How to Write a Programming Book

Adrian Kosmaczewski

2021-09-10

Writing a programming book is not very complicated, to be honest: it just consists of putting one word after another. Shocker! I'm not kidding.

I assume that if you're reading this, you are interested in writing a book. In that case, the best you can do to write your book is to have a deadline, and then keep a simple routine to achieve it: *write at least 500 words a day.* That's it.

Of course, it's easier said than done, at least at the beginning. When I began writing, I could barely reach the 500 words mark after hours of trying. Now it only takes me twenty minutes to get that far, and sometimes I can get at least a thousand relatively coherent words per hour, sometimes even more. As Joel Spolsky said, writing is a muscle, and the more you do it, the easier it is.

But how much is a thousand words? To have an idea, a book of 250 pages (without code or illustrations, just text) is approximately a volume of 50'000 words. Books with code snippets have less words, maybe 30'000, but no less information; actually more, because of the higher density of information in said code snippets. My second book was around 30'000 words, precisely, around 280 pages. Which means that, at a rate of 500 words a day, I needed 60 days of work (around 12 weeks, at 5 days each) to finish… the draft.

Because this is very important: do not edit during the draft writing phase. Never. Just… write. Pour text on the screen. Do not do anything else than writing. If you start editing yourself… well, you'll never finish the book. Don't censor yourself during that writing phase. Get the draft out.

But wait, what about inspiration? It doesn't matter, really. Write even if you're not inspired, even if it hurts, even if you hate the words that come out. Write. Get your 500 words even if you know that it's utter garbage. *Get. The. Draft. Out.*

Maybe it's not utter garbage, but you're in a bad day. Maybe you'll find something great in those ugly words in the future. For the moment, just write. I know, it sucks. But it works. Believe me.

After the draft comes edition (that is, the self-censorship part). I personally

prefer, for this phase, to print the text on good old paper, go to the nearest café, and read it, away from everything, just me, the draft, some coffee, and a highlighter marker. A yellow one. Never green or red. I'm kidding, any color will do. OK, yellow *is* better.

Do not edit your draft immediately, though; wait a whole week before starting the editing phase. It is important to "forget" what you wrote a little bit before editing it.

But what would be a programming book without code snippets? Well, the rule of thumb is the following: *first write the code, then write the text that explains the code.* That's it. Many of the most important books in programming have been written this way. Dave Mark told me this, and he knows a thing or two about writing programming books.

I know, I should have stated this last rule first. Sorry about that. I hope you haven't written many pages of your draft before reaching this paragraph. First the code, then the text.

How does edition work? You will read, edit, read, edit, and re-read, and re-edit your draft a gazillion times. Not really that much, but a lot, or until it reads well.

But what does "reads well" mean? You'll know when you read yourself and you like what you wrote. There is no other gauge. If you like what you read, then it's done.

And when will you be done with your book? Never, probably, but that's why publishing exists: as Jorge Luis Borges once said, publishing is a great way to stop editing. Generate a PDF, sell it on Gumroad, Leanpub or Lulu, and you're done.

(Don't sell your book in Amazon, please.)

With these simple rules you should get your book project done in a few months.

Another thing: don't use Word, LibreOffice, Pages, AbiWord or Calligra Words or anything like that. Those editors mess your mind with toolbars filled with options and buttons and menus and things that will distract you from the important bit: the text itself.

Because what's important is to *Get. The. Draft. Out.* Capisce?

Instead, use the closest plain text editor you can get your hands on: Notepad, Notepad++, Vim, Emacs, Visual Studio Code, TextEdit, EditPlus, WordGrinder, Gedit, ~~TextWrangler~~ BBEdit, Sublime Text, TextMate, Ulysses, Apostrophe, UltraEdit, Joplin, etc. I've written extensively about text editors (and keyboards) in one of my books, so I'll just let you download it and read it if you want to know more. It's a funny book, you should read it.

Remember to create separate files for each chapter, and of course, use version control; Git or something else that you feel confortable with. Yes, Subversion is

fine, if that's your thing. Version control is like undo on steroids. You can also open an account on GitHub or GitLab and get a backup copy of your whole project on a private repository there. Backups are important. Backup your book.

(If version control is too geeky for you, well, maybe writing a programming book would be already a challenge in itself; but if that's the case, Dropbox has some kind of version management thing integrated, but somewhat restricted for free accounts. I prefer Git + GitLab in any case.)

With a plain text editor you can of course save your work as a bunch of `.txt` files, but I personally use Asciidoc. It is plain text with some markers for titles, bold text, and more. There is also the venerable and über popular Markdown format, which is similar, but I prefer Asciidoc instead of Markdown for books.

I think Markdown is great for blog posts (like this one), and shorter texts; but nothing beats Asciidoc for longer works, like books, precisely. Of course, if you like LaTeX, that's another great option.

If you want a longer version of what you just read, from a real rock star in the world of programming book writing, and with lots of great information about publishing houses and royalties and more, check out Scott Meyers' (of "Effective C++" fame) own guide to writing technical books.

In short, writing consists of a set of very simple rules applied iteratively. It sounds awfully similar to functional programming. And both are equally wonderful.