BOOKS

Book Reviews

MARK LAMOURINE AND RIK FARROW

Effective Python: 90 Specific Ways to Write Better Python, 2nd Edition
Brett Slatkin
Pearson Education Inc., 2020, 444 pages
ISBN 978-0-13-485398-7

Reviewed by Mark Lamourine

In *Effective Python*, Slatkin offers a rather different twist on the cookbook format for programming references. In the conventional form, each chapter opens with a problem or a question. The body of the chapter then consists of a solution with some exposition. The premise is that the reader is learning the language features and capabilities. The recipes provide language-specific ways to achieve what are normally common goals.

Slatkin's approach is more of a catalog of best practices for the Python coder. The book is subtitled "90 Specific Ways to Write Better Python." Each of the 90 small "items" referred to in the subtitle opens with a recommendation. For example, Item 9: "Avoid `else` Blocks After `for` and `while` Loops." The main body of the item is a presentation of an argument for the recommendation. The arguments range from readability and performance to avoidance of common coding errors. Slatkin's arguments tend to follow a pattern. First he shows how the feature or construct is used commonly. He talks about why the typical usage makes sense at first and then how it can lead to problems. Only then does he offer his alternative, using new code fragments and explaining how the new code's behavior addresses the problems cited. Each item ends with a summary bullet list of things to remember.

The items are grouped into chapters thematically. Most are related to language features like lists, functions, or classes. The opening and closing chapters are more about idiom, style, and human behavior and are entitled, respectively, "Pythonic Thinking" and "Collaboration."

These two chapters directly express a thematic undercurrent that runs throughout the book: coding is a human endeavor and a craft and in every instance involves the judgment of the developer. Despite his recommendations favoring specific behaviors and constructs over others, Slatkin always appreciates why the common usage *is* common. His recommendations are always presented in a way that is meant to be persuasive rather than strident or proscriptive.

I started using Python with version 1.5, and version 2.x has been a staple for me since it was released in 2000. For me, version 3 was always "someday." I'm embarrassed to realize it's been 12 years. With the sunsetting of version 2 in January 2020 [1], it has become important not just to learn the differences, but to commit to version 3. The second edition of *Effective Python* treats only version 3, with none of the back references or porting comments that have been common for a decade.

There have been a couple of changes that I didn't really internalize. One was the introduction of the `bytes` and `str` types for representing strings. I understand the difference between ASCII byte strings and UTF-8, but the treatment in Python and the idiomatic usage have never become second nature. Slatkin addresses this as Item 3 in the first section. He shows how to recognize them and how to convert between them. More importantly he indicates *when* to convert between them and when to leave them alone.

I had been in the habit of using the `print()` function in Python 2 from the `__future__` module and the `str.format()` method instead of the formatting operator (%) for a long time. I was surprised to learn that there is a new string formatting method introduced in version 3.6 called *Literal String Interpolation*, or *f-strings*, for the prefix used to indicate one in the code. These work more like Jinja2 templating, where you use the name of the variable inline to resolve and replace the value in the string. What I really like about this item is the way Slatkin demonstrates the earlier methods, showing how it is easy to make errors with them. Finally, he demonstrates f-strings in a way that highlights how they resolve the problems.

Don't be fooled by my initial examples. *Effective Python* addresses some rather deep theoretical constructs as well. It has a fantastic treatment of generators, not just what they are and how to use them, but how they work and *why* to use them. Hint: avoid large in-memory arrays of computable values. Slatkin's treatments of metaclasses and concurrency also brought me some "aha!" moments. The references in these cases are provided for anyone who is learning about these topics for the first time and that's a good thing.

The only real quirk I noted with Slatkin's style is that it really requires you to properly read the text. In many reference works, once you're familiar with the topic you can skim to find just the bit of code you need. While none of the items here are particularly long or deep, the teaching style requires the reader to follow the narrative. I don't think that this is a problem, but it was an adjustment I had to make to get the most out of what I was reading.

Normally, I would encourage new coders to set something like this book aside until they had some experience and context to bring to it. In the case of *Effective Python*, I would consider suggesting they get it, skim the first few sections, and then set it aside. It will be there when they begin to ask the questions it tries to answer. It's a book I expect to return to.

Dependency Injection Principles, Practices, and Patterns

Steven van Deursen and Mark Seemann
Manning Publications, 2019, 522 pages
ISBN 978-1-61729-473-0

Reviewed by Mark Lamourine

I remember my confusion the first time I heard the term *dependency injection* (DI). I'd seen it used in some Ruby code with unit and functional tests, but I didn't know it had a name and didn't understand the formal basis for it. Since then I've spent significant time failing to create testable or flexible code using DI. Understanding DI has been on my back burner, and when I saw this book I had to see what DI is about.

"Principles, Practices and Patterns" is actually a pretty good description of the book. The authors are clearly fans of Martin Fowler, Robert Martin, and the *Design Patterns* [2] "Gang of Four." They make explicit reference to several design patterns that are extensively used to implement DI constructs. They also make good use of proper UML diagrams to illustrate object dependency relationships and life cycle. While prior knowledge of design patterns and UML isn't required, it will definitely help a reader understand the theory and the assertions the authors make about the structure of software and the effect that has on testability and maintainability.

At the end of Chapter 1 I found a paragraph that is easy to miss but critical to understanding this book. The goal of the authors is to help readers implement code designed with *loose coupling*. That is, code that depends on interface and API definitions of the code it uses rather than on the specific implementation. The authors' core assertion is that loose coupling is a generally desirable trait of well-designed systems. Dependency injection is just the technique they are offering to enable loose coupling. It is easy to lose track of that emphasis when trying to absorb the somewhat dense concepts that follow.

One thing becomes evident during the first three chapters: loosely coupled code looks more complex than tightly coupled code, at least at first look. In Chapters two and three, the authors show a simple three-layer application with a database, a user interface, and some business logic sandwiched in between. They do a good job of showing the options and decisions that lead to tightly coupled code. The design decisions are primarily a function of the desire for initial simplicity. They are natural and straightforward, based on the intent of the application.

In the following chapter the authors re-implement the application with a design in which the components are carefully decoupled. Each of the interacting classes defines an interface rather than just providing a function or method for callers. The design is significantly larger, increasing from four classes to nine and with three interfaces. The chapter is also twice as long. That extra text is used to explain the different considerations that are needed to design decoupled code. It takes a deliberate approach and the development of a set of habits to view a problem this way.

The second section is where the theory gets deep. These chapters present the DI design patterns and set of anti-patterns, concluding with a chapter on DI code smells. The final two sections show how to implement applications with DI, first directly and then using a kind of DI factory called a DI Container. These take existing classes and reflect them to create a new class that allows DI. The examples given are specific to .Net, though the illustration is useful.

I was a little concerned when I realized that the examples and code samples are written in C# and make use of .Net libraries. My worries were unfounded. The C# code will be clearly understood by anyone familiar with C++ or Java, and the .Net library references are reminiscent of Java APIs. The examples have the camelcase verbosity one would expect from those languages as well, but it doesn't interfere with clarity. Very few of the code fragments are longer than a single page, and the typeset annotations are well placed and clearly associated with the lines they describe.

Dependency Injection Principles, Practices, and Patterns provides a lot to chew on, and it's going to take me a while to properly consume and digest it. I have several web projects going where I hope to make use of it.

Building Secure and Reliable Systems

Heather Adkins, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea, and Adam Stubblefield
Google LLC and O'Reilly Media Inc., 2020, 557 pages
ISBN 978-1-492-08312-2

Reviewed by Mark Lamourine

When Google publishes a guide for infrastructure, you can be sure that it's worth reading. The real question is: is it something you can use? Google works on a scale that few other companies can. As a purely practical matter, few companies have the resources or the strictest requirements for efficiency that characterize the handful of truly colossal Internet companies. I picked up *Building Secure and Reliable Systems* with a bit of skepticism.

I was also concerned with the size of the book. At over 500 pages it's still not the largest infrastructure book I've read. I wanted to see how the authors managed the challenge of providing a useful level of information in a manageable amount of space.

This is the third book in a series Google has published on the topic of site reliability engineering (SRE) [3]. This is Google's refinement of the system administration model that has been called DevOps. The first defines and describes the philosophy of the SRE model and the role that the SRE plays in an operational organization. The second is a "workbook" for SREs, describing how they go about their job. This third volume provides a set of best practices both for the enterprise and for the service groups. It puts the SRE into the context of a complete organization in a way that can be appreciated both by the SREs and by their management and business peers.

Where the earlier volumes focused on workers and their tasks, this one illuminates the factors to consider in design and implementation of computer systems. The chapters alternate between discussion of a single desirable aspect of a system and case studies to give concrete examples.

Those design aspects are not things that are usually put high on the system requirements list: understandability, resilience, and recovery. The only element specifically for security is least privilege. The authors recognize that encryption and user authentication get a lot of attention. Defense in depth requires more care and thought. Security vulnerabilities will always be present, but exploits can often be neutralized by limiting what an attacker can access.

Each of these chapters really just provides additional incentive to follow ordinary design best practices. These discussions provide weight to arguments against cutting corners in design and implementation, and they provide rationale for better decisions than are often made.

The implementation section addresses considerations for reliability during the realization of the design, with chapters on writing, testing, and deploying the code and on surveying and debugging systems. In these chapters, the real nature of the writing comes through. This is a volume of collected wisdom: it's a series of thoughts and reminders—remembering to stop and think when things go wrong, for example, and to pair work where one is typing and the other is a scribe, both to avoid losing information and for mentoring.

The final couple of chapters talk about what I think has become the most critical aspect of software development and system administration: culture. There can be a lot of focus on technical stars in hiring and team formation. What experience has shown me is that people want to do good work and to learn and challenge themselves. The most common frustration is poor team empowerment and communication. All of the preceding chapters are nullified if the developers and admins aren't given the freedom and incentives to collectively evaluate and then act on their decisions.

Each of the chapters is nicely self-contained. The writing is clean, almost sanitary. This reflects the Google aesthetic of minimal bling, flash, and distraction. The authors provide frequent cross references, and each chapter concludes with a summary and a list of references. The spare nature of the writing style makes for a surprisingly readable text.

Except for a few details and discovery of a small set of obscure but useful tools, I didn't learn a lot that was new. For the developer or sysadmin, this book is a good complete compendium of the highest level considerations for system design. For project management, it is a window into the kinds of things the team should be discussing and resolving throughout a project. I didn't see anything here that made me think "you can only do that if you're Google." I'll keep *Building Secure and Reliable Systems* handy for when I need to champion more thoughtful, purposeful design and operational behaviors. "See, this is how Google does it."

Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats

Alex Matrosov, Eugene Rodionov, and Sergey Bratus
NoStarch Press, 2019, 448 Pages
ISBN-13: 978-1-59327-716-1

Reviewed by Rik Farrow

I chose this book to review after listening to an invited talk at WOOT '20 by the main author, Alex Matrosov, and because Sergey Bratus is also an author. I would ordinarily have steered clear of books primarily about Windows, but once you get past Part 1, about rootkits, you find yourself in territory relevant to Linux systems. The authors cover information relevant to anyone running software on Intel or AMD chipsets.

The book is well written and organized, and it includes example code (all Windows) and dumps from malware and firmware samples. I had little trouble reading the book, although I did want a glossary of abbreviations handy after a while, as there are loads of obscure TLAs.

The authors start out by describing TDL3 and Festi rootkits. These are "old," designed for 32-bit versions of Windows that have long been out-of-date but likely still run. Most of the techniques used—plugins to extend the malware, using a rolling XOR as "encryption," changing registry keys—seem familiar. What makes Festi interesting is the malware designers' knowledge of kernel internals. They hook both file and network drives very deep in the software stack, making them difficult to discover through Host Intrusion Prevention System (HIPS) products, as these tools also install hooks at the same layer.

As security in Windows improves, malware writers have shifted their focus to bootkits, methods of infecting the kernel during the boot process. Here again you will find information relevant to any operating system that relies on x86 chipsets. The authors cover the boot process and provide analyses of bootkit samples, as well as the arms race in bootkits that leads to UEFI Boot. UEFI is supposed to provide secure boot, with checks of the authenticity of code, but in many cases vendors have not properly implemented the standard.

Chapters 15 and 17 demonstrate the use of a tool, called Chipsec, that allows you to probe your firmware settings. The tool works for Windows, Linux, and macOS, and you can find the tool on GitHub at https://github.com/chipsec/chipsec. With the tool, you could see if your firmware is write-protected and whether SPI flash memory protections also have been enabled. The authors have tested a number of motherboards, and many of them have either not enabled or included firmware protections, making the system more susceptible to bootkit malware.

I really hadn't been paying much attention to Windows malware over the last decade, and the focus of this book is on two specific areas that cover some of the most sophisticated attacks possible. I enjoyed reading the book and learning about the malware, even if it was not particularly relevant to me, as "I don't do Windows." Still, there's more than enough here that's relevant to Linux users, as malware writers are now turning their attention to Linux servers.

References

[1] https://www.python.org/doc/sunset-python-2.

[2] E. Gemma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional, 1994).

[3] https://landing.google.com/sre/books/.

Statement of Ownership, Management, and Circulation, 09/30/2020

Title: USENIX Association/ *;login:*

Pub. No. 1044-6397

Frequency: Quarterly

Number of issues published annually: 4

Subscription price: $90.

Office of publication: 2560 9th St., Suite 215, Berkeley, CA 94710-2565
Contact Person: Sara Hernandez. Telephone: 510-528-8649 x100

Headquarters or General Business Office of Publisher: USENIX Association, 2560 9th St, Suite 215, Berkeley, CA 94710-2573

Publisher: USENIX Association, 2560 9th St, Suite 215, Berkeley, CA 94710-2573

Editor: Rik Farrow; Managing Editor: Michele Nelson, located at office of publication.

Owner: USENIX Association. Mailing address: As above.

Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, or other securities: None.

The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes have not changed during the preceding 12 months.

Publication Title USENIX ASSOCIATION/ ;login:			**Issue Date for Circulation Data Below** 09/01/2020 — Fall ’20 Issue	
Extent and Nature of Circulation			**Average No. Copies Each Issue During Preceding 12 Months**	**No. Copies of Single Issue (Fall 2020) Published Nearest to Filing Date**
a. Total Number of Copies *(Net press run)*			1863	1875
b. Paid Circulation *(By Mail and Outside the Mail)*	(1)	Mailed Outside-County Paid Subscriptions	805	818
	(2)	Mailed In-County Paid Subscriptions	0	0
	(3)	Paid Distribution Outside the Mails	513	529
	(4)	Paid Distribution by Other Classes of Mail	0	0
c. Total Paid Distribution			**1318**	**1347**
d. Free or Nominal Rate Distribution *(By Mail and Outside the Mail)*	(1)	Free or Nominal Rate Outside-County Copies	79	79
	(2)	Free or Nominal Rate In-County Copies	0	0
	(3)	Free or Nominal Rate Copies Mailed at Other Classes	20	25
	(4)	Free or Nominal Rate Distribution Outside the Mail	73	30
e. Total Free or Nominal Rate Distribution			**172**	**134**
f. Total Distribution			**1490**	**1481**
g. Copies Not Distributed			374	394
h. Total			**1863**	**1875**
i. Percent Paid			88.46%	90.95%
Electronic Copy Circulation				
a. Paid Electronic Copies			410	444
b. Total Paid Print Copies			**1729**	**1791**
c. Total Print Distribution			**1899**	**1925**
Percent Paid (Both Print and Electronic Copies)			91%	93%

Submit Your Work!

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

30th USENIX Security Symposium

AUGUST 11–13, 2021 | VANCOUVER, B.C., CANADA

The 30th USENIX Security Symposium will bring together researchers, practitioners, system administrators, system programmers, and others to share and explore the latest advances in the security and privacy of computer systems and networks.

Winter paper submission deadline: Thursday, February 4, 2021

PROGRAM CO-CHAIRS

Michael Bailey
University of Illinois at Urbana–Champaign

Rachel Greenstadt
New York University

www.usenix.org/sec21

Seventeenth Symposium on Usable Privacy and Security

Co-located with USENIX Security '21
August 8–10, 2021 | Vancouver, B.C., Canada

The Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021) will bring together an interdisciplinary group of researchers and practitioners in human computer interaction, security, and privacy. The program will feature technical papers, including replication papers and systematization of knowledge papers, workshops and tutorials, a poster session, and lightning talks.

Mandatory Paper Registration Deadline:
Thursday, February 18, 2021

Symposium Organizers

General Chair
Sonia Chiasson, *Carleton University*

Technical Papers Co-Chairs
Joe Calandrino, *Federal Trade Commission*
Manya Sleeper, *Google*

www.usenix.org/soups2021