


12 Most Influential Books Every Software Engineer Needs to Read

 jasonroell.com/2015/03/16/12-most-influential-books-every-software-engineer-needs-to-read

March 16, 2015

This is a question that I get a lot, especially from co-workers or friends that are just beginning their journey as a software craftsman.

| What book should I read to become a better developer? Do I need to read books?

I think it's a great question, and it is one that I asked many of my mentors as I was becoming a software engineer. The problem was that many people suggested different books on different topics. All the books they suggested were great in their own right, but no one was able to give me a list that would be the **ESSENTIAL** books, the **MUST READS**, that any engineer with hopes of being *great* should most certainly read.

Well, I've learned a lot from my mentors and realized that I still had a lot to learn with the many different books that were suggested to me. I decided to develop a routine to read one book a month in my profession field (software engineering). Over the years, I've aggregated a list that, I believe, to be **MUST READS** for anyone that wants to be a top tier developer.

Now let me state the obvious – just reading all of these books on the list will not make you a great developer. That will come with years of experience and applying the principles in these books into real practices and developing your problem-solving skills in the real world.

However, reading these books will help you avoid the major pitfalls and mistakes that many developers make early on in their careers. I wish that someone would have told me about these books just starting out, but I was lucky enough to have found and read them over the years. You might have read some of these books in college for your computer science or engineering classes. Maybe at the time, you didn't think they were important, but I can say first hand that I've used and applied many principles from **each and every one** of these books.

Let me also point out that this is not an exhaustive list. Many great books come out every year. These are just the ones that have had the biggest impact on myself and my career. Also, these are mostly language agnostic, and can be applied using any of the many software languages. (I will do another post with the best books targeted at certain technology platforms and stacks)

Well, let's get to it then! (drum roll, please)

THE LIST

(All these are essential, but I put them in descending order from which ones had the biggest impact on me. I have also provided the link (click on the book cover) to where you can purchase the book on Amazon if interested. Read the reviews and decide for yourself!)

12. Working Effectively with Legacy Code

I love this book because almost every software developer, at some point in their career, has to support and work with a legacy system. In this book, Michael Feathers offers start-to-finish strategies for working more effectively with large, untested legacy code bases. This book draws on material Michael created for his renowned Object Mentor seminars: techniques Michael has used in mentoring to help hundreds of developers, technical managers, and testers bring their legacy systems under control.

11. The Mythical Man-Month

This book is a classic, but recently revised and corrected. The amazing thing is how relevant the book still is to software product development. If you are involved in software, this book is a must-read. The most valuable part of the book, I believe, is the "plan to throw out" prototype chapter. While the goal is always to make a bigger, better, fast whatever, it is almost an axiom that you **WILL** build something that has to be discarded and reworked. This happens every time, I can tell you from first-hand experience. Therefore, it is vital to plan to throw out so you can migrate your users to whatever will follow. If you dream that the first product is **THE ONE**, you risk abandoning them on a product that will inevitably evolve. Planning to throw-away also helps meet the schedule goals by setting reasonable milestones that can be obtained.

10. Design Patterns

If you are planning to be an architect or designer of a system, you will most likely be required to read this book. Hailed as one of the greatest software development books ever written, this book goes into great detail on the many different design patterns that have been developed over the years to help software engineers avoid and handle common problems that the industry faces. Following the strategies in this book will allow you to build higher quality, flexible, and maintainable software. This book also goes by the name "Gang of Four" in software groups because of its famous four authors that put this book together.

9. Programming Pearls (2nd Edition)

This book is slightly different from the other books on the list. I would say this book helps a person "think like a programmer". Programming Pearls is a compendium of 15 columns previously published in Communications of the ACM. The columns cover a broad range of topics related to programming: from requirements gathering to performance tuning. The focus is primarily on coding techniques and algorithms.

Each column has been reorganized as a chapter. Chapters usually start with the presentation of a practical problem. Then various solutions are presented and are used as lessons to be learned. The writing style is clear and fun.

Programming Pearls is not a usual book teaching new programming concepts. Although it contains good and sometimes quite novel ideas, the aim of the book is not to teach something new but to help you become a better problem solver.

8. CODE: The Hidden Language of Computer Hardware and Software

This book cleared up a lot of the “Magic” that goes into creating and developing complex systems. There are so many abstractions these days that the low-level details are sometimes unknown to the developer. Though you may not find yourself using this book 24/7 in practice...I believe it is a good idea to have an understanding of what you are building on top of and how the whole orchestration works. It may come in handy when you need to open up that “Black Box” and deep dive into the software or hardware to fix a pesky bug. “CODE: The Hidden Language of Computer Hardware and Software” by Charles Petzold deals with a number of programming concepts starting from number systems – decimal, octal, binary to high-level languages. The book explains packet based communication protocols and TCP. Many chapters are about hardware concepts, and five chapters are devoted to software and teach about the operating system, floating point arithmetic, and GUIs.

7. The Art of Computer Programming

This is another classic. This was written by the famous computer scientist Professor Donald Knuth and is highly praised by many of the top programmers in the industry. Even Bill Gates is quoted saying

“If you think you’re a really good programmer... read [Knuth’s] Art of Computer Programming... You should definitely send me a resume if you can read the whole thing.”

The book begins with basic programming concepts and techniques, then focuses more particularly on information structures—the representation of information inside a computer, the structural relationships between data elements and how to deal with them efficiently. Elementary applications are given to simulation, numerical methods, symbolic computing, software, and system design.

6. Refactoring

“Refactoring” by Martin Fowler is about improving the design of existing code. It is the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure. With refactoring, you can even take a bad design and rework it into a good one. This book offers a thorough discussion of the principles of refactoring, including where to spot opportunities for refactoring, and how to set up the required tests. There is also a catalog of more than 40 proven refactorings with details as to when and why to use the refactoring, step by step instructions for implementing it, and an example illustrating how it works. The book is written using Java as its principal language, but the ideas apply to any OO language.

5. Clean Code

“Clean Code,” written by Robert C. Martin, is divided into three parts. The first describes the principles, patterns, and practices of writing clean code. The second part consists of several case studies of increasing complexity. Each case study is an exercise in cleaning up code—of transforming a code base that has some problems into one that is sound and efficient. The third part is the payoff: a single chapter containing a list of heuristics and “smells” gathered while creating the case studies. The result is a knowledge base that describes the way we think when we write, read, and clean code.

4. Introduction to Algorithms

This has to be the single best book for understanding and using algorithms (which you will be doing a lot of in software development). Some books on algorithms are rigorous but incomplete; others cover masses of material but lack rigor. Introduction to Algorithms uniquely combines rigor and comprehensiveness. The book covers a broad range of algorithms in depth, yet makes their design and analysis accessible to all levels of readers. Each chapter is relatively self-contained and can be used as a unit of study. The algorithms are described in English and in a pseudocode designed to be readable by anyone who has done a little programming. The explanations have been kept elementary without sacrificing depth of coverage or mathematical rigor. The first edition became a widely used text in universities worldwide as well as the standard reference for professionals. The second edition featured new chapters on the role of algorithms, probabilistic analysis, and randomized algorithms, and linear programming.

3. Structure and Interpretation of Computer Programs

With an analytical and rigorous approach to problem solving and programming techniques, this book is oriented toward engineering. *Structure and Interpretation of Computer Programs* emphasizes the central role played by different approaches to dealing with time in computational models. Its unique approach makes it appropriate for an introduction to computer science courses, as well as programming languages and program design. The book further explains the four best-known paradigms of programming languages – imperative, object-oriented, logic based and applicative programming.

2. Pragmatic Programmer

This was one of the first programming books I read. I had a friend recommend it to me in my first professional job. I’m glad he did. Though the book was written in 1999 (I believe), the concepts are the basis of how we go about developing a complex system in a practical manner. Programmers are craftspeople trained to use a certain set of tools (editors, object managers, version trackers) to generate a certain kind of product (programs) that will operate in some environment (operating systems on hardware assemblies). Like any other craft, computer programming has spawned a body of wisdom, most of which isn’t taught at universities or in certification classes. Most

programmers arrive at the so-called tricks of the trade over time, through independent experimentation. In *The Pragmatic Programmer*, Andrew Hunt and David Thomas codify many of the truths they've discovered during their respective careers as designers of software and writers of code.

Some of the authors' nuggets of pragmatism are concrete, and the path to their implementation is clear. They advise readers to learn one text editor, for example, and use it for everything. They also recommend the use of version-tracking software for even the smallest projects and promote the merits of learning regular expression syntax and a text-manipulation language. Other (perhaps more valuable) advice in the book is more light-hearted. In the debugging section, it is noted that "if you see hoof prints think horses, not zebras." That is, suspect everything, but start looking for problems in the most prominent places. There are recommendations for making estimates of time and expense, and for integrating testing into the development process. You'll want a copy of *The Pragmatic Programmer* for two reasons: it displays your own accumulated wisdom more clearly than you ever bothered to state it, and it introduces you to methods of work that you may not yet have considered.

1. Code Complete 2

And this is it! The number one book (IMHO) to read if you are going to be a great software engineer. Widely considered one of the best practical guides to programming, Steve McConnell's original CODE COMPLETE has been helping developers write better software for more than a decade. Now this classic book has been fully updated and revised with leading-edge practices—and hundreds of new code samples—illustrating the art and science of software construction. Capturing the body of knowledge available from research, academia, and everyday commercial practice, McConnell synthesizes the most effective techniques and must-know principles into clear, pragmatic guidance. No matter what your experience level, development environment, or project size, this book will inform and stimulate your thinking—and help you build the highest quality code.

Discover the timeless techniques and strategies that help you:

- Design for minimum complexity and maximum creativity
- Reap the benefits of collaborative development
- Apply defensive programming techniques to reduce and flush out errors
- Exploit opportunities to refactor—or evolve—code, and do it safely
- Use construction practices that are right-weight for your project
- Debug problems quickly and effectively
- Resolve critical construction issues early and correctly
- Build quality into the beginning, middle, and end of your project

Well, that's it for now!

Let me know in the comments if you have read any of these or have any other must-reads for software developers!

If you have enjoyed this post, the biggest compliment you could give would be to ***share this with someone that you think would enjoy it!***

Additionally, if you never want to miss a post, subscribe to this blog by clicking the follow button in the bottom right corner! Thanks for reading, have a great day, and never stop learning!