FROM THE EDITOR

Take Notes As You Code—Lots of 'em!

A small discipline that pays huge rewards

by Andrew Binstock

November 21, 2019

There are many things that developers are exhorted to do beyond just writing code, and there is little point in going over the basic ones yet again. For more-advanced practices, I highly recommend the book I review in this issue, *The Pragmatic Programmer* (20th Anniversary Edition), which contains a wealth of interesting and useful suggestions. I discussed one of them at length in my previous editorial: learning your primary development tool really well and how you can tell if you've reached the needed level of mastery.

In this editorial, I examine an underused best practice that *really* helps development: taking notes as you go. In an earlier editorial I discussed the importance of writing comments prior to writing code. This step lets you plan what you're going to do, provides documentation, and keeps your programming consistent with your design. As you program, you update the comments to make sure they remain in sync with the code. At the end, you have fully commented code that does what it set out to do. Terrific. This is a valuable step, but only a minor aspect of the note-taking I have in mind.

The note-taking I'm referring to begins as the equivalent of a programmer's to-do list, except rather than written beforehand, it's produced in the heat of battle. There are many entries that go into this. For example,

- The small tasks that need to be done for the current code to work properly. These tasks can be reminders (see whether `Message` can be made immutable), a list of refactorings you saw were needed but could not make at the time (extract a method to calculate postage), and so on. Some developers keep such a set of notes in an open window from which they obtain their next task, once they're done with the present one.

- Glossary entries: The book I mentioned earlier suggests having a glossary for each project so that users and developers are in sync on what terms mean, especially terms that are specific to the project domain. These notes are *not* the place for those entries because that glossary should be a separate document. However, there is a need for glossary entries for your codebase. This is a help for the age-old problem of naming. As you know, the larger the project, the more difficult it is to find meaningful names when you're layers down in the engine room. `HttpRing6` might look like a perfectly apt description. That's because for the last two days, you've been using the term `HttpRing` to describe a ringback-like mechanism that rides on HTTP. The `6` here refers to the severity of the action. While you might have captured some of that in the Javadoc comments, chances are good you might not have memorialized `HttpRing` conceptually. This should be in the notes you're taking and eventually placed in a codebase glossary, so that two years from now when you need to maintain this code, you can read through the code with some facility.

- A third kind of entry to make is more experimental in nature but very illuminating. This is a recommendation from Kent Beck, which attempts to give insight into what goes on during your work. Try to capture in your notes all the decisions you made in one day. The small things such as the number of retries you'll attempt or the length of time to wait for a future to complete. You will find, I expect, that you make many more decisions than you actually think you do. Many of these should later work their way into your comments or Javadocs. Putting them in your notes preserves them for just this purpose.

- The final set of entries are breadcrumbs, which are often added as the work day winds down. In the past, I used a feature in JetBrains IntelliJ (my longstanding IDE), which allows me to create a custom marker, similar to `//TODO` and `//FIXME`. Mine was `//CURR` and it showed the current location I was working at when I quit for the day. Over time, however, I found this marker was useful, but insufficient. We've all had the experience of looking at yesterday's code and wondering "What was I doing?" By adding breadcrumbs to my notes, I make it much easier to resume work later. Especially after a weekend.

For the most part, the note-taking should be fast, so I recommend using plain-text files. Markdown is a useful option if you want different sections for the categories I've listed above. An alternative is multiple notes files, one for each category.

As with comments, maintenance of notes is crucial: Tasks should be removed as completed, or they should be transferred to a project list or Kanban-style board. Likewise, breadcrumbs should be deleted when they're no longer needed. And the log of myriad decisions can be erased once they're no longer useful. Only the coding glossary should be kept for future purposes. Everyone will have their preferred way to structure the glossary: by package, by subsystem, by project.

Like so many practices, wide-ranging note-taking works best if it suits your working style. But if you don't use some version of it, chances are you're carrying way too much live baggage in your head, you frequently forget things you wish you'd remembered, and you make it very difficult to maintain your code.

Also in This Issue

Understanding the JDK's New Superfast Garbage Collectors
Epsilon: The JDK's Do-Nothing Garbage Collector
Understanding Garbage Collectors
Testing HTML and JSF-Based UIs with Arquillian
For the Fun of It: Writing Your Own Text Editor, Part 2
Quiz Yourself: Identify the Scope of Variables (Intermediate)
Quiz Yourself: Inner, Nested, and Anonymous Classes (Advanced)
Quiz Yourself: String Manipulation (Intermediate)
Quiz Yourself: Variable Declaration (Intermediate)
Book Review: The Pragmatic Programmer, 20th Anniversary Edition

Andrew Binstock

Andrew Binstock (javamag_us@oracle.com, @platypusguy) is the editor in chief of *Java Magazine*. Previously, he was the editor of *Dr. Dobb's Journal*. He co-founded the company behind the open-source iText PDF library, which was acquired in 2015. His book on algorithm implementation in C went through 16 printings before joining the long tail. Previously, he was the editor in chief of *UNIX Review* and, earlier, the

founding editor of the *C Gazette*. He lives in Silicon Valley with his wife. When not coding or editing, he studies piano.

Share this Page