# Eleven Quick Tips for Writing a Technical Book

Jess Haberman[1], Greg Wilson[2*]

**1** Anaconda Inc., Austin, Texas, United States
**2** Third Bit, Toronto, Ontario, Canada
* gvwilson@third-bit.com.

## Introduction

The Internet has put a world of information just a click away, but that information is often fragmentary, contradictory, incomplete, or out of date. People still need coherent narratives that provide a coherent overview and lay out a learning path that will get them there. In other words, they still need book-length expositions, which means someone has to write them and publish them.

One of us (Haberman) has been helping writers and experts produce instructional content for most of her career: prior to becoming Director of Learning Solutions at Anaconda she spent 14 years in book publishing, with several years at O'Reilly Media signing and developing technical books. The other (Wilson) has written four books on programming, co-authored three others, and edited six. The tips below summarize what we've learned along the way.

## Tip 1: Don't write for money.

The first thing any prospective author needs to understand is the economics of publishing. Typically, one-third of the books on a publisher's list lose money, one-third break even, and one-third are profitable to a greater or lesser extent. For most, it's "lesser": publishers actually make most of their revenue from a handful of bestsellers.

Your publisher will typically give you royalties ranging from 10% to 15% of net revenue (i.e., revenue after discounts). You can expect to sell 200–1000 copies; a handful of books do much better than that each year, but only a handful. Downloading has obviously impact this: PDFs and BitTorrent files often appear online within hours of the first digital sales of books (particularly textbooks). Assuming a US$50 price tag, this means that you will probably earn between US$1000 and US$7500. Since it's going to take at least a thousand hours to write the book, you'll almost certainly be better off financially doing some consulting (or even getting a part-time job in a fast food restaurant).

So why write a book?

**To build your reputation.** Many musicians now view albums as a way to get people to buy concert tickets and t-shirts. Similarly, many authors and publishers view books as a way to get people to enrol in tie-in workshops and speaking engagements.

**To give back to the community.** None of us got here without help, and we have a moral obligation to repay that by helping others.

**Because you enjoy doing it.** Some people like working with wood or wool; you might like working with words.

**To gain a deeper understanding of the topic.** Writing requires research, and you'll gain expertise during the writing process.

## Tip 2: Don't start with a book.

Writing a book is a big undertaking, and it's easy to get discouraged. Instead of tackling it head-on, start blogging regularly (e.g., one article a week, each a few hundred to a thousand words long). Don't try to write them in the order you think they'll eventually go into the book: one reason to blog is to get a better idea of what should come before what. As you become more comfortable, though, try writing multi-post series that can eventually become chapters.

## Tip 3. Test your material.

If you run even a single workshop based on the material for your book while you are writing it, you will almost certainly realize that several of your core assumptions about your audience and material are completely wrong *while you still have time and energy to fix things*. Conferences are always looking for tutorials, your local college or university might be looking for a sessional course, and dozens of learning sites exist to publish training materials (though in this case you must be careful that their licensing allows you to re-use the material). Your colleagues will probably welcome some lunch & learn sessions, and if all else fails, you can create a YouTube channel to find out how your material sounds.

A corollary is that there's no point trying material out if you don't **incorporate what you learn**. If you pay attention to feedback, you will probably rewrite every line of your book at least twice and revise some parts a dozen times or more. You can skip this work if you don't collect or act on feedback, but your book will almost certainly be a lot worse for it.

### Union and Intersection

No matter how carefully you plan, you will hit part 5 and realize that there was something you should have explained back in part 3. You can fill this in on the fly if you are teaching in person, but that doesn't work in a book. For this reason, a live lesson should be the *intersection* of what audience members don't know (i.e., the things you're sure they all need) while a book has to be the *union* of what readers don't know (i.e., the things any of them need).

## Tip 4: Read, then write.

Go through two or three of your favorite technical books and make notes about what their authors have done well. Doing this will give you some ideas for your own book, but it will also help you become more conscious of your writing—in musical terms, it will help you learn how to listen to yourself while you're playing. One of us (Wilson) learned a lot from the classic Unix books by Kernighan et al [1–4] and from [5], but your tastes may vary.

## Tip 5: Start with a learner persona.

A *learner persona* is a fictional character that captures key aspects of who you are trying to teach, what they already know, and what they want to learn [6]. Create one or

two that are fairly similar (i.e., a primary and secondary audience) and write for them, because a book that's meant for everyone is actually useful to no-one.

As you're doing this, keep in mind the difference between a *novice* (who is trying to build a mental model), a *competent practitioner* (who has one and wants to fill in gaps in their knowledge), and an *expert* (who wants higher-level discussion of tradeoffs and alternatives). Novices need tutorials that help them build a coherent mental model of the domain; competent practitioners want to fill in the gaps in their knowledge, and experts are usually interested in higher-level discussions of patterns, alternatives, and tradeoffs. No single book can serve all three well.

As a corollary, **don't write a general introduction to a topic** unless you're going to be one of the first two or three to market. Instead, focus on a particular aspect (e.g., "Python for Web Scraping") or a particular audience (e.g., "R for Librarians"). Your potential audience may be smaller, but you'll reach a much higher percentage of that audience, and a focused book is much easier to write.

And please, **don't write a reference book**: it can never be as comprehensive or stay as up-to-date as the Internet.

## Tip 6: Know your competition.

Conduct research to find out what other resources exist for this topic, keeping in mind they don't have to be books. Where is the audience that you have already identified learning more about the topic right now? What can you offer that differs from and improves on that? The most common weakness in other resources is they are out of date or no longer accurate, but other times they are poorly organized, not well edited, lack some unique insight you can bring to the topic, or are trying to be too general. Every publisher's proposal form will ask about this; finding an answer *before* you have written several chapters can save you a lot of fruitless labor.

## Tip 7: Avoid common mistakes.

**Avoid banal advice** (like "avoid common mistakes"). Few things are as frustrating to the reader as sentences like, "You should carefully consider users' needs," because no sensible person would recommend the opposite. If you can't provide a checklist of specific things to consider, or a scenario to give the reader insight into how you think through that class of problem, you're giving them the intellectual equivalent of junk calories.

**Don't be too granular.** Don't spend months or years writing a book in such specific detail that a new release of software makes your advice obsolete or simply wrong. Instead, generalize and focus on principles that are likely to be long-lived without violating the rule about banality. Your goal should be for your book to remain relevant for three to five years.

**Don't write to cover your ass.** We have read dozens of books that start with a short introduction to XYZ. We cannot remember a single one that was actually useful. If your audience knows Python, they don't need a chapter-length intro to the language, and if they don't, one isn't going to help them. Authors usually include those short introductions so that they can claim the book is suitable for novices when they knew in their hearts it isn't. Remember, it's perfectly acceptable to target an audience with more sophisticated knowledge than "none" or "basic".

**Don't try to be funny.** Very few jokes are funny the second time you hear them; even fewer can stand a third re-telling unless they're very, very dry. On a related note, please don't use exclamation marks: what you're writing probably isn't that surprising, and certainly won't be the second time around. Most bold, italics, and underline formatting is superfluous[1].

## Tip 8: Write a proposal.

If your goal is to have your book professionally published, write a proposal before you dig into writing the book. You should spend a few weeks on this and make it as complete and convincing as possible. When you pitch it to your preferred publishers, do some research to find the likely editors so you can be sure they see it instead of it winding up in the slush pile of unsolicited proposals and manuscripts.

If you have a strong proposal, it should at the very least land you some time with editors who are knowledgable about the market for the topic. The best editors will help you hone the proposal by asking good questions and steering your book in the right direction.

If you receive an offer for publication, negotiate. Let editors know if you have received or anticipate an offer elsewhere, or if you're considering self-publishing. There's no harm in seeking the best offer you can (i.e., the one most aligned with your goals, whatever they are): even if an offer doesn't improve, you've wasted nothing and can move forward knowing that others are truly invested in your book and will help keep you accountable.

### Shop Around

Know what your publishing goals are. Do you prefer the style, brand, or reach of a specific publisher? Are you seeking the top financial offer? Identify your goals before you begin pitching. Don't feel you have to send your proposal to one publisher at a time, but if you are shopping it around, it is common courtesy to let editors know if the proposal has garnered interest elsewhere. Communicating your goals and timelines can only help you in navigating the pitch process smoothly and achieve your publishing goals.

## Tip 9: Drywall, then paint.

In other words write a rough draft with placeholders and *then* worry about diagrams, bibliography citations, glossary entries, and so on. For example, the word 'FIXME' appeared over a hundred times in first draft of [6]; half of those turned out to be irrelevant by the time the text settled down.

An important special case of this is that you should do diagrams on a whiteboard first. It's faster and more flexible than any computer drawing tool (even fingertip sketching apps for tablets) and a photo taken with a cell phone is good enough for your first readers. Many publishers will re-create your sketches anyway, so don't fuss over them too much.

## Tip 10: Automate, but proofread.

This is the technical author's equivalent of "trust, but verify". Depending on the tools you use, you may be able to regenerate all of the examples in your book with a single

[1] And remember: nobody reads footnotes.

command, but you must still re-read the discussion about those examples every time you make a change to make sure they haven't fallen out of step. Similarly, it doesn't matter how well you know your chosen subject: something will have changed since the last time you looked, and something else will change in the year or more it takes you to complete your manuscript. Your publisher will keep track of errata (i.e., printing errors); it improves longevity of your book if you address them at regular intervals.

Keep in mind that most people won't read your book linearly: they will skip some sections or jump ahead to find an answer to a problem they need to solve right away. You can help them by making liberal use of cross-references (e.g., "as covered in Chapter 2") to help them navigate [7]. And because page and chapter numbers frequently change, use chapter titles instead.

Speaking of chapter titles, consider titles that begin with verbs. If you're writing an instructional book, this will help ensure you're focused on the task or skill you're trying to teach. (See [8] for an example.)

#### There Are No Good Choices

We have worked on books using LaTeX, Microsoft Word, computational notebook, and Markdown-based static site generators, and they are all frustrating. Tools that store everything as lines of text with manually-typed formatting commands and a compilation step for previewing impose a high cognitive load, particularly when it comes to creating tables or diagrams, while WYSIWYG tools don't play nicely with version control. There is no good technical reason for us to have to choose, but until programmers stop insisting on backward compatibility with punch cards, authors have to accept frustration as a fact of life.

## Tip 11: Trust your reviewers and your editor.

Lots of people will do cursory reviews or give your manuscript a single careful read; people who will look at changes over and over again and see what's actually on the page each time are worth their weight in rubies. Trust them: if a passage doesn't make sense to them, then it doesn't matter if it makes sense to you.

Similarly, as Tom Wilkie once said, an author's job is to produce the manure in which an editor grows something worth reading. You may sweat over an example for days, but your audience doesn't see your effort: they see your results. If explanations that seem clear, concise, and elegant to you don't make sense to your readers, then they're right and you're wrong.

A professional editor doesn't get paid extra for making more edits. So if they mark something, they probably have a good reason. Ask why if you're not sure. Keep in mind that your motivation and your editor's are the same, to make the book the best it can possibly be.

## Tip 12: Don't be afraid to set it aside.

One of us (Wilson) wrote three quarters of an introduction to R for Python programmers that will probably never see the light of day. This is not unusual: many authors find that the more they write, the less they believe the book will actually help its intended audience. There is nothing to be ashamed of in this: scientists don't expect every hypothesis to turn out to be true, and authors shouldn't either.

Of course, if you have signed a contract and are suddenly rethinking whether your book should exist, have an honest conversation with your editor about next steps. Don't hide information, and don't become impossible to reach. Editors are used to authors

becoming discouraged and can provide a much-needed reality check. They can also help you evolve the reasoning, scope, timing, or approach to your book: it's actually part of their job to do so.

## Tip 13: Stop, ship, and celebrate.

Your book will never be perfect. It will probably never even feel finished, any more than software does, but you should ship it anyway and then celebrate what you've accomplished. Writing a book can be a hard slog, but it's not easy to replicate the feeling of accomplishment when you find your book listed online or receive a shipment of printed copies. Don't be afraid to tell your friends or post on your social media accounts: it isn't bragging if you actually did it, and if it was worth your time to write the book, it's worth your time to help people find it.

## References

1. Brian W. Kernighan and P. J. Plauger. *The Elements of Programming Style*. McGraw-Hill, 1979.
2. Brian W. Kernighan and P. J. Plauger. *Software Tools in Pascal*. Addison-Wesley Professional, 1981.
3. Brian W. Kernighan and Rob Pike. *The Unix Programming Environment*. Prentice-Hall, 1983.
4. Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, 1988.
5. Jon Udell. *Practical Internet Groupware*. O'Reilly Media, 1999.
6. Greg Wilson. *Teaching Tech Together*. Chapman & Hall/CRC Press, 2019. How to create and deliver lessons that work and build a teaching community around them.
7. Sarah Lin, Ibraheem Ali, and Greg Wilson. Ten quick tips for making things findable. *PLOS Computational Biology*, 16(12):e1008469, 12 2020.
8. Dan Meador. *Building Data Science Solutions with Anaconda*. Packt, 2022.