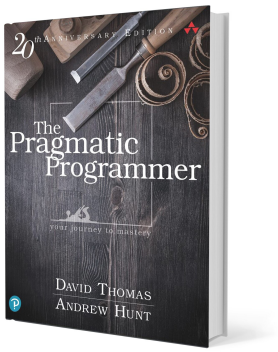

The Pragmatic Programmer, 20th Anniversary Edition

By David Thomas and Andrew Hunt
Hardcover, 352 pages

The Pragmatic Programmer, 20th Anniversary Edition

Reviewed by Andrew Binstock

November 21, 2019

The rate at which computing evolves means that even good books tend to enjoy short lifetimes and leave a limited footprint. A select few masterworks, however, have a lasting impact that transcends time. Among this small pantheon is the original version of *The Pragmatic Programmer*, which not only codified best practices in a rapidly evolving programming field, but also was directly responsible for the popularity of the Ruby language. (I'll explain the latter first: The book propelled its authors to the front rank of programming cognoscenti. So, when they later published an introduction on an unknown language called Ruby, developers paid attention and tried it out. From this small start, the language became admired for its expressiveness and was eventually adopted by David Heinemeier Hansson to write Ruby on Rails, which became the killer app for Ruby and largely changed the way commercial websites were created.)

What made that first edition so popular? It came out in an era when programming was evolving very quickly because high-quality, inexpensive developer tools were becoming available on the PC. At the time, everyone who programmed hacked out code, and there were few guidelines and best practices. For example, use of version control was the mark of disciplined developers and by no means universally adopted by enterprises. Unit testing had not yet emerged as a core practice, and so on. The book captured many emerging best practices and presented them as pragmatic guidelines. They were straightforward, clear, and convincing because the authors articulated problems accurately and solutions intelligently. If you could view your own habits with clarity, you could see where the pragmatism offered of Thomas and Hunt would help your work.

Fast-forward 20 years and you have a new edition, which for all intents is a completely new book rather than a mere update. As with the first volume, it is brimming with good, practical counsel. The advice to "Use a VCS" has been expanded to "Use VCS to drive builds, tests, and releases." This point of view is then amplified through discussion of what should be in the VCS, how deployed systems should be tags in the VCS, and so on, so that what starts out as a fairly unremarkable bit of advice evolves into a larger set of actions for you to consider in the light of your own organization's setup.

Another piece of offered advice is to really know your primary IDE or editor. That would seem like advice barely worth mentioning, but being pragmatic, Thomas and Hunt don't let you off easily by simply nodding in agreement. Instead, they define what *knowing* your IDE really means. I discussed their list of criteria in my editorial in the previous issue. Look it over: I expect you'll find you have homework to do.

Many times, the observations drive right into the heart of an issue. For example, in discussing the known difficulty of obtaining usable (or useful) requirements from users, the authors don't till the same ground we've all seen before (users don't know what they want, they know but they can't express, they express but they change their minds...); rather, Thomas and Hunt emphasize: *Users never read requirements documents*. So,

even if you clean up the previous known difficulties, you need to solve this one.

Not all the advice consists of revising known best practices. For example, in one of the 99 recommendations (each of which has a 3-4 page explanation), the authors urge that testing include property-based testing, claiming (accurately) that it will often reveal defects that you would overlook because it exercises functions with values you would never consider.

Some recommendations are entirely original (that is, I've not seen them in other books on developer best practices). For example, maintain a project glossary. It makes sense, but you gotta think of it to do it. The idea here is to make sure that developers are using terms from the problem domain in the same way users are. This avoids confusing misunderstandings as well as working on features incorrectly.

As I mentioned earlier, the book consists of 99 best practices, many original. And those that aren't strictly original are taken in unexpected directions that are intensely practical. The book is an easy read and because all the points stand by themselves (although there are many cross-references), you can pick it up to read the topics that most interest you, or better yet, the topics where your organization needs the most help.

This book is a treasure trove of great advice. I strongly recommend it. You'll thank me later. I guarantee it.