

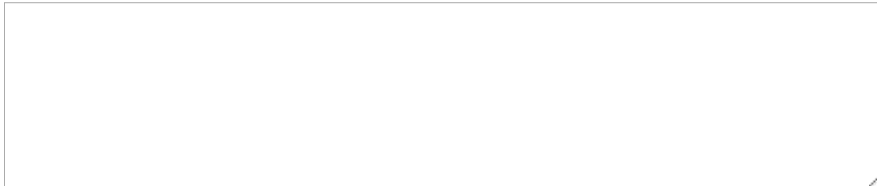
Ask HN: What language-agnostic programming books should I read?

861 points by robschia 1 day ago | hide | past | web | 326 comments | favorite

Until now I've read:

- The Healthy Programmer
- Clean Code
- Test-Driven Development by Example

What should I add to my library?



DoofusOfDeath 1 day ago [-]

Some of the *most interesting* books I've read in support of my software-development work are:

- * "Compilers: Principles, Techniques, & Tools" by Aho et al. (i.e., "the dragon book")
- * "Data Flow Analysis: Theory and Practice" by Khedker et al.
- * "Understanding MySQL Internals" by Sasha Pachev.
- * "Transaction Processing: Concepts and Techniques" by Gray and Reuter.
- * "Fundamentals of Wireless Communication" by Tse and Viswanath.
- * "Genetic Programming: An Intrduction" by Banzhaf et al.
- * "Applied Cryptography" by Schneier.

EDIT: A few additional comments:

(1) Although these books are problem-domain specific, some of them had benefits outside of their problem domains:

- * The Dataflow book has some great coverage of fixpoint algorithms. It's really helpful to recognize when some problems are best solved by fixpoint analysis.
- * The "dragon book" takes a lot of the mystery out of compilers. That's somewhat helpful when writing code that needs to be fast. It's *super* helpful if you want to work with compiler-related technologies such as LLVM.
- * Understanding the fundamental challenges of transaction processing helps you avoid massive misadventures when dealing with databases or concurrent / multithreaded systems.

(2) YMMV, but I've found it hard to soldier through these books *unless* I had a need to for my job or for school.

[reply](#)

barrkel 1 day ago [-]

Don't read the dragon book if you're interested in compilers. It's lex and parse heavy, and is not up to date with more recent best practice on codegen. Further, the lex and parse end of it is focused on the kind of theory you need to build tools like lex and yacc, rather than stuff you need to know if you want to write a compiler.

For a good intro of modern back end development, check out Engineering a Compiler. It's also got the lexical and parsing end of things, but a bit better done, but still quite theoretical.

One of the best books I read on programming is "Concepts, Techniques, and Models of Computer Programming"; it's not strictly speaking language agnostic, based as it is on Oz, but Oz isn't a language you'd ever use in production and is a reasonable base to introduce you to ways of programming that will probably be completely unfamiliar, like constraint programming, dataflow, concurrent logic, etc.

[reply](#)

timtadh 1 day ago [-]

Most people who criticize the Dragon (Compilers etc... by Aho et al.) book seem to focus on chapters 3 and 4 which are the chapters on lexical analysis and parsing. The book has 12 chapters. Whatever your feelings on the parsing techniques, the book covers WAY more than that. It has a really good introduction to code generation, syntax directed translation, control flow analysis, dataflow analysis, and local, global and whole program optimizations.

As someone who has quite a few books on compilers, program analysis, type theory, etc... I find the Dragon book an irreplaceable reference to this day. It has a breadth of content shared by very few other books. For instance, Muchnick's classic "Advanced Compiler Design and Implementation" is really good for analysis and optimization but neglects all front end topics. The only area where I believe the Dragon book is inadequate in is type theory (I recommend Types and Programming Languages [TAPL] by Pierce and Semantics with Application by Nielson for a gentler intro).

As to parsing, its chapter on parsing (4) is not as "hip" as some people want. However, it is solid and will teach you how to do parsing. There are newer and fancier techniques not covered in Chapter 4 but in general most people would benefit just having a solid understanding of recursive descent parsing!

[reply](#)

beat 1 day ago [-]

Don't read *only* the Dragon book if you're interested in compilers (or parsing in general). But it *should* be read, in addition to whatever else you read on the subject. It's classic for a reason.

Most programmers don't understand parsing worth a crap, imho. Some compiler theory would do most of us some good. I read it because I was doing data stream parsing back in the olden days before XML (which was before JSON), when we had to write our own stream formats. It really changed my whole way of thinking about a fundamental class of programming problems.

[reply](#)

eru 1 day ago [-]

I found the Dragon Book unbearable---and I am one of those people interested in theory. Pick something better, plenty of choice available.

'Essentials of Programming Languages' and 'Types and Programming Languages' are good choices. (But slightly off-topic for this particular subthread, since they don't deal with parsing.)

[reply](#)

astrange 1 day ago [-]

The Dragon book is not good at teaching parsing, but is obsessed with the idea and spends 2/3 of the time talking about it, so you'll just be left thinking it's really difficult. The whole thing is extremely boring and lacks confidence.

Try "Parsing Techniques: A Practical Guide" and then a more concise book like Muchnick.

[reply](#)

DoofusOfDeath 1 day ago [-]

I also found my theory-of-computation course in college to be a wonderful help in understanding parsing.

Being familiar with the Chomsky Hierarchy ([1]), and the kinds of language recognizers required for each level in that hierarchy, can save parser authors a lot of wasted time.

E.g., "regex's can't count!"

[1] https://en.wikipedia.org/wiki/Chomsky_hierarchy

[reply](#)

aalhour 1 day ago [-]

Hey, I seconded this, the Dragons Book is not the best textbook or introduction to the topic, there are a lot of better ones. The Dragons Book is more of a classic. If you want to read a good book on compilers construction then you're better off reading "Engineering a Compiler" by K. Cooper and L. Torczon or "Modern Compiler Implementation in ML" by A. Appel.

If you want more resources on Compilers other than books, such as tutorials, guides, talks and papers, then go ahead and check the Awesome Compilers vertical I compiled a while ago: <https://github.com/aalhour/awesome-compilers>

Cheers!

[reply](#)

glangdale 1 day ago [-]

Do note that there are multiple editions of the Dragon book and later editions have a far better codegen section.

One of the more famous compiler writers from that era once told me that the Dragon book's original codegen section wasn't even *historically* how anyone wrote a compiler and it certainly didn't reflect modern practice even back in the 90s.

However, the update significantly improves it with the addition of Monica Lam's contribution. I don't know if it's state of the art for codegen (probably not), but I'm not sure what book is (Muchnick is great too, but this is also quite old now).

[reply](#)

snakeanus 1 day ago [-]

CTMCP is awesome indeed, one of my favourite along with SICP I would say.

[reply](#)

eastWestMath 1 day ago [-]

Writing a parser is broadly applicable to a wide range of problems, not just writing a compiler.

[reply](#)

barrkel 1 day ago [-]

The criticism is that the strongest part of the book is focused on writing compiler compilers - that is, writing tools that write parsers - not writing parsers.

[reply](#)

eru 1 day ago [-]

My criticism is that the book is unbearably imperative and doesn't actually teach you any of the theory well.

[reply](#)

k__ 1 day ago [-]

yes.

Wrote many for XML/json based configs.

[reply](#)

TallGuyShort 1 day ago [-]

I would recommend Schneier's Cryptography Engineering (or its predecessor) Practical Cryptography over Applied Cryptography, given the question. I agree Applied Cryptography is a very interesting book, but that's because I'm a crypto nerd. For someone looking to improve general software engineering skills, I think the other books do a better job of presenting best practices and the understanding behind them, with a bit less mathematical theory, etc.

[reply](#)

schoen 1 day ago [-]

Also, Applied Cryptography includes a ton of totally obsolete stuff, which tptacek has complained is still leading people astray by giving them seemingly attractive choices that are in fact broken or deprecated. Even at the time, with its focus on breadth of coverage, it presented technology which wouldn't have been a state-of-the-art choice, while today many techniques there are even less appropriate (and some good new crypto has been invented in the interim).

Applied Cryptography was an important political statement that we can and should have access to cryptographic technology, but I don't think it can be recommended as an appropriate introduction to the field in 2017.

[reply](#)

sethrin 1 day ago [-]

> "Understanding MySQL Internals" Why this, as opposed to a more general text on databases? MySQL is popular, but few would accuse it of actually being a good database. I feel like the point at which knowing about the internals of MySQL becomes relevant is probably the point at which you should use a better database.

[reply](#)

DoofusOfDeath 1 day ago [-]

That's a good point; let me clarify.

It's definitely a good idea to have prior general knowledge of database internals before reading an "internals" book.

I'm not sure if it's out of date, but my favorite book on the topic was "Database System Implementation" by Garcia-Molina et al. I found it to be super-readable.

What I like about the "internals" book is you get to see a *real, specific* database up close, warts and all. It gives you a different perspective from an overview book like the one mentioned above.

[reply](#)

fipar 1 day ago [-]

I think a good book that helps you understand a complex piece of software is a good read for programmers, regardless of the use you make of that software.

Additionally, I don't see how the need to know about the internals of MySQL means one should use a better database. PostgreSQL is normally proposed as the better alternative to MySQL if one sticks to Open Source, and all the good experts at it I have met were familiar with its internals too.

[reply](#)

sethrin 1 day ago [-]

You seem to have misinterpreted my objection. There are better books on databases, and MySQL is not necessarily a good example of a database. It tends to do things in its own way. When you get into the sort of situation where the difference between 'how MySQL does things' and 'how things are supposed to work' matters, then typically it's MySQL that's quirky.

[reply](#)

fipar 1 day ago [-]

Thanks for taking the time to reply.

I think your last sentence is what caused me to misinterpret you. I agree with your comment up to "... being a good database.", but I think the point at which you need to know the internals of MySQL are irrelevant to whether you should use a database that is better for the problem you're trying to solve or not.

[reply](#)

jerf 1 day ago [-]

I would assume that any book about the internals of a database would be suitable. MySQL is plenty real enough to be educational to learn about its internals, yea verily, even its quirks, and to learn a lot about how other databases work. Even the "NoSQL" databases will use many of the same primitives.

An equivalent substitution for another database would be fine; anyone have any suggestions? Then again, such a substitution is really solving a non-problem.

[reply](#)

DoofusOfDeath 1 day ago [-]

It may very well be the case that looking at the internals of some database other would provide a different set of lessons.

IMHO, MySQL is especially ugly due to an attempt to insert an abstract interface around its storage engines. IIRC that interface is leaky, and its documentation is pretty spotty. Also, the InnoDB storage engine has a *lot* of complexity that seemingly duplicates what's provided on the storage-engine-agnostic side of that interface.

One of the big lessons I took away from studying MySQL's internals, and partially confirmed by the "internals" book, was this: Be careful when designing a pluggable storage-engines framework for a DBMS; MySQL has some examples of what can go wrong.

That's something you're unlikely to find mentioned in a generic DBMS-implementation book, but is very good to know about regardless.

[reply](#)

eof 1 day ago [-]

And just abandon all that domain knowledge your garnered along your journey?

[reply](#)

tannhaeuser 1 day ago [-]

+1 for the "dragon book". Though the criticism about it being too focused on lex/yacc-era compilers and not being relevant today might have a point, I still find the mix of pragmatism and formal language theory absolutely worth the read, stylish/Unix-y, and even entertaining - for me, truly a hallmark of "American-style" comp.sci. standard literature (as opposed to more theoretical expositions to the topic).

Also, Donald Knuth's "The Art of Programming" (at least the fundamental and sorting/searching algorithms books), plus a book about graph theory and algorithms.

[reply](#)

pvg 1 day ago [-]

You can probably make a book out of the recommendations for better books than dragon and applied crypto you can find just online.

[reply](#)

mikekchar 1 day ago [-]

Working Effectively with Legacy Code by Michael Feathers. It's a bit hard to wrap your brain around the Java and C++ examples unless you have experience with them, but the techniques are timeless. You may need to practice them extensively before you understand how important they are, though. In a recent book club we did at work, a common complaint was, "This just looks like common sense". Indeed it does... though the common sense is uncommonly hard to find when you are staring at the actual situations this book helps you with.

[reply](#)

sepent 1 day ago [-]

I just finished reading this book yesterday. I am not an expert in refactoring but the book seems too old, although most advises could be still useful. Anyway, today I found a new book by the author: Brutal Refactoring: More Working Effectively with Legacy Code. Unfortunately, I couldn't find a good review of the book on the Internet. But I think it could be more useful.

[reply](#)

fsloth 1 day ago [-]

"the book seems too old"

2004 is old now? Really?

Honestly, good books don't age as long as their core domain stays valid. Structurally C++ is pretty much the same as when the book came out.

[reply](#)

johndubchak 1 day ago [-]

Modern IDEs can deliver many of the recommendations of the book. Our productivity has increased a great deal. Back then there was no intellisense, code navigation through clicking on method names/classes etc. highlighted syntax errors, built-in unit test frameworks etc.

It was written for a different type of developer and a different type of development environment and a specific language, C++.

You haven't looked at C++ lately if you think it's the same as it was in 2004. The ISO has released new versions of the language in 2011, 2014 and ratified a new standard here in 2017. If you're writing C++ code that is consistent with 2004 C++ then you're writing a really bad version of "C with classes", not C++.

Edit:

Modern C++ contains native support for the filesystem, threads, lambda expressions, variants, upcoming networking library, coroutines (at least in Visual Studio), no more new/delete memory management, parallel algorithms and a ton more. This is a completely different language now and the code you write looks nothing like 2004 C++ code.

[reply](#)

fsloth 1 day ago [-]

"You haven't looked at C++ lately if you think it's the same as it was in 2004."

"If you're writing C++ code that is consistent with 2004 C++ then you're writing a really bad version of "C with classes", not C++."

I applaud your attempt at an authoritative voice. But you focus on mostly technical trivia that are thin scaffolding on top of the language. I agree modern C++ is nice but it's the same language still.

"This is a completely different language now and the code you write looks nothing like 2004 C++ code."

Are you trolling? This reads like a transcript from a TV commercial.

[reply](#)

khedoros1 1 day ago [-]

They sound like they're echoing the kinds of things that are normally said about C++11.

> I agree modern C++ is nice but it's the same language still.

Well...in the same sense that *any* language is the same language after you add a bunch of things to it that weren't there before and shift to using those new features as idiomatic parts of the language. I *would* expect C++ written in 2004 to use different patterns than C++ written in 2017. Not "completely different"...but different.

[reply](#)

fsloth 1 day ago [-]

Idiomatic sounds like preference to *dogmatism* rather than *pragmatism* which is generally the worse tradeoff.

I would say generally C++ style has evolved through last decades with people understanding class based architecture as an antipattern and data pipelines based on preferably immutable data as the more robust and understandable approach. This has nothing to do with language standards or 'idiomatic' constructs - both can be expressed as perfectly elegant C++, using the '98 or '11 or '17 variant.

But anyway, within this context - refactoring old code - I would not expect a legacy codebase to resemble 2017 C++ as much as 1997 C++. This is the main

reason I find the claim of methods to understand circa 2004 C++ to be outdated to be silly.

reply

khedoros1 1 day ago [-]

> Idiomatic sounds like preference to dogmatism rather than pragmatism which is generally the worse tradeoff.

That's never been the way that I've read it, and it's not how I meant it. New language constructs allow for more-natural ways for the code to express the intent of the programmer, replacing the use of older constructs *in the places that they were clumsy*.

The entire point is to make the language more practical. Patterns of use in a language aren't idiomatic because of dogma (or at least, they shouldn't be). They're idiomatic because they're a clear and elegant way (or at least the most elegant way available) to implement something.

> I would not expect a legacy codebase to resemble 2017 C++ as much as 1997 C++.

Agreed, but then we come back to the fact that we might refactor a 1997 codebase differently in 2004 than in 2017.

reply

fsloth 2 hours ago [-]

"Agreed, but then we come back to the fact that we might refactor a 1997 codebase differently in 2004 than in 2017."

Well, if the original code is of the worst kind of a mess, Feather's circa 2004 collection of methods to make it more understandable but functionally the same work just fine for the first part of the refactoring. In my experience this is the most difficult part as well. To what dialect of the language the code is ported after it is understandable, is a relatively trivial syntax transform after this.

I speak from experience from having recently had to implement features to a production codebase with millions of lines of code, some of which date back to Fortran, and that took the final step of evolving into C++ sometimes in the late 90's.

I prefer the definition of legacy code that it's any code that does not have unit tests. In this case every transformation to a production codebase needs to retain the original behavior - without exact understanding what that behavior is.

I think this comment should have been my original response to this thread instead of the relatively cheeky responses I wrote earlier.

reply

mattmanser 1 day ago [-]

Intellisense (the MS one) was introduced in 1996. VB6 had good intellisense. I seem to remember having intellisense while working in classic asp vb (as in pre .Net), but could be wrong, it's so long ago now.

[reply](#)

fatso83 10 hours ago [-]

Modern IDEs can do many of the refactorings required when modifying code, yes. But you still need to know where to find seams or how to create them. Refactorings are just tools for getting there. It doesn't seem like you have read the book.

[reply](#)

atilaneves 1 day ago [-]

A lot of people have to maintain legacy C++98 codebases and don't get to play with new compilers.

[reply](#)

pvg 1 day ago [-]

Our productivity has increased a great deal. Back then there was no intellisense, code navigation through clicking on method names/classes etc. highlighted syntax errors, built-in unit test frameworks etc.

Just about all of these things were around in 2004.

[reply](#)

hinkley 1 day ago [-]

Every day I wished I'd listened to the XP guys the first time I encountered it, and most of that material is from the late 90's.

And Fred Brookes' book is over 30 and sadly just as relevant.

[reply](#)

johndubchak 1 day ago [-]

Me and a friend continually send IM's to each other, "LEAN ON THE COMPILER!".

That statement alone, one of the sections of the book, recommending you to allow the compiler to find your errors for you, is an example of the "age" of the book.

At the time I read it, when it first came out, I loved it. I still love Michael's work and advice to this day, but this book was written for another time.

[reply](#)

TeMPORaL 1 day ago [-]

I do not understand this. It seems like a perfectly good advice, and in fact it's what I'm doing every day - leaning on my compiler, and in Java world, on my IDE.

And also it seems to be something lots of programmers don't realize for some reason - many times I had to instruct people to crank up their compiler warning settings and *actually read them*. Especially in C-land, I can't count the number of times I solved someone's problem by appending -Wall to the gcc invocation and telling them to come back after they fixed all the warnings...

[reply](#)

johndubchak 1 day ago [-]

My point was that with a modern IDE we can do that without resorting to a full compile, which can be burdensome with large codebases.

The IDE space has improved a lot since 2004, eg. JetBrains' tools around refactoring and code cleanup suggestions, make things super simple.

Don't get me wrong, I've read this book, multiple times, it's on my bookshelf and think it's a great book, but it was written when the state of development was a much different landscape, IMO.

[reply](#)

biot 1 day ago [-]

Leaning on incremental compile via a modern IDE *is* leaning on the compiler. The *principle* is the same, even if the *implementation* slightly differs.

[reply](#)

mikekchar 1 day ago [-]

Just looking at the responses to this comment, I have to say that I agree with both sides. The book is *really* dated, but the advice is timeless. The biggest problem I've had trying to get people to take the book seriously is that they can not identify with it. I had not realised that Michael Feathers had written a followup. I will definitely take a look. Thanks!!!

[reply](#)

otodic 1 day ago [-]

The Pragmatic Programmer: <https://pragprog.com/book/tpp/the-pragmatic-programmer>

[reply](#)

suyash 1 day ago [-]

Plus 1 to this one and another one by the same Publisher is 'Pragmatic Thinking & Learning Refactor your Wetware'

[reply](#)

beat 1 day ago [-]

That's by Andy Hunt, one of the Pragmatic Programming authors. Pragmatic Programmer, as a publisher, has produced dozens of books. They're like another O'Reilly.

[reply](#)

d0m 1 day ago [-]

Structure and Interpretation of Computer Program (it's written in scheme but it's mostly for convenience and its lack of syntax).

Definitely the best book I've read on programming.

[reply](#)

eru 1 day ago [-]

I re-read SICP recently, because I was teaching a friend.

I had very rose-tinted nostalgic memories. They did not hold up.

"How to Design Programmes" is a much better book. (But made for rank beginners.) And for the more advanced, "Types and Programming Languages".

See "Why Calculating is better than Scheming" (<https://www.cs.kent.ac.uk/people/staff/dat/miranda/wadler87....>) for a critique.

Don't get me wrong, SICP is still a good book to have read. Just no longer my favourite.

[reply](#)

zerr 1 day ago [-]

And original MIT video lectures (for HP employees) are must watch. The mood/feel they have, impossible to receive from the book alone.

[reply](#)

antfarm 1 day ago [-]

<https://www.youtube.com/watch?v=2Op3QLzMgSY&list=PLE18841CAB...>

[reply](#)

Chaebixi 1 day ago [-]

> And original MIT video lectures (for HP employees) are must watch. The mood/feel they have. impossible to receive from the book alone.

Link?

[reply](#)

ucsdrake 1 day ago [-]

I assume it's: <https://www.youtube.com/playlist?list=PLE18841CABEA24090>

[reply](#)

gm-conspiracy 1 day ago [-]

Link, please?

[reply](#)

tarp 1 day ago [-]

<https://www.youtube.com/playlist?list=PLE18841CABEA24090>

[reply](#)

chii 1 day ago [-]

I'd find it quite hard to apply lessons from SICP using java or c...

[reply](#)

jerf 1 day ago [-]

These are indications of the weakness of Java and C, not SICP.

I do not mean this as empty snark; it is something you can see across the board in programming language evolution. It is virtually impossible to propose a new general-purpose programming language today that does not have closure support, which is the biggest thing you're finding missing from C and Java to use the techniques in SICP. (It is not the *only* thing, but it is the *biggest* thing.) You'll note Java is going to bolt them on soon, or recently bolted them on, but it'll be another decade before Java is using them fluently either way.

I've often recommend SICP to self-taught programmers; it is not a complete substitute for a computer science education, but it is a concentrated dose of the stuff you're probably missing out on. Expect it to be the work of some months to work through it. However, in the end, a self-taught programmer who has also worked through SICP systematically and carefully is probably a better programmer than 80% of the modern computer science graduates nowadays. (If you want to avoid that fate, oh ye who are still in school, take the freaking compilers course at the very least. Do your peers complain that it is hard? Then maybe it is a good one.)

[reply](#)

jcadam 1 day ago [-]

The "Compiler Design" course at my university was the hardest CS course I've ever taken -- and it was generally taken your very last semester (and was infamous for being a common cause of many senior CS students not graduating on time).

Had it been optional, I would have been sorely tempted to skip it. But, taking it was probably good for me in the long run :)

[reply](#)

pvg 1 day ago [-]

They're indications the book is also not 'language-agnostic', for all of its merits. The language is there for pedagogic purpose.

[reply](#)

jerf 1 day ago [-]

I don't think a book is "not language agnostic" because two particular languages aren't very good at it. C's lack of closure support is unique among any language still in use, for a variety of reasons. Java... was Java. If you weren't there you may not remember or understand, but at the time OO was at its most dogmatic and either it was an object or it was not included. Anything you could do with a closure could be done with an anonymous inner class. (Which is, technically, true. But the amplification factor on the effort and the sheer quantity of *text*

involved to do so in Java is astonishing.) You can hear some of the echos of this in the older Ruby rhetoric, though I haven't heard it in a while.

Any other general-purpose language you'd be inclined to pick up right now would be fine.

IIRC, you end up implementing a scheme, which you can do nearly in the language of your choice, then the book covers that language you are implementing.

reply

pvg 1 day ago [-]

The book is not language agnostic because it's not language agnostic, mostly. The fact that many other languages are a poorer fit for what it's trying to teach is pretty much the definition of 'not language-agnostic'.

reply

jerf 1 day ago [-]

I don't know what task could possibly exist for which there are not significantly better and worse languages. Even today's "list of practical projects that anyone can solve in any programming language" [1] I can see at a casual glance that any of them that are not simply trivial will have better and worse languages. A definition of "language agnostic" that either covers no tasks, or covers only very trivial tasks, isn't very useful. (Although in the latter case one can attain some insight by pondering why only such trivial tasks might be considered "language agnostic".)

[1]: <https://news.ycombinator.com/item?id=14481941>

reply

pvg 1 day ago [-]

I'm not sure what any of these logical contortions have to do with SICP being 'language agnostic'. It isn't. The first section of the first chapter talks about why a Lisp dialect is used. This is the first figure.

<https://mitpress.mit.edu/sicp/full-text/book/ch1-Z-G-1.qif>

Etc.

reply

fsloth 1 day ago [-]

"I'd find it quite hard to apply lessons from SCIP using java or c"

IMO, you're supposed to write your own Scheme to reap full benefits from SICP.

The book contains a recipe for an interpreter that can be transferred to any other language. Chapter 5.4 : The explicit-control evaluator[0].

Strangely this fact is not advertised that much, I find it to be one of the coolest features of the book.

The implementation is not a direct syntax translation - one needs to figure out how to implement the base substrate and how to handle e.g. tokenization, but well rounded software engineer should have some model in ones head how to manage these (so if there are missing pieces this is a good opportunity to brush on those).

[0] <https://mitpress.mit.edu/sicp/full-text/book/book-Z-H-34.htm...>

reply

Pitarou 1 day ago [-]

After SICP and a little study of functional programming, I started learning about design patterns, and my immediate reaction was, "What's all the fuss about? I mean ... c'mon ... how else would you solve that problem?"

It took me a while to understand that, if you've been raised on Java or C, these things are not obvious at all.

[reply](#)

codehd7 1 day ago [-]

It's been "redone" for Python: <http://www-inst.eecs.berkeley.edu/~cs61a/sp12/book/>

[reply](#)

yomly 1 day ago [-]

What were your main takeaways from SICP?

While it certainly espouses a functional style, I found the greatest lessons to be higher level, focusing on abstraction.

Though I have limited knowledge of C and Java so would be keen to hear your elaborated thoughts on this.

[reply](#)

chii 1 day ago [-]

when you start to get to the parts about abstracting a circuit, and making it streamy/lazy, you'd find that the impl details of a non-lazy language can make it more difficult than it needs to be.

Also, as a beginner, you don't want both the difficulties of concepts `_and_` the difficulties of language to combine to stop you from learning as effectively.

[reply](#)

yomly 1 day ago [-]

But Scheme isn't a lazy language - granted, you'll need a language that supports first class functions to lift their implementation of a stream though.

>Also, as a beginner, you don't want both the difficulties of concepts `_and_` the difficulties of language to combine to stop you from learning as effectively.

Are you referring to the need to learn Scheme to work through SICP? While the asker specifically requested language-agnostic books, Scheme was chosen as a language for SICP for being a "syntax-free" language...

[reply](#)

gumby 1 day ago [-]

> Are you referring to the need to learn Scheme to work through SICP?

That's not particularly demanding requirement. The class began with a single lecture about how Scheme worked and after that Hal and Gerry assumed you knew all the language you needed to and dove in.

When the course was introduced (early 80s), most incoming MIT freshman had not programmed a computer (!), including those who planned to do course 6 (EE or CS -- 6.001 was a requirement for either degree).

[reply](#)

chii 1 day ago [-]

> Are you referring to the need to learn Scheme to work through SICP?

no, i'm referring to the fact that this book isn't language agnostic (i.e., you wouldn't want to use anything other than scheme/lisp).

[reply](#)

pvg 1 day ago [-]

Scheme was chosen as a language for SICP for being a "syntax-free" language...

First-class functions, lambdas, closures, lexical scoping, etc. Abelson wrote, for instance:

"6.001 differs from typical introductory computer science subjects in using Scheme (a block-structured dialect of Lisp) rather than Pascal as its programming vehicle. The subject's developers feel strongly that Pascal is hopelessly constraining, and that important ideas (such as functional programming and object-oriented programming) can be addressed within Pascal only awkwardly, if at all."

[reply](#)

evbots 1 day ago [-]

Second this. I'm making my way through the book now. I'm impressed by how well it articulates concepts from the ground-up.

[reply](#)

suyash 1 day ago [-]

Plus 1

[reply](#)

petercooper 1 day ago [-]

Programming Pearls by Joe Bentley. And its followup. It's old but it does get you thinking about things.

I'd also recommend *The Linux Programming Interface* by Michael Kerrisk as it teaches so much about what makes modern Unix what it is but.. it's arguably quite oriented around C by necessity. It's not a "C book" by any means though.

[reply](#)

kristjan 1 day ago [-]

Correction: Jon Bentley

[https://en.wikipedia.org/wiki/Jon_Bentley_\(computer_scientis...](https://en.wikipedia.org/wiki/Jon_Bentley_(computer_scientist))

[reply](#)

neves 1 day ago [-]

Can't say enough good things about Programming Pearl. Don't forget to do the exercises (or at least think about them).

[reply](#)

henrik_w 1 day ago [-]

I really like:

- Code Complete by Steve McConnell
- The Effective Engineer by Edmond Lau
- The Pragmatic Programmer by Andrew Hunt and David Thomas

[reply](#)

stinos 1 day ago [-]

Second the Pragmatic Programmer, however: I read it when I just started programming (like not much more experienced than basic hello world and some fiddling) and, looking back, didn't really get it. I mean, I got the principles, but couldn't apply it as there was just not enough experience for the higher level abstract stuff. I recently read it again and it makes complete sense now, but I didn't learn much new from it. So I'd suggest that if you read it, do so after a couple of years of programming.

[reply](#)

danielbarla 1 day ago [-]

Indeed, the Pragmatic Programmer book's subtitle is "from Journeyman to Master", and that's exactly what journeyman is supposed to mean - someone who's a few years into it, not bad, but not great (yet?).

[reply](#)

mytec 1 day ago [-]

Yes to Code Complete. I read the first edition at a time I wasn't sure if I'd pursue a career in computer programming. I hadn't used a computer for nearly a month making that decision. After reading and re-reading that book during that time, I pursued programming with a passion. That book struck all the right chords with me.

[reply](#)

patrick_99 1 day ago [-]

I read the second edition of Code Complete when I started my first job. Very good book to transition from fresh grad to professional. It has some very dry sections (eg. a chapter on how to name variables) but these are the little things that make a big difference and allows you to 'level up' as an engineer.

[reply](#)

wenc 1 day ago [-]

I really liked The Effective Engineer because it had more systems thinking behind rather just how to manage a code project.

It covers some really foundational concepts like idempotency which programmers don't often think about when architecting systems.

[reply](#)

rusanu 1 day ago [-]

I second Code Complete

[reply](#)

pier25 1 day ago [-]

I third it

[reply](#)

vram22 1 day ago [-]

I agree with your 1st and 3rd titles. Both are really fundamental and highly recommended. Will check out the 2nd one.

[reply](#)

henrik_w 1 day ago [-]

Yes, The Effective Engineer is newer and not so well known. I wrote about why I like it here: <https://henrikwarne.com/2017/01/15/book-review-the-effective...>

[reply](#)

vram22 1 day ago [-]

Great, thanks. Will read it.

[reply](#)

H_Romeu_Pinto 1 day ago [-]

> - Code Complete by Steve McConnell

It has good parts. But it has some very bad parts too, when it reinforces myths that, IMO, are very wrong (e.g.: cone of uncertainty, X times better programmer, etc).

I'd recommend to read it with care.

> - The Pragmatic Programmer by Andrew Hunt and David Thomas

After many years I still read parts of it, again and again. And I still like it.

[reply](#)

kris-s 1 day ago [-]

Code: The Hidden Language of Computer Hardware and Software by Charles Petzold. I love this book.

[reply](#)

throwaway7645 1 day ago [-]

Very good. A good book to go with it is: Learning Computer Architecture with the Raspberry Pi.

[reply](#)

bogomipz 1 day ago [-]

This looks amazing(and affordable). I had not heard of this. Thanks for sharing!

[reply](#)

throwaway7645 1 day ago [-]

No problem. It is surprisingly advanced too and was co-authored by Eben Upton who started the Raspberry Pi. Nice whirlwind tour of computing architecture and history.

[reply](#)

jrmg 1 day ago [-]

This is an amazing book. Hard to describe, and maybe not even all that interesting sounding if you managed to describe it accurately, it's actually fascinating.

If you don't know much about hardware (and maybe even if you do) it'll change the way you think about computing devices.

[reply](#)

JackFr 1 day ago [-]

You hear people say "Computers are all just 0's and 1's", which when you sit down at a modern computer interface is virtually meaningless. In an accurate, but still comprehensible way, 'Code' takes you from the AND's and XOR's and 0's and 1's to the experience you know from a modern computer interface. It is a brilliant book.

[reply](#)

mukeshm 11 hours ago [-]

I am currently reading it and would definitely suggest it.

[reply](#)

doktrin 1 day ago [-]

Seconded. I'm one of those savages who didn't really find much value in classical tomes like MMM or Code Complete (probably because all their core points have already been distilled into contemporary best practices) - but I love this book, and even recommend it to non-technical friends & family.

[reply](#)

drivers99 1 day ago [-]

Also check out The Pattern on the Stone, a similar book by the inventor of the Connection Machine.

[reply](#)

JonoW 1 day ago [-]

I *loved* this book, a must read for any dev

[reply](#)

gigonaut 1 day ago [-]

I'll also +1 this, it is a great read. Has to be top 5 interesting books I have read over the years.

[reply](#)

juliangamble 1 day ago [-]

Agree - this is an amazing book.

[reply](#)

eriknstr 1 day ago [-]

"The Design of Design: Essays from a Computer Scientist" by Frederick P. Brooks [1] is language-agnostic and worth reading.

It's about software engineering but also about hardware and some different kinds of design outside of IT.

From an interview about the book [2]:

> Eoin: Your new book does talk about software design in places, but it's really about design generally, and the case studies span buildings, organizations, hardware and software. Who is the book aimed at? Are you still writing primarily for people who design software or are you writing for a broader audience?

> Fred: Definitely for a broader audience. I have been surprised that The Mythical Man-Month, aimed at software engineers, seems to have resonated with a broader audience. Even doctors and lawyers find it speaks to some of their team problems. So I aimed this one more broadly.

Brooks is also the author of The Mythical Man-Month which is often mentioned on HN.

[1]: [http://www.informit.com/store/design-of-design-essays-from-a...](http://www.informit.com/store/design-of-design-essays-from-a-...)

[2]: <http://www.informit.com/articles/article.aspx?p=1600886>

[reply](#)

codemac 1 day ago [-]

Just purchase this book after finding a pdf, it's really quite great, until you get to View/360 and get jealous.

[reply](#)

pjmorris 1 day ago [-]

'Implementation Patterns', Kent Beck. A semi-language-agnostic extension of his 'Smalltalk Patterns' for how to clearly and consistently express what you're saying when you code.

'Facts and Fallacies of Software Engineering', Robert Glass. Glass presents a list of things everybody knows, or ought to know, and gives both academic and opinionated support and/or critique for why they are and aren't so.

'Making Software', Oram and Wilson. An edited collection of papers on evidence-based software engineering.

'The Deadline', Tom DeMarco. A thinly disguised commercial for his advice on how to organize software development teams and organizations, packaged as light, light novel.

[reply](#)

brightball 1 day ago [-]

SQL Performance Explained - Markus Winand

- Excellent book that gets into the internals of what developers need to know about SQL and covers each part as it relates to the 4 major SQL databases (Oracle, SQL Server, Postgres, MySQL)

- Also has an online version: <http://use-the-index-luke.com/sql/table-of-contents>

The Code Book - Simon Singh

- It's just a good read that covers cryptography and message hiding throughout history. Probably a solid book for somebody around high school age.

[reply](#)

stephenwilcock 1 day ago [-]

+1 for SQL Performance Explained. One of the best vendor agnostic resources for understanding how databases work.

[reply](#)

alecco 1 day ago [-]

I like his writing but that is not "language-agnostic"

[reply](#)

gtrubetskoy 1 day ago [-]

One that hasn't been mentioned yet: "Coders at Work". A very enlightening book about how some of the best programmers in the world approach the craft in their own words.

[reply](#)

LostInTheWoods2 1 day ago [-]

Also noteworthy in that series is Founders at Work, and Gamers at Work.

[reply](#)

henrik_w 1 day ago [-]

Yes, "Coders at Work" is great!

[reply](#)

emodendrocket 1 day ago [-]

Yes, this book is really interesting.

[reply](#)

badperson 1 day ago [-]

loved that book, haven't read it in a number of years, I should break it out again.

[reply](#)

alfiedotwtf 1 day ago [-]

Books I'll treasure forever:

- Advanced Programming in the Unix Environment by Stevens
- Operating Systems: Design and Implementation by Tanenbaum
- The Art of Unix Programming by ESR
- Parsing Techniques by Grune and Jacobs
- Applied Cryptography by Schneier

[reply](#)

vram22 1 day ago [-]

> - The Art of Unix Programming by ESR

Yes. TAOUP has a lot of good content about Unix programming styles and traditions, much of it useful generally. ESR's writing style is a bit heavy / verbose / uses verbal flourishes, but if you can let that not put you off, the book is worth reading. I've read most of it. One of the good sections is about Rules (of programming in Unix, which are not really rules, of course, but informal guidelines developed over many years of experience by many people - ESR has sort of codified them, a thing he tends to do a lot :)

The Rule about separating the interface from the implementation is a good one. I've seen developers at even enterprise companies sometimes not follow it in practice.

> - Parsing Techniques by Grune and Jacobs

Just saw that there is a new edition of this book:

https://dickgrune.com/Books/PTAPG_2nd_Edition/

[reply](#)

microsage 1 day ago [-]

I'll second The Art of Unix Programming - despite the title, it has many broad programming and software architecture lessons, and "The Unix Philosophy" is applicable far beyond Unix.

[reply](#)

sethrin 1 day ago [-]

I have a pretty negative opinion on ESR these days. I could find technical arguments about many aspects of his good pieces of writing, and much of his writing is not good. His abilities as a coder are not generally remarkable, either. But mostly I don't think he's a good person. I would probably not say "Don't read ESR", but I think it has to be "Read ESR* "

*but be aware that he's a polarizing figure and opinions of his merits differ

(and if you happen to have a differing opinion, please use the reply button instead of the downvote button)

[reply](#)

eru 1 day ago [-]

I share your sceptical opinion about ESR, but I concur with the other commenters that The Art of Unix Programming is still worth reading. (In contrast eg to his thing about Bazaar vs Cathedral.)

[reply](#)

sethrin 1 day ago [-]

The Cathedral and the Bazaar was not an excellent piece of writing, but it was sort of necessary at the time. It has served its evangelical purpose well, to the point where it's no longer necessary. Open Source won the day, in no small part due to ESR. (I think it vital to give my enemies their due praise).

TAoUP is a good book on the Unix Philosophy, but it's worth noting that the ultimate expression of these ideas (Plan 9) was a failure, and the 'everything is a file' metaphor is arguably incorrect. I hope I didn't give the impression that it was not worth reading, especially in the sense that there are few other good sources for that information (even using Unix is not likely to teach you much about its whys and wherefores). I just think that it should be read in the proper context, and that recommendations should try to include that context.

[reply](#)

alfiedotwtf 19 hours ago [-]

Yep.

And I'd bet that a lot of programmers wouldn't be working on and with Linux right now if they hadn't have read that book long ago.

Both of them changed my career path from Dos/Windows to Linux

[reply](#)

BlanketLogic 1 day ago [-]

+1 for Parsing techniques. Excellent book.

[reply](#)

solatic 1 day ago [-]

Nobody recommended The Phoenix Project yet by Gene Kim?

Unless you have some understanding of your system's architecture, how it's run in production, and why a production environment is Really Different and a Big Freaking Deal, and how operations is supposed to look like, you'll never be an effective programmer, no matter whether you run your own operations in a small start-up or work for a large enterprise with dedicated operations teams.

[reply](#)

fazkan 1 day ago [-]

Its odd that no one mentioned it but head first into design patterns is a great/light book on design patterns....

[reply](#)

matt_s 1 day ago [-]

I'd second the Head First Design Patterns. The style of writing is light and fun. There are definitely concepts about abstraction, etc. that may take a while to grok and the different approaches help.

[reply](#)

fbomb 1 day ago [-]

YMMV. I found the style extremely distracting to the subject matter.

[reply](#)

eru 1 day ago [-]

They might work for some, but I always find all the "Head First" books to be very fluffy.

[reply](#)

vram22 1 day ago [-]

I think that is part of their goal / style. They are meant to be lightweight and fun and easy-to-read intros to topics for beginners, who would get put off by more formal / less joke-y prose.

[reply](#)

mcguire 1 day ago [-]

My impression was that they were aimed at passing certification tests. But that may just be the couple I looked at.

[reply](#)

vram22 16 hours ago [-]

Didn't think of that. You could be right.

[reply](#)

vram22 1 day ago [-]

A bit like the Dummies series of books, in fact. Which is not to say that those books are not good. I've read a few of them, and they have some good matter.

[reply](#)

hoorayimhelping 1 day ago [-]

I loved this book. It really helped me understand the power and effectiveness of OO design when used correctly.

[reply](#)

deepaksurti 1 day ago [-]

- The Elements of Computing Systems: Build the virtual hardware and software from scratch. Software includes writing a compiler in a language of your choice, so agnostic in that sense. - The Art of Metaobject Protocol: Extremely insightful treatment of OOP! Alan Kay called it as the 'best book in ten years' at OOPSLA 97.

[reply](#)

lotophage 1 day ago [-]

Also in MOOC form now too:

<https://www.coursera.org/learn/build-a-computer>

[reply](#)

i_feel_great 1 day ago [-]

Structure and Interpretation of Computer Programs.

I have attempted some of the problems in Lua, Python, Erlang and Ada. It is very doable. So not just for Scheme.

[reply](#)

jaddood 1 day ago [-]

Agreed. Scheme itself is "not much of a language." It's like a language-agnostic language.

[reply](#)

0xbadf00d 1 day ago [-]

I would recommend "Sorting and Searching" Volume 3 from Donald Knuth's "The Art of Computer Programming". Fantastic, in-depth read.

[reply](#)

twohlix_ 1 day ago [-]

I also emphatically suggest D.Knuth's "The Art of Computer Programming". Super dense, not like a sit down and read from cover to cover, but really good set to have around.

[reply](#)

Posibyte 1 day ago [-]

I also heavily recommend this. It was gifted to me from a professor at my university while leaving, and I left it alone for a few years.

But when I finally opened it up, wow. You can appreciate just how dense the information is. It really reads as if no sentence was written without some purpose of conveying information. Definitely go slow reading this series so you can properly ingest what's written.

[reply](#)

xemdetia 1 day ago [-]

Another Knuth-related one that is less programming that I found rewarding to read was 'Concrete Mathematics.' It is per its own description a lengthier version of the TAOCP, but it's mostly or entirely exempt of actual code. But because of the subject matter it spawned from it tends to cover math of 'what comes later' so there is a very specific perspective of Knuth style that comes from it and it's easier to see the eventual application.

[reply](#)

_asummers 3 hours ago [-]

I had a professor who would joke that it was called Concrete Mathematics both because it was foundational math for computer science, but also because it was as hard as concrete.

[reply](#)

qpre 1 day ago [-]

Not a book per say, but "Out of The Tar Pit" by Moseley and Marks is definitely a must-read.

Abstract:

``` Complexity is the single major difficulty in the successful development of large-scale software systems. Following Brooks we distinguish accidental from essential difficulty, but disagree with his premise that most complexity remaining in contemporary systems is essential. We identify common causes of complexity and discuss general approaches which can be taken to eliminate them where they are accidental in nature. To make things more concrete we then give an outline for a potential complexity-minimizing approach based on functional programming and Codd's relational model of data. ```

Link: <http://shaffner.us/cs/papers/tarpit.pdf>

[reply](#)

mtreis86 1 day ago [-]

Gödel, Escher, Bach: An Eternal Golden Braid by Douglas Hofstadter is about the intersection of music, math, and computers.

[reply](#)

randcraw 1 day ago [-]

And it's playful. It seems like, 20+ years ago, there were a lot more authors who wrote playful tech books. I miss 'em.

[reply](#)

mtreis86 1 day ago [-]

Any to recommend?

[reply](#)

binarymax 1 day ago [-]

Programming Pearls.

Great short book to get you thinking creatively and how to dissect algorithmic problems, language agnostic with pseudocode examples.

Non-programming but still highly relevant for a professional programmer: Mythical Man Month, and Peopleware.

[reply](#)

eckza 1 day ago [-]

MMM is a must-read, for anyone that wants to effectively engage in the software development lifecycle, IMO. Developer, project manager, doesn't matter.

[reply](#)

ryan-allen 1 day ago [-]

I bought this and lent it to someone shortly after I bought it, I guess I'm gonna have to buy it again!

[reply](#)

smcgrow 1 day ago [-]

"Thinking Recursively" by Eric Roberts. Completely changed the way I think about recursive programming and easy to pick up.

<https://cs.stanford.edu/people/eroberts/books/ThinkingRecurs...>

[reply](#)

jrapdx3 1 day ago [-]

Yes it's a classic and very well written. When I read this more than 25 years ago it was a real eye-opener and made the workings of recursion clear to me. Curiously acquiring an understanding of recursion in programming allows me to "think recursively" in regard to many other matters as well, so important for thinking "outside the box".

I'd venture that besides recursion, the other key concepts in programming have been pointers and first-class functions. Getting a grasp on these 3 ideas has been essential to learning.

[reply](#)

fatjonny 1 day ago [-]

I've been thoroughly enjoying "Designing Data-Intensive Applications" by Martin Kleppmann. It primarily deals with the current state of storing data (databases, etc) starting with storing data on one machine and expanding to distributed architectures...but most importantly it goes over the trade-offs between the various approaches. It is at a high level because of the amount of ground it covers, but it contains a ton of references to dig in deeper if you want to know more about a specific topic.

[reply](#)

silentsea90 1 day ago [-]

+1. Great for getting a lay of the land on distributed systems problems. I haven't ever studied distributed systems formally though, so I wonder how this differs from that.

[reply](#)

MikeTaylor 1 day ago [-]

Definitely, definitely, Kernighan and Plauger's 1976 book *Software Tools*. The code is in RATFOR (a structured dialect of FORTRAN) but all the ideas are language-independent. It remains, four decades on, the best book I have ever read on how to solve the real problems of real program development. Very practical, and covers a vast amount of ground. (As it happens, I am re-reading it right now.)

[reply](#)

pjmorris 1 day ago [-]

+1. I have a corner of a bookshelf reserved for everything Kernighan, e.g. both *Software Tools* editions, both C Language editions, *The Unix Programming Environment*, *The Practice of Programming*, *The AWK Language*, even a copy of *The Elements of Programming Style*. I, and my employers and clients, may owe more to Mr. Kernighan for his teaching and programming style, than anyone else.

[reply](#)

bewuethr 1 day ago [-]

There is also an edition using Pascal, "Software Tools in Pascal" from 1981, which might be a little more accessible than the RATFOR edition. I fully agree with the recommendation as such, though.

[reply](#)

rpeden 1 day ago [-]

The code from "Software Tools in Pascal" mostly just works in FreePascal, too. So it's pretty easy to jump in and work through the examples in the book.

[reply](#)

\_\_bearMountain 1 day ago [-]

Agile Software Development, Principles, Patterns, and Practices - by Uncle Bob Martin

One of the most influential programming books I've ever read. The code is in Java, but it's easy to follow even for a non-Java developer, and the truths are universal. Learn the most fundamental design and encapsulation patterns. Uncle Bob Martin is a legend. This book has probably made me tens of thousands of dollars.

<https://www.amazon.com/Software-Development-Principles-Patte...>

[reply](#)

sgt 1 day ago [-]

## Thinking Forth

<http://thinking-forth.sourceforge.net/>

Teaches you to think simple and elegant.

[reply](#)

protomyth 1 day ago [-]

Thinking Forth is a great suggestion since the approach goes way beyond Forth. I also would read the Brad Cox (objc creator) books if you can get them.

[reply](#)

throwaway7645 1 day ago [-]

Reading this one after Starting Forth. I wonder how many have read this that have never written a line of Forth?

[reply](#)

zimmund 1 day ago [-]

Introduction to algorithms[1] is a great book to improve how you think about code and the way you implement your solutions. Even if you are a seasoned programmer you'll find it useful.

[1]: <https://mitpress.mit.edu/books/introduction-algorithms>

[reply](#)

flor1s 1 day ago [-]

We used it in my algorithms class in uni, I found it a bit heavy on the math side of things. For me a more code oriented book such as Skiena's The Algorithm Design Manual works better.

[reply](#)

dustingetz 1 day ago [-]

Joy of Clojure & SICP. To a lesser extent, Learn You a Haskell. 7 Languages in 7 Weeks is an excellent good baby step book if these are too daunting. 7in7 was my first intro to many new ideas.

Any language worth learning has this property of influencing the way you think forever. TDD, Code Complete & co are all very integrated into mainstream industry and are no longer novel. If you find yourself needing to recommend your colleagues to read Code Complete you might consider working on the skills to get a better job.

[reply](#)

bphogan 1 day ago [-]

Can I plug my book, Exercises for Programmers? <https://pragprog.com/book/bhwb/exercises-for-programmers>

It's a collection of programming exercises I used when I taught introduction to programming. They start out incredibly trivial, ("prompt for a name, print "hello [name]" back to the screen. But the trivial part is, in my opinion, the fun part when you work with a new language.

That program is a two line program in Ruby. But it might be much more complicated if you implemented that as your first GUI app in Swift for iOS.

I wrote the book to teach beginners, but I and others use those exercises to learn new languages. The book has no answers, just the problem statements.

[reply](#)

lcuff 1 day ago [-]

An Introduction to General Systems Thinking. Gerald Weinberg. This book, now over 40 years old, addresses the 'core within the core' of the reality of systems. Unbelievably good, with a very light-hearted tone.

[reply](#)

andyjohnson0 1 day ago [-]

Also *Systemantics* by John Gall.

[reply](#)

swah 1 day ago [-]

Why this was downvoted?

[reply](#)

swah 18 hours ago [-]

Why was this downvoted?

[reply](#)

ctrlp 1 day ago [-]

If you consider C to be language-agnostic, here are some gems. These are personal favorites as much for their excellent writing as for their content.

The Unix Programming Environment was published in 1984. I read it over 20 years later and was astonished at how well it had aged. For a technical book from the 80's, it is amazingly lucid and well-written. It pre-dates *modern* unix, so things have changed but much that goes unstated in newer books (for brevity) is explicit in UPE. (Plus, the history itself is illuminating.) It gave me a much deeper understanding of how programs actually run over computer hardware. Examples in C are old-school and take a bit of close reading but oh so rewarding. <https://www.amazon.com/Unix-Programming-Environment-Prentice...>

Mastering Algorithms in C. Another fantastically well-written book that shows (with practical examples) how to implement common algorithms. This is just such a great book! <https://www.amazon.com/Mastering-Algorithms-Techniques-Sorti...>

Also:

Code (Petzold). This one is truly language-agnostic. Others have mentioned it already. Can't recommend enough if you're iffy on the internals of computers and programming. <https://www.amazon.com/Code-Language-Computer-Hardware-Softw...>

Write Great Code (Volumes I and II). Randall Hyde's books are fantastic explications of the underlying computer operations. Examples are in assembly or pseudo-code but easy to understand. <https://www.amazon.com/Write-Great-Code-Understanding-Machin...>

[reply](#)

drio 1 day ago [-]

+1 to Code (Petzold). I would absolutely start with that. One of my favorite books. The build a computer course from coursera (<https://www.coursera.org/learn/build-a-computer>) is the natural next step after reading code.

[reply](#)

Rannath 1 day ago [-]

Not really programming books, but these have helped me with programming jobs.

-How to make friends and influence people. Anyone who works collaboratively with people needs to be able to communicate effectively.

-The Elements of style. Writing understandable code is similar to any other type of writing.

[reply](#)

Cerium 1 day ago [-]

Elements of Style, also referred to as "Strunk and White" is a gem of a style guide. It can be read every few years to get back on track.

[reply](#)

cestith 1 day ago [-]

There's also The Elements of Programming Style by Kernighan and Pike

[https://en.wikipedia.org/wiki/The\\_Elements\\_of\\_Programming\\_St...](https://en.wikipedia.org/wiki/The_Elements_of_Programming_St...)

[reply](#)

chuckdries 1 day ago [-]

> How to make friends and influence people

I swear I've had this book recommended to me at least a thousand times, I'm almost suspicious though I don't know that suspicious is the right word

[reply](#)

beat 1 day ago [-]

The title seems, well, creepy, in modern language. Like it's manipulative.

It's not.

It was written in the 1930s, and has been continuously in print since then. How many books have achieved that honor? That's because it's one of the best books you will ever read. Absorbing its lessons will make you a better person.

[reply](#)

Swizec 1 day ago [-]

Read it. It's great.

[reply](#)

happy-go-lucky 1 day ago [-]

No one has mentioned *The Little Schemer*.

Edit: Written in a question-answer style, it's geared toward luring you into recursion and functional programming.

[reply](#)

xyjprc 1 day ago [-]

Love the Little Schemer! Lots of interesting and challenging questions!

[reply](#)

agentultra 1 day ago [-]

*Programming in the 1990s* by Edward Cohen. A rather practical introduction to the calculation of programs from their specifications. Plenty of introductions to computer programming involve guessing your program into existence. This is one of those rare books that give a solid, pragmatic approach (with examples) of developing software from solid, mathematically sound specifications and avoiding errors by design.

Even if you don't adopt formal methods in your day-to-day work (often we're not building skyscrapers) it's a useful book to give you insight into the kinds of questions one should be asking and thinking about when designing software systems.

[reply](#)

sateesh 1 day ago [-]

*The Cuckoo's Egg* by Clifford Stoll. This book taught me how important it is to keep a log of events. These logs come in very handy when the problem one trying to debug spawns multiple complex systems.

[reply](#)

hyperdeficit 1 day ago [-]

Seconded. This is also a great read on how to track down a deep problem, going through all of the steps required to figure out where it was coming from. That's on top of it being an entertaining view of early computer networking.

[reply](#)

bandrami 1 day ago [-]

*Let Over Lambda*. Not entirely agnostic, but delves into Forth, Smalltalk, C, Scheme, and Perl while overall being about Lisp. Fascinating book; really a look at metaprogramming (macros) and closures (that's what "let over lambda" is).

[reply](#)

weavie 1 day ago [-]

Whilst being a great book, you are definitely correct in your assertion that this book is not agnostic!

[reply](#)

bandrami 1 day ago [-]

True, but it goes into a lot of different languages, which can amount to the same thing

[reply](#)

Insanity 1 day ago [-]

In addition to those already mentioned here, I enjoyed the book 'Algorithms' by Robert Sedgewick & Kevin Wayne.

The algorithms are explained, and demonstrated (in java). But with the knowledge of how the algorithm works you should be able to use them in another language.



(And even though henrik\_w already mentioned it, Code Complete2 is a really good book to read!)

[reply](#)

cessor 19 hours ago [-]

Good advice! I have an older version, in which the implementations are explained in Pascal. I understand that newer issues cover java. I'd believe the book goes well with his coursera course:

<https://www.coursera.org/learn/algorithms-part1>

[reply](#)

RossBencina 1 day ago [-]

I see you read Kent Beck's TDD book. A good follow-up might be Roy Osherove's "The Art of Unit Testing." I found it to have a lot of pragmatic, practical advice. It's not the final word, but it is a good next step after Kent Beck's book. It has some C#-specific material, but that stuff is interesting to read about even if you're working in other languages.

Let's good suggestions in this thread, here's one I didn't see:

"Software Runaways - lessons learned from massive software project failures," by Robert L. Glass.

[reply](#)

demircancelebi 1 day ago [-]

I have been reading Game Programming Patterns lately. It explains the design patterns with examples from games, and it is really well written by an engineer at Google (Bob Nystrom): <http://gameprogrammingpatterns.com/>

After I complete this book, I think I'll read his other book: Crafting Interpreters. This one teaches about implementing a programming language from scratch, once in Java and a second time in C.

[reply](#)

petra 1 day ago [-]

Sanzy Metz is a great teacher of object-oriented design. Read her ruby book.

"Introduction to algorithms : a creative approach" by Udi Manber. ". Great book to learn algorithm design.

[reply](#)

euske 1 day ago [-]

The Psychology of Computer Programming by Gerald M. Weinberg

It's an old book but the most eye-opening one to me.

[reply](#)

pjmorris 1 day ago [-]

From its preface: "This book has only one major purpose - to trigger the beginning of a new field of study: computer programming as a human activity, or, in short, the psychology of computer programming."

Still a worthy topic, still worth your time. As was mentioned about 'The Pragmatic Programmer', I think reading this works best once you have a few years of experience.

[reply](#)

mcguire 1 day ago [-]

Sure, it was written in the '70s. It talks about punched cards and 24-hour turn around for program listings. On paper.

But programmers haven't changed much since then. You'll recognize all the weirdness in software work.

[reply](#)

masterzachary 1 day ago [-]

\* "Clean Code" Robert C. Martin (978-0132350884)

\* "Refactoring: Improving the Design of Existing Code" Martin Fowler (978-0201485677)

\* "Computer Systems: A Programmers Perspective" Randal E. Bryant (978-0134092669)

[reply](#)

paublyrne 1 day ago [-]

Practical Object-oriented Design in Ruby is a great read with a lot of advice on approaching design problems, approaching refactoring and thinking about how to model. It's in Ruby but I feel a lot of its advice is general.

[reply](#)

beat 1 day ago [-]

aka "Why duck typing is better than inheritance". Yes, great book, but if you're using a language that doesn't support duck typing, some of the concepts will frustrate you and make you wish you were using Ruby instead.

[reply](#)

rimliu 1 day ago [-]

You can do some (most) of the stuff using interfaces/protocols in languages that support these.

[reply](#)

beat 9 hours ago [-]

It's not quite as natural, though. On the other hand, making it explicit is often a Good Thing.

[reply](#)

jacquesm 1 day ago [-]

Any good book on statistics would be a huge asset, as well as a book about debugging strategies.

[reply](#)

a\_bonobo 1 day ago [-]

Statistics books I liked:

Motulsky's Intuitive Biostatistics - this one goes over all the usual methods used in science from distributions to t-test to ANOVA to regressions etc., the basics, but doesn't introduce the maths (you use R for that) but the assumptions and pitfalls of all of those methods.

Statistics Done Wrong: The Woefully Complete Guide - this is all the stuff that's going wrong in applied statistics, a bit short but enlightening

Discovering Statistics Using R - a whopper of a book (~1000 pages?), it goes through *everything* while also being funny (the constant humor may not be for everyone). Graphs, correlations, regressions, all the MLMs and GLMs, linear models etc. pp., their assumptions, how to run them in R, how to interpret R's sometimes annoying output, etc. pp. Like Motulsky's, but wayyyyy more in-depth on the language's specifics.

Naked Statistics - an intro to stats for laypeople with a focus on politics/economics, good for interpreting and assessing daily news

[reply](#)

TeMPORaL 1 day ago [-]

> *as well as a book about debugging strategies*

Any examples of books focused on that? I can't think of any.

[reply](#)

randcraw 1 day ago [-]

<https://news.ycombinator.com/item?id=9242384>

[reply](#)

pjmorris 1 day ago [-]

'The Cartoon Guide to Statistics', Gonick and Smith, is surprisingly substantial for its style and target, an easy way to get started.

[reply](#)

drio 1 day ago [-]

Think stats (<http://shop.oreilly.com/product/0636920020745.do>) is very enjoyable and very practical. You can read it online for free.

[reply](#)

BOOSTERHIDROGEN 1 day ago [-]

Any title ?

[reply](#)

vitomd 1 day ago [-]

If you liked Clean Code, read Clean Coder. A quick summary:

Robert C. Martin introduces the disciplines, techniques, tools, and practices of true software craftsmanship. This book is packed with practical advice—about everything from estimating and coding to refactoring and testing. It covers much more than technique: It is about attitude.

[reply](#)

gonzofish 1 day ago [-]

I feel that books like Clean Coder should be recommended reading for people in their first job. I really feel like it made me think about how I do my job in a more professional, quality-focused manner than even Clean Code did.

[reply](#)

mcguire 1 day ago [-]

<https://maniagnosis.csr.net/2012/03/clean-coder.html>[reply](#)

csneeky 1 day ago [-]

"Types and Programming Languages" (aka "tapl")

[reply](#)

ericssmith 1 day ago [-]

Although the OP seems to be asking for "engineering" not "programming" books, I'm going to second this. Benjamin Pierce's "Types and Programming Languages" will help you get down to what programming is really about. If you are not familiar with lambda calculus and its notation, it may be rough going at first. But lambda calculus is VERY simple, and Pierce takes you through it. The book progresses methodically to the concepts found in most common programming languages.

This is one of the few truly language agnostic books on programming. SICP is close, but it is limited in relevance at times due to the limitations of a particular language (Scheme).

[reply](#)

agumonkey 1 day ago [-]

Also "Software Foundations"

[reply](#)

BFatts 1 day ago [-]

'The Pragmatic Programmer' is a fantastic language-agnostic manual that still applies heavily today.

[reply](#)

IndrekR 1 day ago [-]

"The Elements of Style" by Strunk & White. Not exactly a standard programming book. Not really language-agnostic either -- quite English-centric.

I here assume your source code will be read by others; or by yourself after more than three months has passed.

[reply](#)

SomeHacker44 1 day ago [-]

As someone married to an journalist and now EIC and one who prides himself on concise and solid writing, I really detest this book. It's full of terrible recommendations that the authors themselves don't even follow in their own book. I strongly recommend any English authors stay miles away from it.

If you want this sort of thing, read a well-respected publication's style guide (e.g., from the AP, NYT or Economist).

[reply](#)

Bobbleoxs 1 day ago [-]

It is an English language and writing style classic. Recommend for sure but not sure abt the relevance here.

[reply](#)

capt3m0 1 day ago [-]

Get the new edition (the old 1930 edition is still sold, but is not the one I'd recommend)

[reply](#)

unfocused 1 day ago [-]

"How to Solve It" by George Pólya

[reply](#)

g051051 1 day ago [-]

"Peopleware: Productive Projects and Teams" by DeMarco and Lister.

[reply](#)

OliverJones 1 day ago [-]

An old standard: The Mythical Man Month by Fred Brooks.

[reply](#)

humanrebar 1 day ago [-]

Eh. I read it but a lot of the particulars of how code is made doesn't really apply anymore.

The *concepts* and *terminology* are absolutely invaluable to understand, but you can pick those up other ways.

[reply](#)

sAbakumoff 15 hours ago [-]

"Practices of an Agile Developer: Working in the Real World" - this book was like the Bible for me when I started my career in IT 10 years ago. I re-read it multiple times and I still stick to the practices described in this book. They are language agnostic, they are pretty clear and easy to follow and they can really improve your skills.

[reply](#)

OJFord 1 day ago [-]

Sipser's *Theory of Computation*. It covers automata and languages, computability, and complexity - and is brilliantly written, the proof style in particular: clear 'proof idea's followed by the details that can be easily skipped if you're not interested, or it is clear from the 'idea'.

[reply](#)

fastbeef 1 day ago [-]

Not a programming book per se, but seeing that you had "The Healthy Programmer" in your list I'll throw it out there:

"The underachievers manifesto" - a short book that does wonders for your mental health in a world that values productivity and superficial, short-sighted goals over everything else.

<https://www.amazon.com/Underachievers-Manifesto-Accomplishin...>

[reply](#)

jordigh 1 day ago [-]

It's not language-agnostic, but it's still a great book: The D Programming Language. The reason I recommend it is because Alexandrescu is a great writer who knows a lot about programming languages and the kinds of tradeoffs that a low-level, practical, and safe programming language like D must do.

Even if you never intend to program in D, I encourage you to read this book to get a different view on metaprogramming, memory safety, and concurrency.

[reply](#)

dyarosla 1 day ago [-]

I've really enjoyed "Dependency Injection in .NET"- despite the name, the book itself is really 95% about Dependency Injection and relatively language agnostic. It exhibits a bottom-up approach to using inversion of control in a way that makes sense and is scalable.

<https://www.manning.com/books/dependency-injection-in-dot-ne...>

[reply](#)

stcredzero 1 day ago [-]

Martin Fowler's *Refactoring*

[reply](#)

timclark 1 day ago [-]

Domain Driven Design by Eric Evans

Implementing Domain Driven Design by Vaughn Vernon

Clean Code by Robert Martin

I think you will find some code in all of the books but the ideas are applicable almost everywhere.

[reply](#)

pjmlp 1 day ago [-]

A few ones:

"Algorithms and Data Structures" from Niklaus Wirth.

"Introduction to Algorithms, 3rd Edition" from Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

"The Garbage Collection Handbook: The Art of Automatic Memory Management" from Richard Jones and Antony Hosking

"Code Complete" from Steve McConnell

"From Mathematics to Generic Programming" from Alexander Stepanov and Daniel Rose

[reply](#)

cakeface 1 day ago [-]

I really liked "Building Microservices" by Sam Newman. It's a good review on current software architecture and software development process in addition to going over microservices. Honestly microservices are a topic in the book but it could just be called "Software Architecture in 2016".

[reply](#)

maksa 22 hours ago [-]

- Pragmatic Programmer, From Journeyman To Master

- Code Complete 2nd Ed.

- Quality Software Management Vol 1-4, by Gerald M. Weinberg

I'd also throw in: Code Reading - The Open Source Perspective

[reply](#)

Jeaye 1 day ago [-]

The Reasoned Schemer: <https://mitpress.mit.edu/books/reasoned-schemer>

Learning more and more about imperative programming, OOP, design patterns, etc is good, but branching out into declarative programming and the functional and logic paradigms will stretch your mind for the better.

The great thing, I think, about The Reasoned Schemer is that it tackles a complex topic with almost no prose. The whole book is basically one code example after another, in a Q/A style. "What does this do?" <allow you to think about it> "Here is what it does, and here's why." Rinse and repeat. I think more technical books should try this.

[reply](#)

vram22 1 day ago [-]

The Reasoned Schemer sounds like it was part of a series, is that right? I remember reading about The Little Schemer and maybe The Seasoned Schemer. Are those all parts of the same series? Haven't read any of them yet, but hope to do so some day.

I like reading the classics of the field. Not only because they are classics, but also because they tend to be well-written and hence more readable as well (than your average text). But maybe that is a tautology :) - they are classics because they are well-written ...

Update: Answering my own question - I saw here:

<https://mitpress.mit.edu/books/reasoned-schemer>

that it *is* part of a series, which includes the books I mentioned above.

[reply](#)

mytec 1 day ago [-]

Smalltalk Best Practices by Kent Beck. I feel the advice and experience this book provides goes well beyond Smalltalk.

[reply](#)

neves 1 day ago [-]

Any computer themed book of Gerald M. Weinberg is a must read:

<https://leanpub.com/u/jerryweinberg>

If at first sight you may think that they are outdated and superficial, but you can't be more wrong.

[reply](#)

fd5773 23 hours ago [-]

If you can find a copy, get the 1st Edition of Bertrand Meyer's "Object Oriented Software Construction". (1st ed. is a classic; 2nd ed. is much larger and worse for it, IMO.)

If you can, forget about the title and just read it; most of the good stuff in this book is less about objects and much more a fantastic seminar on programming in general.

[reply](#)

ranko 1 day ago [-]

*Growing Object-Oriented Software Guided by Tests*, by Steve Freeman and Nat Pryce. The examples are, IIRC, in Java, but the ideas about TDD are applicable to any OO language. It'll make you think more about how you write testable code and the tests themselves.

[reply](#)

mathnode 1 day ago [-]

- The Pragmatic Programmer
- The Practice of Programming
- The Mythical Man Month
- The Cathedral and the Bazaar
- The Art of Motorcycle Maintenance
- Introduction to Algorithms
- Hackers and Painters

Some of these do contain mixed language code examples, but they are expressed in a way to be agnostic. A problem is a problem; in any language.

[reply](#)

optimuspaul 1 day ago [-]

The Art of Motorcycle Maintenance?

[reply](#)

inerg 1 day ago [-]

I believe he was talking about Zen and the Art of Motorcycle Maintenance: An Inquiry Into Values (<https://www.amazon.ca/Zen-Art-Motorcycle-Maintenance-Inquiry...>).

It's a nice fiction book that goes into how different people view the world. At least that's what I've gotten out of it so far, I'm only about a quarter of the way through it so I might be missing some of the things it covers.

[reply](#)

optimuspaul 1 day ago [-]

superpope99 1 day ago [-]

yes, I assumed that to be the book they meant. I read it 25 years ago. I am just surprised that it would be included in this list. I see no connection to the question posed.

[reply](#)

inerg 1 day ago [-]

I'm not sure why he included it in his list. However I do think that it does a very good job of bringing to light that your customers/users may not be appreciative of buggy code. They want something that "just works" and to not have to deal with vague error messages.

I started reading the book about a year into my first development job and it really brought to light the frustrations my users were having as I was seeing as it was just another interesting problem to me. As I said in my past post I haven't been diligent in reading it so I'm only a quarter of the way through but here's hoping I'll get more insights out of it when I pick it back up.

I do think that what this book talks about can be discovered in other ways, but for people who are just starting in the technology field it's a good primer to be aware of how others experience what you implement.

[reply](#)

randcraw 1 day ago [-]

In a way, the entire book is about debugging, but the target is the author's thought process rather than code. While Robert Pirsig fictionalizes the story, in fact it's closely based on his own life, in which he sought understanding of values (via zen and academics) but somehow went wrong and ended up catatonic and in shock therapy. The plot is a revisit of the mind trek he was on then and a quest to see where he went wrong. Pirsig approaches this not as a psychologist, but as a reductionist philosopher/scientist, trying to identify *what* troubled him and caused him to fail -- a debugging of the mind.

[reply](#)

dreeri 1 day ago [-]

Zen and the Art of Motorcycle Maintenance is about how people see quality.

[reply](#)

LyndsySimon 1 day ago [-]

<https://www.amazon.com/dp/0060589469/>

IMO, It's more "philosophy" than "programming", but it makes you think about the role of technology in our lives and our role as creators of that technology.

[reply](#)

superpope99 1 day ago [-]

They mean 'Zen and the Art of Motorcycle Maintenance' by the late Robert M. Pirsig - I second the recommendation

[reply](#)

cpr 1 day ago [-]

Zen and the Art...

[reply](#)

ryan-allen 1 day ago [-]

I found this book very difficult to read, and I didn't understand it's significance :(

[reply](#)

l0stkn0wledge 1 day ago [-]

If you are writing code, you are doing yourself (and anyone using your code) a disservice if you do not read something on secure coding. There are not a ton of code agnostic resouces,

but you may want to start with "Software Security: Building Security In"

I would then look for language specific options as well, because programming for security can vary a lot amongst languages. Writing securely for native applications running on a system is much different than writing secure web apps.

[reply](#)

tmaly 1 day ago [-]

Understanding the Four Rules of Simple Design by Cory Haines

I like the 4 simple rules, I think he originally got these from Kent Beck. It is easier to keep this system in your mind if it is just a few basic principles.

Working Effectively with Legacy Code by Michael Feathers

As others also mentioned this. I think this is becoming more important as people transition to new jobs where they have to take on existing software. Having a process to deal with code that lacks documentation and tests is really important.

[reply](#)

svec 1 day ago [-]

"Programmers at Work: Interviews With 19 Programmers Who Shaped the Computer Industry" by Susan Lammers.

<https://www.amazon.com/Programmers-Work-Interviews-Computer-...>

This book is from 1989, but it's a timeless and fascinating look at the minds of Bill Gates, Andy Hertzfeld (apple/mac), Dan Bricklin (visicalc), and 16 others.

[reply](#)

rajadigopula 1 day ago [-]

A short book (25 pages), but I find it worth mentioning here - 6 Things About Programming That Every Computer Programmer Should Know - <https://www.amazon.co.uk/Things-Programming-Computer-Program...>

A really well written short glimpse of things every programmer must know.

[reply](#)

ishmaelahmed 1 day ago [-]

"The Pragmatic Programmer" by Andy Hunt and Dave Thomas

[reply](#)

More

[Guidelines](#) | [FAQ](#) | [Support](#) | [API](#) | [Security](#) | [Lists](#) | [Bookmarklet](#) | [DMCA](#) | [Apply to YC](#) | [Contact](#)

Search: