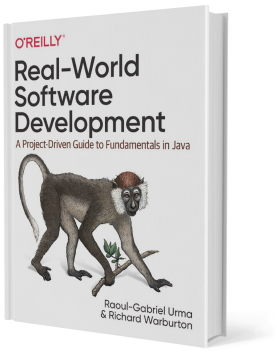

Real-World Software Development

By Raoul-Gabriel Urma and Richard Warburton
202 pages

Real-World Software Development

Reviewed by Andrew Binstock

January 10, 2020

When you learn a new programming language, there are always two distinct phases: The first is the initial acquisition of the language itself accompanied by a general sense of its core libraries and APIs. The second phase is the longer and, for many, the more difficult one—learning the conventions of programs written in the language. This step is occasionally referred to as "writing idiomatic code." But it entails more than acquiring the standard idioms. In fact, it means thinking *in* the language, becoming conversant with the major tools used by the language's community, and knowing where to go online for help and how to ask questions. While there are many books that teach all you'd want to know about the first phase of language acquisition, very few resources are available to developers to learn the idioms and culture of a language. Even 25 years on, I'm hard-pressed to name one book that does this for Java—that is, until the arrival of this new volume, *Real-World Software Development*.

This slim book teaches Java programming idioms and tools to junior developers who have some familiarity with the language. It does this through a series of small-scale projects. For example, the text starts with a "bank statement" project, which involves reading comma-separated values (CSV) records that relate to a mythical user's bank transactions and then performing some analysis on the data. This project presents several design patterns, acts as an introduction to some of Java's SOLID principles, explains coupling and cohesion, and provides a simple introduction to testing.

The following chapter develops the project further, brings in more design patterns and more advice (such as the use and overuse of interfaces and of exceptions), and introduces build tools.

Then there follows another project on managing documents, including automating the creation of metadata. It introduces subtler questions in design (the various trade-offs between options), more principles, and more tools.

A subsequent project creates a small-scale rules engine, which serves as the vehicle for demonstrating test-driven development, mocking, modeling state, and learning more-subtle principles than SOLID.

A final project, called Twooter, shows how to send messages to subscribers and gets into advanced topics such as distributed architecture, creating and using WebSockets, building a UI, persistence, and so on. A final chapter extends Twooter, presents functional programming in Java and additional tools, and points to new horizons for the now more-mature developer to explore.

Each project helpfully ends with suggestions for the reader to extend it with new features.

The writing is clear. The authors have written for *Java Magazine* many times (most recently here). They are expert Java developers and have presented at numerous conferences as well as taught in formal educational settings. The result is that they know their intended audience, so the pacing of the projects and the information is very smooth. Nonetheless, the book feels short at just over 170 pages, and I believe it

would have benefited from somewhat deeper coverage of the topics it presents.

I ran into a couple of frustrations working through the examples, which created a sense that the book was rushed. In the section on exception handling, for example, the authors rightly examine alternatives, such as `Optional<T>` and `Try<T>`. Wait, what? I looked at the latter item wondering how I didn't know about this. In the description of its use, however, the authors indicate it's not part of the language but it is part of some libraries. Which libraries, they don't say. (Vavr is the answer.) Frankly, it seems odd to include this feature at all. Similar small gaps appear here and there.

A second frustration is that many of the code listings contain code that wraps to the next line. Sometimes the breaks occur mid-word and sometimes at white space. But everywhere they appear, the word wrapping looks like carelessness. Formatting code to the column width to avoid word-wrapping is so trivial an exercise that there is no excuse for not doing it, especially since both the book and the PDF version allow for wide columns of text. The decision to let the wrapping stay is even more astonishing in a book whose stated goal is to teach idiomatic Java to the reader.

If you can put up with the latter and abide the occasional oversights, this is a very useful book to help junior Java developers to come up to speed.