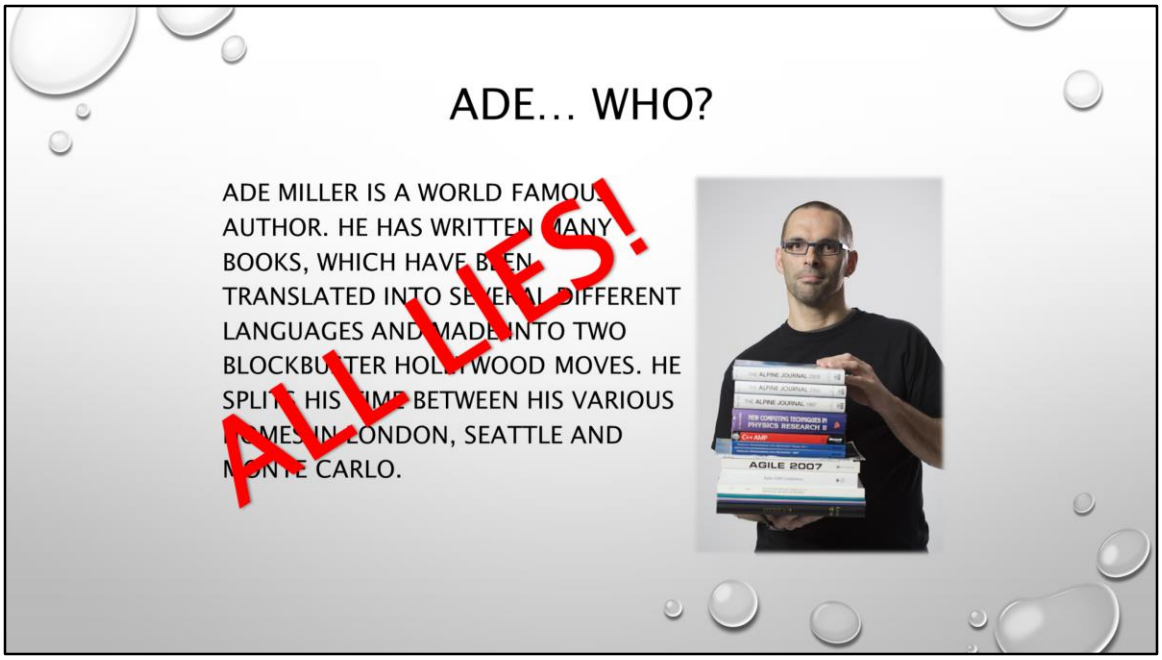
WRITING FOR DEVELOPERS

BOOKS, BLOGS, DOCUMENTATION AND SAMPLE CODE



Written a lot:

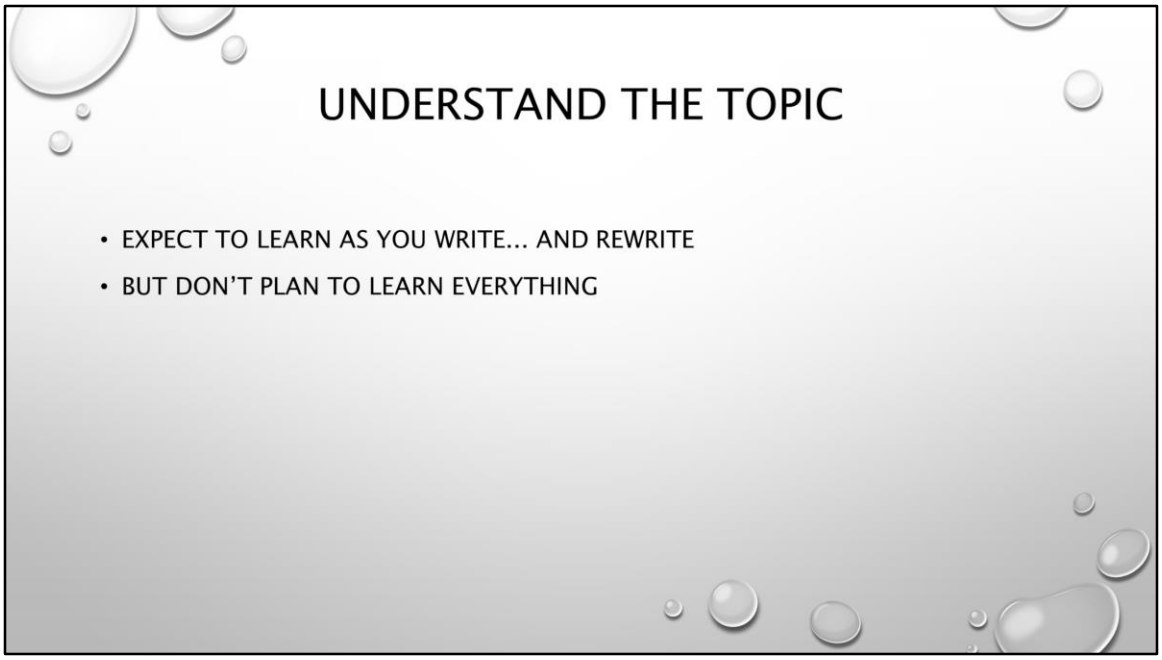
Co-authored 3 programming books and a thesis
Many technical and research papers as well magazine articles.
Thousands of blog posts

THE FIRST AND ONLY RULE

Blogs -> Short Docs -> White papers -> Books

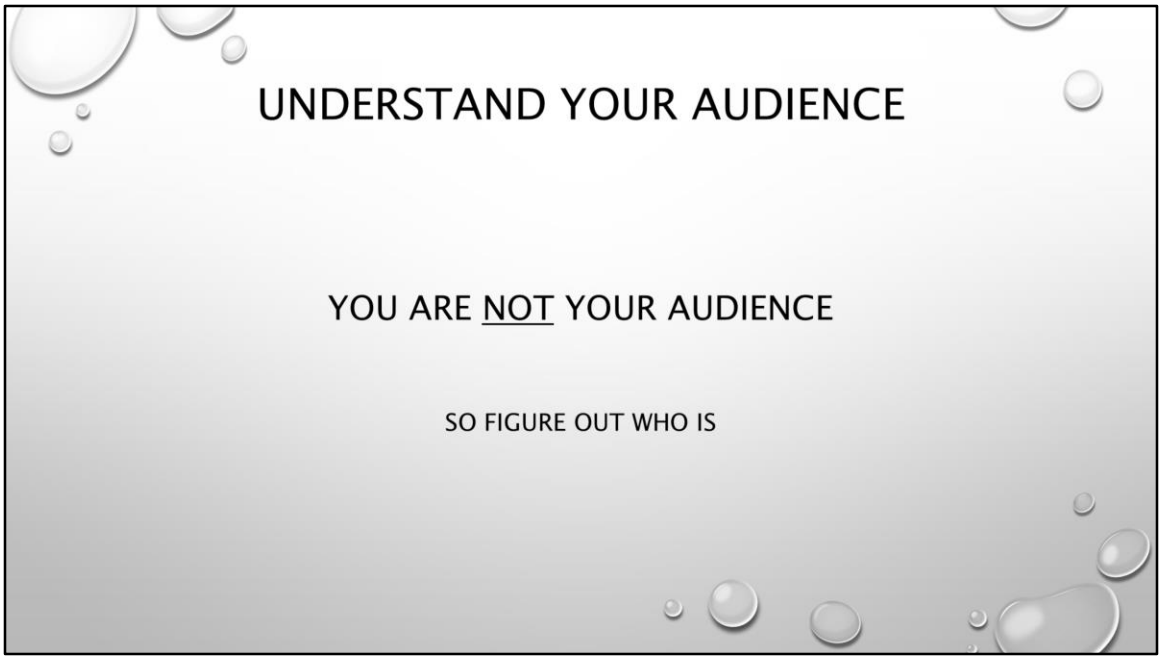

So little to say...
and so much time

HOW?



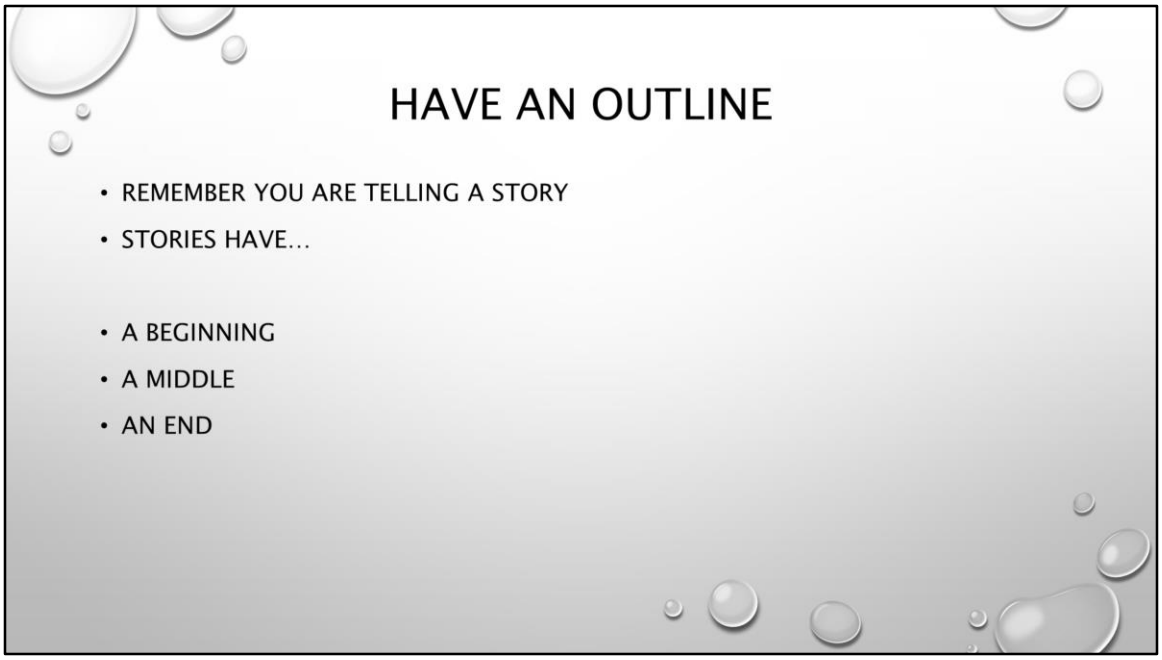
If you're contemplating writing a technical book on something, then you probably already consider yourself to be something of an expert. Writing a book is a great opportunity to round out all the dark corners of the topic and understand it completely. There's a lot to be said for learning something by forcing yourself to explain it to someone else. This is not an invitation to decide to write a book something as you're learning it. The result will read like a Presidential candidates manifesto but with fewer outright lies.

Even if you know the subject cold that doesn't mean you understand what explaining it to others is going to entail. When you assemble the outline (see #3) do it with an eye to identifying topics that will be harder to explain or less familiar to your audience (see #2).



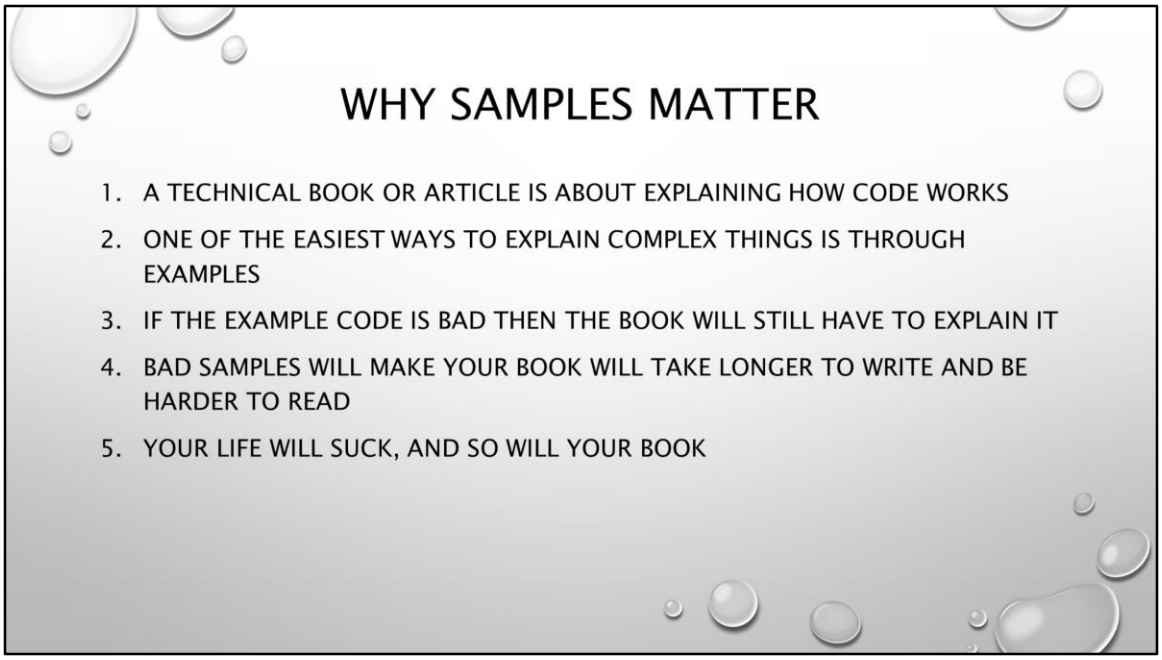
Who is the book for? What parts of the technology are they familiar with and what will be completely new to them? This will help identify additional material you may have to add when it comes to writing the outline. Creating a persona to describe your reader may be helpful.

For example when we wrote the C++ AMP book we decided that our readers would have a reasonable amount of C++ and STL experience but might not be familiar with the new features in C++11 so we included some additional material on lambdas and references to additional reading material for readers who needed it.



A book outline lays out each chapter of the book, the topics covered in each chapter, any accompanying material like sample code, and the approximate page counts. This will help you estimate the size of the Herculean task ahead of you and at some point will boost your spirits when you realize that you might be halfway done and the end to the use of senseless split infinitives may be in sight.

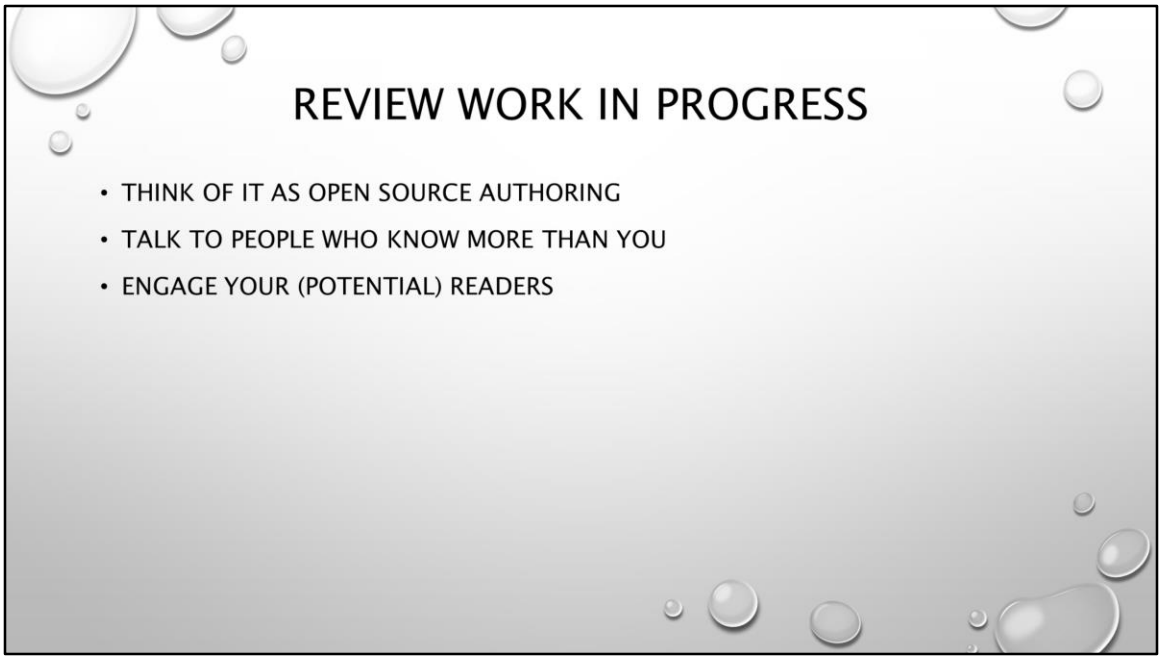
Of course only an idiot would write a book without a complete outline. Believe it or not we didn't do a very though job on the C++ AMP book outline. The original was missing several chapters we had to write for the final book. This was, erm, well shall we say, painful.



In many ways a technical book is really an explanation of the samples. In particular programming books have samples or case studies associated with them. If the code is hard to understand the book will still have to explain it. The better to code the less time you will spend doing additional writing.

Expect to write the samples before the chapter they are described in and then to rewrite the samples as you write the text to make them seem like they go together. For example you may find that and explanation is better served using a specific terminology like master/servant. Don't hesitate to change the code to use the same terminology in method and variable names. There'll be another blog post on writing good sample code for books, which turns out to be a bit different from writing good production code. That way we can all bicker about bracket placement later.

Make the samples open source and distribute them for review with the draft chapters and ask for reviewer feedback. When I wrote the samples for parallel programming with Visual C++ I hadn't written any C++ in a long time. I was lucky enough to have members of the PPL, STL and MFC teams take a look at my code before we shipped. It was better as a result and I learned a thing or two along the way.



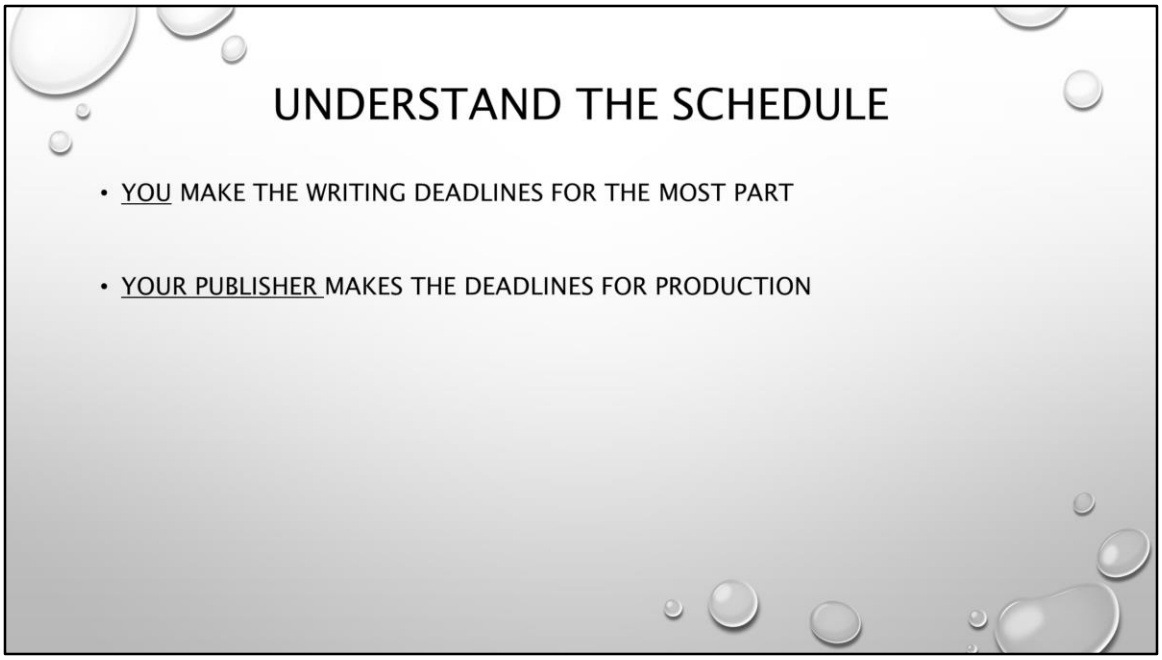
Even fiction authors have people who read early drafts of their work. For technical writing this is absolute must. Ideally you should have reviewers who represent your audience and can give you feedback on the overall readability of the book. You should also have subject matter experts who can ensure that the content is completely correct. You can also put draft chapters up on the web and allow anyone to send you feedback. Some publishers offer draft books for review and ask people to pay for them. Personally I'm not a fan of this as it limits the potential readership and thus feedback. Not to mention paying for a "beta" doesn't cut it in the rest of the software world.

Sign up more reviewers than you think you need. Reading a book is a big time commitment and people often don't manage to provide feedback on all the content. Remember all they get is an acknowledgement (see #9).

f you've gotten a lot of review feedback (see #5) then you have to decide what to do with it. Sometimes different reviewers will provide conflicting feedback. Reviewers have many different motivations and perspectives. Some reviewers may be more or less like your intended audience. Some may be keen that you show everything in the best possible light, others may be far more critical.

All feedback should be addressed but this means you should take it into consideration and reword or rewrite accordingly. Don't feel like you have to take a reviewer's suggestions verbatim, especially if you don't think it will help your audience. I can think of several cases where I outright did not include feedback because I didn't think it served the reader and others where I completely rewrote sections of the book based on feedback from the same person.

A great way to see this process in action is at a patterns writing workshop. It's amazing to have readers explain their interpretation of your writing to you in person. Ralph Johnston's students at the University of Illinois provided a lot of feedback like this and it proved really helpful for our C# book.

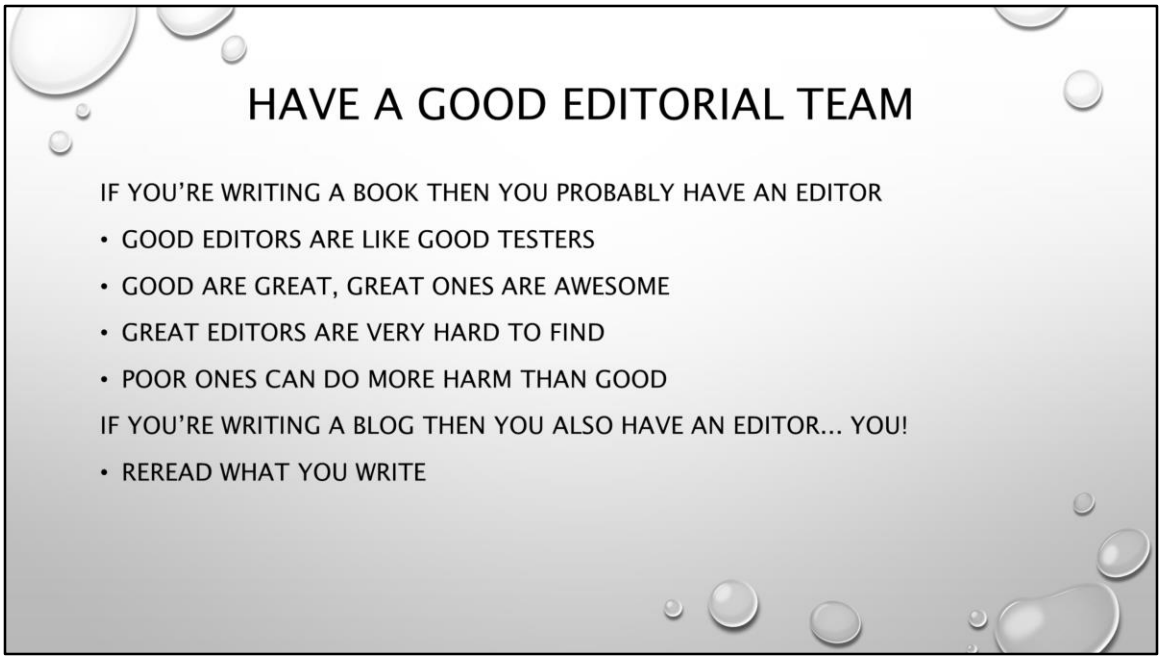


Books are somewhat odd things. The writing doesn't have many hard deadlines, except for the ones you set yourself, and the effect of being late with one chapter has a linear ripple effect through the rest of the schedule. The same is not true of production.

Many publishers use contractors for editing, indexing and layout or have staff booked to work on your book for a set time period. If you blow a deadline for production by a day it might cost you a couple of weeks as the people you are relying on for the next production step are now unavailable. Most of these steps are also very linear. If layout is late then indexing cannot start (it requires final page numbers) and you may loose that person's time too. A slip on your part of a day or two might delay the book by several weeks or a month.

In short you want to really understand the production schedule and make sure you hit it. Ideally the production team should be continually reminding you about this and making sure you're clear on the deadlines and what it means to miss them. This really helps when all you can think about is how to explain some really tricky concept in Appendix A. Good production people will keep you on track, bad ones will mention in passing on a Wednesday that they need a piece of writing or review on Friday or the

schedule is out the window.



Good editors are like good testers. We all think we can ship without them, the first time. if you try a second time you were either lucky of not that smart. Good editorial and copy editing can really make your book. An editor can really improve the overall structure of your book and flush out inconsistencies. A good copy editor will make your book far more pleasant to read. Conversely, a poor editing team may actually do more harm than good, even introducing errors in technical content.

Make sure you're clear with you editorial team as to how you think things should be laid out, especially when it comes to technical terms. I spent many hours italicizing array when it referred to a C++ AMP *array<T>* type and un-italicizing it when it simply referred to a sequence of values. This is not a fun way to spend a sunny Sunday afternoon.

ACKNOWLEDGE YOUR TEAM

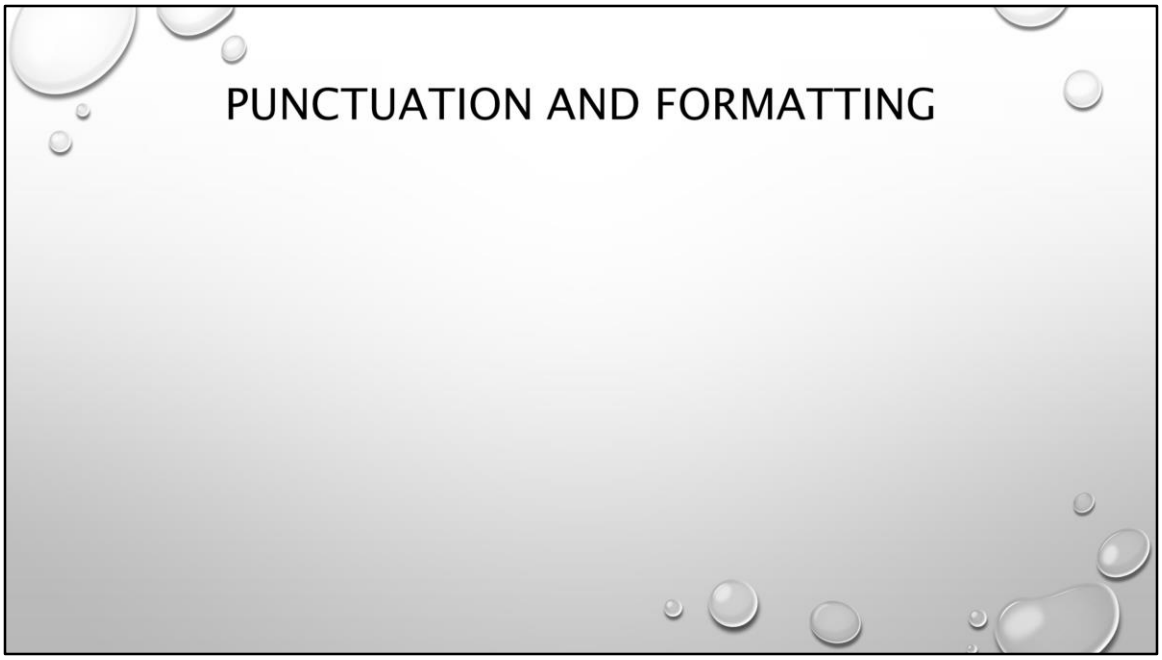
JUDITH BISHOP
COLIN CAMPBELL
ROANN CORBISIER
RALPH JOHNSON
ROBERTA LEIBOVITZ
KEVIN MCLANE
STEVE TOUB
…

Writing a book is a lot of work. I lost count of the number of nights and weekends I worked on either the text or the sample code. I spent many a Saturday afternoon with nobody but my co-author to talk to on IM. Remember, writing a technical book isn't going to make you rich. Don't blow what little money you do make on a divorce attorney.

SOME NOTES ON STYLE

Can your technical book be read on a single flight across the US?
Can your blog post be read while drinking a single cup of coffee.

Often fewer words is better.



Get your editor to help
Get WORD to help
Set consistent style guidelines up front.

SAMPLE CODE

WRITE THE SAMPLES FIRST

REMEMBER

BAD SAMPLE => BAD BOOK, AND MORE WORK

- EXPECT TO SPEND A LOT OF TIME ON THE SAMPLE CODE
- EXPECT TO REWRITE IT AS YOU REVISE THE TEXT
- BEWARE OTHER PEOPLE'S SAMPLES

THE BASICS

ALL CODE SHOULD BE…

- BE CORRECT
- COMPILE ALL BUILD CONFIGURATIONS; X86, X64…
- BE READABLE
  - USE MEANINGFUL NAMES
  - MAKE SENSIBLE USE OF WHITESPACE AND LAYOUT
  - AVOID OBSCURE LANGUAGE FEATURES
- USE A SELF-CONSISTENT STYLE

THINGS TO AVOID

- LOTS OF IRRELEVANT CODE IN THE SAMPLE
  - ERROR HANDLING CODE
- OVERLY TERSE CODE THAT IS LESS READABLE
  - AUTOMATIC VARIABLE DECLARATION; VAR, AUTO…
  - REDEFINITIONS; #DEFINE, TYPEDEF, USING…
- YOUR CODE SHOULD MATCH THE BOOK
  - FIT THE AVAILABLE COLUMNS, OFTEN < 80
  - NAMES AND METAPHORS SHOULD BE CONSISTENT
- TEST DRIVEN DEVELOPED CODE

CASE STUDIES

- MUCH CLOSER TO PRODUCTION CODE
  - HANDLE ERRORS
  - OPTIMIZED(?)
  - HAVE NICER USER INTERFACES
- TESTED, WITHOUT THE LUXURY OF A (REAL) TEST TEAM
- CAN BE USED AS DEMOS

EXAMPLES DONE RIGHT

The following code uses **accelerator::get_all()** to enumerate all the available accelerators and print their device paths and descriptions to the console:

```
std::vector<accelerator> accls = accelerator::get_all();
std::wcout << "Found " << accls.size()
    << " C++ AMP accelerator(s):" << std::endl;

std::for_each(accls.cbegin(), accls.cend(), [](const accelerator& a)
{
  std::wcout << " " << a.device_path << std::endl
     << " " << a.description << std::endl << std::endl;
});
```

The description property provides a user-friendly name for the accelerator, but the **device_path** property provides a persistent unique identifier that is more useful for programmatically selecting accelerators.


I'M OUT OF BED AND DRESSED - WHAT MORE DO YOU WANT?