Recommended Reading for Developers

blog.codinghorror.com/recommended-reading-for-developers

Jeff Atwood 2004年2月2日

02 Feb 2004

> This list was last updated in **March 2015**.
>
> Why are updates to my reading list so rare? Because computers change a lot in 10 years, but people don't.
>
> To make better software, you need to understand how *people* work, and that is what the books I recommend tend to focus on.

Code Complete 2

Steve McConnell's *Code Complete 2* is **the Joy of Cooking for software developers**. Reading it means that you enjoy your work, you're serious about what you do, and you want to keep improving. In Code Complete, Steve notes that the average programmer reads less than one technical book per year. The very act of reading this book already sets you apart from probably ninety percent of your fellow developers. In a good way.

I like this book so much that the title of this very website is derived from it – the examples of what not to do are tagged with the "Coding Horror" icon. There's nothing funnier than a Coding Horror – until you have to deal with one yourself. Then it's suddenly not so funny any more. Do yourself a favor. Make this the first book you read, and the first book you recommend to your fellow developers.

The Mythical Man-Month

Arguably the only classic book in our field. If you haven't read it, shame on you.

I challenge any developer to pick up a copy of *The Mythical Man Month* and not find this tale of a long-defunct OS, and the long-defunct team that developed it, startlingly relevant. This twenty-five year old book boldly illustrates one point: **computers may change, but people don't.**

Reading this classic work will certainly be a better use of your time than poring over the latest thousand page technical tome du jour.

Don't Make Me Think

The single best book on usability I've ever read. The title says "web usability" but don't be fooled by its faux specificity. Steve Krug covers every important usability concept in this book, and covers it well. It's almost *fun*. **If you choose to read only one book on usability, choose this one**. It's chock full of great information, and it's presented in a concise, approachable format. It's suitable for any audience: technical, non-technical, user, developer, manager, you name it.

Er… yeah. Never been in a meeting like that. The solution to this problem, by the way, is quick and dirty usability testing. Imagine that: making decisions based on actual data instead of never ending, last man standing filibuster style religious debates. Revolutionary!

Rapid Development

The full title of this book is *Rapid Development: Taming Wild Software Development Schedules*, which isn't just long-winded and vaguely ridiculous, it's also an unfortunate misnomer.

Rapid Development isn't about rapid development. It's about** the reality of failure** . The vast majority of software development projects will fail: they will overrun their schedules, produce substandard results, or sometimes not even finish at all. This isn't an argument; it's a statistical fact. The unpleasant truth is that your team has to be very good to *simply avoid failing*, much less to succeed. While that may sound depressing – okay, it *is* depressing– you'll still want to read this book.

Why? Because half* of success is not repeating the same mistakes you, or other people, have made. The epiphany offered in this book is that making mistakes is good– so long as they are all new, all singing, all dancing mistakes. If you're making the same old classic mistakes, you've failed before you've even begun. And you probably have no idea how likely it is that you're making one of these mistakes *right now*.

Our field is one of the few where change is the only constant, so it's only natural to embrace that change and try different "Rapid" development techniques. But the converse isn't true. We can't assume that so much has changed since 1970 that all the old software development lessons are obsolete and irrelevant when compared to our hot new technology. It's the same old story: computers have changed; people haven't. At least have some idea of what works and what doesn't before you start– in McConnell's words, "read the instructions on the paint can before painting." Sure, it sounds obvious enough until you read this book and realize how rarely that actually happens in our field.

* According to the book, technically, one-quarter. But I think it's more than that.

Peopleware

If you've ever seen the performance of an all-star sports team suffer due to poor coaching, you'll appreciate this book. It doesn't matter how many "coding superstars" you've got when none of them can talk to each other, or agree on anything. And it no developer, however talented, can work effectively when constantly being barraged with minor interruptions. Developers aren't known for their people skills, per se, but here's the ironic part: the success of your project may hinge on just that. If you have any legitimate aspirations to be a "Team Leader" in practice instead of in name only, you need to pick up a copy of this book.

While Peopleware is full of great, totally valid points, it also implies a level of employee control over the workplace that is pure fantasy at most companies. But at least you'll know when your work environment, or your team, are the *real* problem – and more importantly, what to do about it.

The Design of Everyday Things

It can be incredibly frustrating to develop software, because so much can go wrong. A lot of what we do is defensive: trying to anticipate what will go wrong before it does. It's mentally fatiguing, and can eventually manifest itself in some negative ways. I sometimes describe this to non-technical people as building a watch with a thousand moving parts, all of which can fail randomly at the slightest provocation. Good times!

Designing software is difficult, to be sure, but *designing a door is difficult too*. The nuances of design extend into every object you touch, whether it's some hot new SQL engine, or a humble shoe. This book will give you a new appreciation of the "devil in the details." If designing a door isn't the no-brainer we thought it was, maybe it's time to give ourselves a break for not being able to design software perfectly, either.

About Face: The Essentials of Interaction Design

Alan Cooper, father of Visual Basic, godfather of usability. I've owned a few versions of this book now (this is version four), and it is the rare book which is getting better and better as it is revised, and more authors are added for different perspectives.

About Face is full of generally applicable guidelines for mobile and web. Of the GUI problems used for illustration – with examples from the hoary old Windows 95 UI – it's interesting to compare which have been mostly resolved (using visual examples to show the effects of dialog selections before you make them), and which have not (stopping the proceedings with modal idiocy).

It's a fantastically useful book; I've used whole chapters as guides for projects I worked on.

The Inmates Are Running the Asylum

This is the book that introduced the world to the concept of **personas:** rather than thinking of users as an abstract, difficult-to-describe, amorphous group of people, personas instruct us to talk about specific users who have names, personalities, needs, and goals. Would our users want a print preview feature? Who knows? But if Gerry Manheim, Account Executive, has to print out his weekly expense report as a part of his job, you better believe print preview needs to be in there. There's nothing magical here; as always, it boils down to knowing who your users are and what they really do – and the personas technique is a great way to get there.

There's also an interesting analysis here of how developers tend to think themselves qualified to make usability decisions on behalf of "regular" users, when in reality they're anything but. Developers are freakish, extreme users at best– "Homo Logicus" versus "Homo Sapiens." Unless you happen to be writing a compiler where developers are the end users.

One hidden lesson in this book is that *sometimes it doesn't matter how good your design is*: the scanner software and the web development software which Alan consulted on, and uses as examples in this book, both failed in the marketplace for reasons that had nothing to do with their usability– which was verifiably excellent.* Sometimes great products fail for reasons beyond your control, no matter how hard you try. Feel free to use this fact to counterbalance the sometimes bombastic tone of the book.

* I owned the exact model of "behind the keyboard" USB scanner pictured in the book, and I was quite impressed with the bundled scanning software. I eventually gave this scanner to my Dad. One time I was chatting on the phone with him and without any prompting at all, he mentioned to me how much he liked the scanning software. This was before the book had been published!

Programming Pearls

I hesitated to include *Programming Pearls* because it covers some fairly low-level coding techniques, but there are enough "pearls" of software craftsmanship embedded in this book to make it well worth any developer's time. Any book containing this graph..

.. is worth its weight in gold. TRS-80 versus DEC Alpha to illustrate 48n versus n^3 algorithms? Come on folks, it just doesn't get any better than that. *Programming Pearls* is the next best thing to working side by side with a master programmer for a year or so. It is the collective wisdom of many journeyman coders distilled into succinct, digestible columns.

I won't lie to you: there are entire chapters that can probably be ignored. For example, I can't imagine implementing sorting, heap, or hash algorithms as documented in columns 11, 13, and 14 respectively, given today's mature libraries of such basic primitives. But for every textbook-tedious exercise, there is real, practical advice alongside. Just scan through the book, ignoring the code sections, and I doubt you'll be disappointed. Column 8, "Back of the Envelope" is essential, probably the best treatment of estimation I've seen anywhere. It also goes a long way towards explaining those crazy interview questions that companies love to annoy us with.
You can read sample sections of the book online if you're still on the fence. I recently used the chapter on strings to illustrate the use of Markov chains in generating synthetic data to fill an empty database with – a performance estimation technique covered in "Back of the Envelope".

The Pragmatic Programmer: From Journeyman to Master

This book reminds me a lot of Programming Pearls, but it's actually better, because it's less focused on code. Instead of worrying about code, the authors boiled down all the practical approaches that they've found to work in the real world into this one book. Not all of these things are technically *programming*. For example, asking yourself "why am I doing this? Is this even worth doing at all?" isn't thinking outside the box; it's something you should incorporate into your daily routine to keep yourself – and your co-workers – sane. And that's what makes Pragmatic Programmer such a great book.

If you'd like to know a little more about the book, I created a HTML version of the pullout reference card included inside, which provides a nice overview of the contents.

Designing Web Usability

Jakob Neilsen is well known for his usability site, and his career as a usability expert extends back to 1989 when his first book was published. *Designing Web Usability* is of course a full-on web usability primer, so it's a bit different than the GUI-oriented Cooper books.

The Visual Display of Quantitative Information

Visual Explanations: Images and Quantities, Evidence and Narrative

Envisioning Information

Beautiful Evidence

Information is beautiful. And so is a well-designed GUI.

You don't need to own all four books in the series unless you're a completist (or a masochist, I suppose), but the first two are essential.

Chris Sells has some interesting insight on the Tufte books based on a Tufte seminar he attended in June 2004.

Regular Expressions Cookbook

UNIX has a well-deserved reputation for being complex and impenetrable. So do Regular Expressions.

I may be a card carrying member of the "Keep It Simple Stupid" club, but I'm making a meteor sized exception for regular expressions. Written properly, they will save you a tremendous amount of time in string manipulation, and I've never run across a project where they didn't come in handy *somewhere*.

Once you delve into the world of regular expressions, you may become drunk with the amazing power and potential they have, which results in things like Perl. Remember, absolute power corrupts absolutely. But it also rocks absolutely.