you to query data by talking about objects and relationships instead of by talking about rows and fields. Graph databases are considerably more efficient at certain kinds of queries than traditional databases. If you want to ask "What did Customer 43 order recently?" any old database will do. If you want to ask "What might Customer 43 order next?" you will rapidly find yourself asking, "What other customers ordered the same items as Customer 43?" and a suitably designed graph database will vastly improve your experience. (Or so the authors claim, very believably.)

If you're in a position to implement a system using new database technology, graph databases are an interesting tool to have available to you, and this introduction, while it clearly doesn't cover all the corners of the space, should get you started.

Advanced Programming in the UNIX Environment, 3rd Edition

W. Richard Stevens and Stephen A. Rago
Addison Wesley, 2013. 994 pp.
ISBN 978-0-321-63773-4

Reviewed by Mark Lamourine

A lot has changed since I first read the late Richard Stevens' *Advanced Programming in the Unix Environment*. Stephen Rago has just released his second update. There are very few books I enjoy rereading and fewer still tech books, but I enjoyed this refresher course.

Advanced Programming describes the interface between programs and the kernel in a *NIX system. Even when a program uses higher level libraries, in the end this is what they come down to.

There are numerous tutorials on programming languages and programming in general. There are textbooks describing the *NIX kernel internals. Stevens and Rago don't just list the kernel system calls and their arguments. They illustrate their use and the behavior of the kernel in response. This gives the reader a sense not just of how to use each call, but when and why. It also gives them the ability to work backward from the behaviors of a system to the calls that would be the cause.

In 1993 the systems described were AT&T System V R4 and 4.3BSD. In 2013 Rago has added FreeBSD, Linux, MacOS, and Solaris 10 (arguing that while Solaris is derived from SYSVR4, it has 15 years of enhancements). While this might seem to add quite a bit, it seems that standardization has largely served its purpose. Variations still exist but they're not nearly as large as might be expected. The 3rd edition is almost 1,000 pages compared to 740 for the first edition, but Rago has added two sections on threading and one on network sockets to address topics that didn't exist in the early 1990s.

There are remarkably few actual system calls (functions that cause a process to switch from user to kernel mode). The remainder of the interfaces are known as system libraries and are generally built on top of the system calls. These are used to manage the core system resources (files, processes, threads, and memory) to communicate between processes (signals, semaphores, shared memory) and between systems and devices (serial I/O and networking). Stevens and Rago explore each of these in some depth, highlighting differences between operating system flavors.

The authors begin most chapters by explaining some aspect of a running *NIX system: files and I/O, processes and interprocess communication, errors and the process environment variables. They show what each feature is for, how it affects the operation of the system, and how programs interact with it. Only then do they introduce the system calls with realistic example code. Each chapter closes with a traditional summary and a set of exercise questions.

Stevens and Rago close the book with a couple of chapters that present complete uncontrived examples of real-world systems programming.

Advanced Programming is known as a classic for good reasons. The writing is clear and precise. The examples are detailed but to the point. The chapters follow a progression from topics that will likely be familiar and commonly used to those that may be more specialized or esoteric. The rationale for or history behind a design choice or variation is provided when it offers some insight into how a feature is to be used. This is one of the rare books that works both as an introduction and as a reference.

I've done a fair amount of systems level programming and I recommend *Advanced Programming* to pretty much anyone who programs *NIX systems seriously; however, I'm a system administrator by trade and avocation. There are two divided and vocal camps on the question of whether programming is required for system administration. I won't weigh in on the question of requirement, but I don't think it can hurt to at least learn how to

read C code. I think it can be a tremendous benefit to understand the kernel system calls, especially when tracing and debugging processes. I recommend *Advanced Programming* to anyone who's interested in understanding the interfaces between *NIX programs and the system that runs them.

The Go Programming Language Phrasebook

David Chisnall
Addison Wesley, 2012. 264 pp.
ISBN 978-0-321-81714-3

Reviewed by Mark Lamourine

In *The Go Programming Language Phrasebook* David Chisnall provides all of the information an experienced coder needs to begin experimenting with Go. He doesn't spend a lot of time on the minutiae of the language and libraries, deferring instead to other books and resources when a reader might want more detail. He concentrates on the features that make Go significant and on the idioms and coding patterns that make the best use of those features.

It turns out that Go is designed not to illustrate some new programming paradigm, but in response to the known shortcomings of the aging C programming language in the context of system software development where it still dominates. Much of the Go syntax looks like C, but where it differs there is a reason. Usually the changes are meant to eliminate common coding errors or to decrease the complexity of implementing modern coding patterns. The most significant new features, "goroutines" and "channels," provide a cleaner means of implementing concurrency both on individual multicore computers and in networked distributed systems. It is also notable that, although Go is a compiled language, the development environment offers a way to run many programs from source code on the command line as if they were scripted.

The author avoids the worst impulses of writers of this kind of book. The phrasebook format can lead an author to provide a code snippet for every variation of every feature of every library. Chisnall focuses on writing about Go and uses the code snippets only to illustrate a point. He details not just how Go differs from other common languages but *why*. Because Go is meant to replace C, a low-level language, the machine details, such as the placement of structures in memory, will peek up through to the coder. Chisnall doesn't shy away from discussing how coding style and idiom will affect the behavior of the machine and how Go features contrast with other languages. Go is still a young language, and Chisnall informs the reader where there are caveats, gaps, or areas of continuing development that might make his examples obsolete.

In addition to the standard language primer and features (variables and types, scoping, objects, arrays, and collections) and the new features (goroutines and channels), Chisnall includes sections on working with the Go runtime environment, packaging and distributing code, and debugging. The one thing notably missing is any mention of a unit testing framework.

The Go Programming Language Phrasebook is an excellent introduction both to a new alternative for systems programming and a survey of the challenges faced by coders implementing modern concurrent and distributed applications. Because Go produces executable binaries for any modern OS and architecture, I will certainly consider trying it the next time I need to code a binary from scratch, and this book will be the first source I pick up.

Realm of Racket: Learn to Program One Game at a Time!

Matthias Felleison, David Van Horn, Conrad Barski, Forrest Bice, Rose DeMaio, Spencer Florence, Feng-Yun Mimi Lin, Scott Lindeman, Nicole Nussbaum, Eric Paterson, Ryan Plessner
No Starch Press, 2013. 294 pp.
ISBN 978-1-59327-491-7

Review by Mark Lamourine and Melissa Gray

I've read a number of books aimed at introducing software development to new readers, but I wouldn't have picked Lisp as a first language. *Realm of Racket* is an introductory text aimed at college freshman and written at least in part by students at Northeastern University. The students get top billing on the cover. The language is Racket, a derivative of Scheme, which is in turn a Lisp variant. The authors try to set an informal tone with comic strip artwork and a game and quest narrative which seemed to me to be a bit childish for the audience. It's been a long time since I was a college freshman.

Luckily I had a handy intern in the cube across from me, and she agreed to read it and give me her impressions. On reading them I had to reconsider my first take on the book. This is what she had to say:

The information is laid out in an accessible and engaging way. I think it would be effective and understandable for college freshmen regardless of previous programming experience. The story and cartoons are engaging. High-level material is clearly explained and given to the reader gradually in a way that builds on the previous chapters. As someone who has taken both high school and college intro programming courses, game-based examples and exercises are a good way to teach logic and decision-based programming. So I think this is a strong feature of this book.

The "read front to back" method this book employs might be bothersome for impatient students who would rather skip through a textbook by topic. However, if a student's goal is really to learn the material, this ends up being a great method because you can learn things in a logical order.

So, Melissa didn't seem to be as put off as I thought she'd be. When I asked her about it she shrugged and directed me back to the text and the teaching arch and I had to take a new look.

The chapter topics and sequence presented are not what I've come to expect for procedural languages, but are natural for Lisp. Variables, conditionals, and functions come first, but then the authors present recursion and lambdas before coming back to looping constructs and trees. They don't stop there, though, and this is where the youthful comic strip cover seems misleading. The authors continue, introducing more advanced topics, memoization, and lazy evaluation. The book closes with several chapters developing a simple distributed game using client-server constructs and messaging.

There's a lot packed into this book and it's not really aimed at the tweens that some other No Starch programming books have been, though I wouldn't hesitate to offer it to a motivated high school student. The DrRacket IDE runs on Windows, MacOS, and Linux so that students can begin work in whatever environment they are comfortable. The IDE is also fairly comprehensive, containing tools for interaction, development, and debugging. The Racket language includes module constructs that I don't remember seeing when I learned Scheme. DrRacket also provides a GUI library that I know wouldn't work on my VT100.

In the end I'm impressed. *Realm of Racket* and DrRacket both are well thought out and well suited to their tasks.

Nominating Committee for USENIX Board of Directors

The biennial election of the USENIX Board of Directors will be held in early 2014. The USENIX Board has appointed Margo Seltzer to serve as chair of the Nominating Committee. The composition of this committee and instructions on how to nominate individuals will be sent to USENIX members electronically and will be published on the USENIX Web site this fall.