Book Reviews

MARK LAMOURINE AND RIK FARROW

Functional Thinking

Neal Ford
O'Reilly Media Inc., 2014; 179 pages
ISBN 978-144936551-6

Reviewed by Mark Lamourine

Functional programming has, for a long time, been the realm of the theorists, the purists, and the AI specialists. Derived directly from the lambda calculus, the mathematical underpinnings of computation theory, functional programming has always felt like it required a different mental model. FP was an alien world. It didn't seem like the concepts could be applied without throwing out everything I know and adopting a new programming language.

While it's true that some languages make pure functional programming easier than others, many languages today provide the most fundamental feature of functional programming: functions as first class objects. This means that it's possible to apply the concepts of FP even in languages that are not pure functional languages. Ford does use languages, Scala, Groovy, and Clojure, that illustrate his ideas clearly, but he also demonstrates code in Java 7 and 8.

Ford really wants the reader to begin to look at certain classes of coding problems differently. Each chapter has a word or phrase that is used to guide the examination. I'm not sure how effective they are really but they may work for some.

In the first chapter, "Shift," Ford talks about recognizing list processing opportunities. He introduces the MapReduce pattern and filtering using chained list processing functions. Ford shows in following chapters how to use higher order functions and recursion to replace iteration ("Cede"), and how to use memoization to create "lazy" data structures to delay processing ("Smarter, not Harder"). In "Evolve" Ford introduces Clojure and the concept of replacing defined data structures with dynamic functions that both manipulate and represent the program state. The final two chapters, "Advance" and "Polyglot and Polyparadigm," introduce design patterns suited to functional programming and examples of mixed-style languages and programming practices.

I admit I'm not a convert to pure functional programming. In the examples given, the code is indeed often more concise than the comparable object oriented or imperative style. To my aging eye, the results often smack of cleverness, which can obscure the intent of the author. I've seen and written long strings of chained functions on lists, and they often seem to reach a point where the meaning is no longer evident to the reader.

I'm also not a fan of the convention of creating a new function for every possible variation of operation on some data structure. The results are an ever increasing catalog of minutely specialized functions that would otherwise be a clean set of methods on a class.

Given all that, I do have a newer appreciation of functional programming, and I'll keep an eye out for opportunities to apply what I've learned. *Functional Thinking* has given me a perspective on functional programming techniques and philosophy that I missed when I learned my first functional languages (Common Lisp and Scheme) in college. I do wish I'd had some of this perspective then.

SDN: Software Defined Networks

Thomas D. Nadeau and Ken Gray
O'Reilly Media Inc., 2013; 353 pages
ISBN 978-1-449-34230-2

Reviewed by Mark Lamourine

I pick up most books from O'Reilly today expecting to breeze through the introduction and the first few chapters at least. The authors of *SDN* made me work. The subtitle of the book is "An Authoritative Review of Network Programmability Technologies," and I think they live up to it.

Nadeau and Gray have a difficult task, too. To discuss the technologies that can be used to create and manage programmable networks, the reader needs first to have some understanding of the network components themselves. While many sysadmins are familiar with the use of Layer 2 (switching) and Layer 3 (IP, routing) devices and the data line protocols, fewer are familiar with the internal architecture and components of these devices. To make things more difficult, most of those components have proprietary names and acronyms or abbreviations.

The authors move very quickly through this first section and don't make many concessions to the networking novice. They introduce the concept of a "Data Plane," in which the network payload moves, and a "Control Plane," which defines the characteristics of the network. The key concept that drives the rest of the book is the idea of a distributed control plane. In hardware-defined networks, the control plane is restricted to the individual switch or router device. In a few kinds of device, blade or stacked switches and routers, the control plane is abstracted one level away, but never farther. The abstraction that software offers opens up the possibilities. In the extreme case, the control plane could be completely centralized. The topology variations and their effects on the construction of a network, with the benefits and flaws, fill the remainder of the chapter.

This first section lays the necessary groundwork, but I would have appreciated a bit more help with understanding the typical components, their relationships and interactions in a static hardware network. This is the base on which the rest of the book is built. I think they could have spent some more effort to make this truly complex topic clearer.

The second chapter introduces OpenFlow, which the authors use as a touchstone. It sets a baseline against which the rest of the software in the book is compared. OpenFlow is the result of the first attempt at a software-defined network. While it has gained support both from corporate and open source contributors, it is largely acknowledged to be insufficient to create a complete network by itself.

Nadeau and Gray proceed to move from the lowest level of network control up to building complete virtual network topologies on top of the same physical hardware that carries packets from one place to another. This is a technical review, so the authors list and describe the software capabilities and characteristics, not how to configure or use them in operations.

Software-defined networks are an incomplete and evolving subject. The body of the book alternates between lists of vendor and open source technologies and some discussion of how they fit into a programmed network. The end of each chapter is a short section in which the authors sum up what we've learned and where the gaps still are.

This book does a good job of illuminating the state of software-defined networks once the authors get past the details of the characteristics of distributed or centralized control. Unfortunately, the field has changed dramatically even in the year since it was published. The introduction (and withdrawal) of OpenStack Quantum and the stumbles of OpenStack Neutron are just a couple of the developments in the SDN space. There are signs sprinkled throughout the text that the authors recognize this and plan to issue updates, but it's not clear when.

Given the current movement toward cloud computing, I've concluded that the role of a sysadmin is expanding rather than contracting. The operator of an OpenStack or commercial cloud service will need to have at least some expertise in all of the traditional enterprise IT silos: Compute, Storage, and Networking. A book like *SDN* might be the best way for someone who needs to get a handle on the problems and possibilities to get conversant.

The Practice of Cloud System Administration
Thomas Limoncelli, Strata R. Chalup, Christina J. Hogan
Addison-Wesley, 2015, 524 pages
ISBN: 978-0-321-94318-7

Reviewed by Mark Lamourine

System administrators are not known for consensus and conformity. It doesn't take long for new admins to fall in love with a tool or a programming language (or to fall into hate). The Editor Wars are probably the most well known ongoing dividing line, but faults can appear around any choice we can make.

This is what makes the books by Limoncelli, Chalup, and Hogan (LC&H) so remarkable. If you ask most sysadmins what single book they should read, the answer will almost certainly be *The Practice of System and Network Administration.* They're going to have a harder time now, with the release of volume 2: *The Practice of Cloud System Administration.* (Just so you know, it's already known by the abbreviation TPOCSA.) I think this is likely to become a must-read.

One of the tenets of TPOCSA (and of all quality design) is "Keep it Simple." The authors present cloud administration in two parts. Pretty simple, eh? First, they define the characteristics of their ideal system, then they go on to describe the methods that they use to try to achieve that ideal.

When I say "describe the system" I mean that in a somewhat abstract way. LC&H aren't talking about which database is best or how much memory you need to render a movie frame. It turns out that all large-scale distributed systems have a set of common characteristics. These, along with the requirements for high reliability and robustness, have led to a set of best practices that have become generally accepted, largely because they have been shown to work. The hitch is that most of them seem counterintuitive and nearly all directly contradict standard practices of two decades ago.

In this section the authors also make clear the scope of what "system administration" means. Up until the advent of virtualization and ubiquitous high-speed networks, it meant OS installation and some network configuration. When the machine was ready it would be handed off to some application and operations team for the rest of the lifetime of a host. The SA tasks would probably include backups and periodic patching (or at least that's what many people thought). Today system administration and operations are largely synonymous. This union even has a word: DevOps (which is contentious, so I won't discuss it further here).

So we're talking about a large-scale distributed system. Whenever you have something big and made up of lots of parts, you inevitably have failures. Much of the rest of the book consists of ways to make that not matter, taking human nature and the "physics" of highly complex systems into account to make robust seamless services that run well even as they are changing.

Scanning over the chapter headings after the section break, I am struck by something that should have been obvious. This is a book about practices. The first section is really a glossary, a base of terminology and concepts on which to build. But what we build from them, the system that results, isn't just the cloud application. The infrastructure that LC&H are talking about here is as much a social one as it is technical. Each of the computational components is meant either to facilitate human communication or to remove painful, time-consuming, or error prone tasks.

System administrators are no longer just brick layers and janitors. They are involved in every phase of application life cycle from inception to a long continuously evolving life span. LC&H discuss the philosophy and practice of each phase, always considering that humans are expensive (and error prone) while computation is cheap. Automation, documentation, and monitoring are all reconsidered with an eye to minimizing drudgery and false rigor and replacing it with a min-set that will evaluate what's really important: comprehension and communication.

I read Gene Kim's *The Phoenix Project* not long after it came out, and while I smiled and nodded knowingly all the way through, it felt a little like a unicorn story. I thought, "This is nice, but no one in business is going to take a novel seriously as a model for business practice." Of course I was wrong, but I still think that something more is needed, not just a parable but a manual. The line where the authors cite Gene for "inspiration and encouragement" indicates that LC&H thought so, too.

There really wasn't much in this book that was new to me. I think much of what's here is already fairly common knowledge. What TPOCSA has done is to bring together in one place the accumulated body of knowledge that has been growing and changing since the birth of the Internet. Today's computer systems are a far cry from the mainframes, minicomputers, and PCs that dominated the 1990s. There have been a number of movements triggered by the changes since then: Agile development, the DevOps movement, Continuous Integration and Deployment. TPOCSA brings them all together and reminds us that the methodology, the philosophy, the ideology are not what matters. The system, running and serving reliably, is what matters. All of the rest are just means to that end.

So who should read it? I think anyone claiming to be a system administrator today should be conversant with what's here, but I think the bigger impact will come when we pass it to a colleague, whether a developer or a manager. There's a lot of confusion around what cloud computing means, and TPOCSA gives us a common base on which to build our systems and our processes.

What If? Serious Scientific Answers to Absurd Hypothetical Questions

Randall Munroe
Houghton, Mifflin, Harcourt, 2014; 305 pages
ISBN 978-0-544-27299-6

Reviewed by Rik Farrow

You are likely familiar with xkcd, the comic strip that uses stick characters to great effect. You may not have heard of another project by Munroe, where he answers questions submitted to him, using math and research, to provide sound answers to some very strange queries. I had encountered a couple of Munroe's posts while searching xkcd.org, looking for potential cartoons to decorate a *;login:* issue, and learned that he was publishing an entire book of answers.

Well, not quite. Munroe interspaces his answers with sets of questions that he wouldn't answer, adding more humor to his book. And like the Mythbusters, Munroe will often take a question, provide an answer, then scale the question up, to prove a point, like how many people with laser pointers would it take to light up the dark portion of the moon, or his hair dryer that has ten settings, each using an order of magnitude more power. Munroe uses scaling and statistics in ways that are effective.

I've used Munroe's *What If?* as a great way of taking a break away from my computer, and think it would be an appropriate gift to most geeks and/or scientists that you may have in your life. You might also be able to use Munroe's writing as inspiration for how to answer those ridiculous questions you may have been asked by your management, although I do advice caution in these circumstances.