THE COMPLETE SOFTWARE DEVELOPER'S CAREER GUIDE

HOW TO LEARN YOUR NEXT PROGRAMMING LANGUAGE, ACE YOUR PROGRAMMING INTERVIEW, AND LAND THE CODING JOB OF YOUR DREAMS

JOHN SONMEZ

# CONTENTS

Preface .................................................................................................................. 7
Is This Book For Me? .......................................................................................... 10
Chapter 1 – How to Use This Book .................................................................. 13

## SECTION 1

**GETTING STARTED AS A SOFTWARE DEVELOPER** 19

Chapter 2 – How to Get Started ....................................................................... 22
Chapter 3 – The Technical Skills You Need to Have ...................................... 34
Chapter 4 – How to Develop Technical Skills ............................................... 49
Chapter 5 – What Programming Language Should I Learn ......................... 60
Chapter 6 – Learning Your First Programming Language ......................... 70
Chapter 7 – Going to College ......................................................................... 83
Chapter 8 – Coding Boot Camps ..................................................................... 96
Chapter 9 – Teaching Yourself ....................................................................... 109

## SECTION 2

**GETTING A JOB** 123

Chapter 10 – Internships ................................................................................. 125
Chapter 11 – Getting a Job Without Experience ........................................... 135
Chapter 12 – How to Find a Job ..................................................................... 150
Chapter 13 – Creating a Resume ..................................................................... 168
Chapter 14 – The Interview Process ............................................................... 184
Chapter 15 – Salaries and Negotiation ............................................................ 203
Chapter 16 – How to Leave a Job .................................................................. 220
Chapter 17 – Switching Mid-Career (Late Entry) ........................................... 232
Chapter 18 – Going From QA or Another Technical Role to Development .... 238
Chapter 19 – Contracting vs. Salary ................................................................. 247
Chapter 20 – How the Recruiting Industry Works ......................................... 259
SECTION 5
ADVANCING YOUR CAREER

Chapter 48 – Creating a Reputation .............................................................. 613
Chapter 49 – Networking and Groups .......................................................... 626
Chapter 50 – Keeping Your Skills Up to Date .............................................. 636
Chapter 51 – Generalist vs. Specialist .......................................................... 646
Chapter 52 – Speaking and Conferences ..................................................... 655
Chapter 53 – Creating a Blog ....................................................................... 672
Chapter 54 – Freelancing and Starting a Business ....................................... 691
Chapter 55 – Career Paths .......................................................................... 712
Chapter 56 – Job Stability and Security ....................................................... 725
Chapter 57 – Training and Certifications .................................................... 737
Chapter 58 – Side Projects ......................................................................... 751
Chapter 59 – Best Books to Read ................................................................. 766
Chapter 60 – Parting Words ....................................................................... 780
IS THIS BOOK FOR ME?

There you were, minding your own business when this book’s awesome cover grabbed your attention and you just had to pick it up (or click on it.)

Now you are flipping through the pages asking yourself, “is this book for me?”

Don’t worry, I can assure you I wrote this book just for you, even though I don’t know anything about you.

How? You ask. After all, I don’t even know that you can read.

Well, actually I do know you can read, because if you couldn’t, my words wouldn’t be magically vocalizing in your mind at this very moment.

I also know one other thing about you, which is that we share a keen sense of humor—or at the very least you can tolerate my humor—since you are still reading thus far.

Well, before I lose your interest permanently and you put this book back down on the shelf or click that other window in your browser, I suppose I should get a bit serious and get down to business.

All joking aside, no matter where you are in your software development career, this book has something to offer you.

For the scanners among us, allow me to quickly categorize you into one of three categories and you can read the category that is most relevant to you:
BEGINNER OR JUST INTERESTED IN LEARNING SOFTWARE DEVELOPMENT

If you are starting out and trying to learn software development / programming or you’ve learned a little, but haven’t gotten your first job a software developer yet, you’ll find the largest amount of value in the first two sections of this book where I talk about how to get started as a software developer and how to get your first job.

The rest of the book you’ll find useful to fill in any knowledge gaps you’ll need to succeed as a software developer, thrive in the workplace and advance your career.

You’ll find relevant topics in this book that you won’t find in any other book on software development (that I’m aware of) which will help you get rid of all the confusion with how to actually get your start, learn your first programming language and navigate the murky waters of college versus coding boot camps versus self-taught education.

MID-CAREER DEVELOPER

You’ll likely find the most value in section three, “What you Need to Know About Software Development.” This section fills in any knowledge gaps you may have and will help you to actively manage your career to succeed in the workplace.

But that doesn’t mean you won’t get any value out of the first section of the book, because even though you may already know how to program, you’ll probably find it useful to learn how to further develop and pick up new technical skills, learn new programming languages, apply for jobs, craft a resume, and negotiate your salary.

And if you are interested in advancing your career—which you should be—you’ll find the last section of this book, “Advancing Your Career,” useful as well.

EXPERIENCED PROFESSIONAL

I know. I know you know all this stuff.
You don’t need a basic book from some “hot-shot” on how to get started as a software developer. You don’t need to learn what source control is or get insight on whether you should go to college or boot camp.

I get it—really, I do.

But, trust me on this one, this book is still for you. Here’s why.

First of all, about half of this book is dedicated to working as a developer and advancing your career.

Even though you’ve been around for awhile and have likely been very successful—congrats by the way—you will probably find some benefit in learning how to better handle your coworkers and boss, sell your ideas, handle leadership and, heck, even get a raise or a promotion.

If you haven’t already, you’ll probably find yourself up against this “glass ceiling” in your software development career, where you don’t feel like you can make much upward progress.

Been there, done that, got the t-shirt.

Fortunately, I also bashed my head right through that glass ceiling and I’ll show you how to do it by telling you how to build a personal brand, speak at conferences, start a side-project and a whole lot more.

Finally, even though the beginning sections might seem a bit basic, you’ll still likely find some value in the information about learning technologies, getting a high-paying job, negotiating your salary and choosing between contract and salaried employment.

Plus, you mentor other developers don’t you?

Would be nice if you had some good advice written down to give them about starting out, wouldn’t it?

So, yes. Yes, I say again, this book is for you, whoever you are.

I’ll even venture so far out and be so bold as to say that even if you don’t have the slightest interest in software development, you’ll still probably get a bunch out of this book, because even though this book is specialized to software developers, it’s really about managing your career and being as successful in it as possible.

And if you’ve made it this far, then this book is really, really for you, because obviously you like me—and you know what, I like you too. :)}
Now that I’ve given you a nice long list of technical skills to develop, you might be wondering how you are going to develop all of those skills and how long it is going to take you.

Well, as for the length of time—don’t worry—you’ll be developing your technical skills as long as you are a software developer. **Think of it as a journey, not a destination.**

You will always be able to get better—if you choose to.

I’ve spent plenty of time developing my technical skills the wrong way.

However, in my three years of creating over 50 highly-technical developer training courses on Pluralsight, I’ve also learned how to develop technical skills at a lightning fast speed while teaching others at the same time.

**I used to think the best way to learn a technical skill was to take a big reference book and read it cover-to-cover.**

Back then, I read too many 800+ page books to count and didn’t benefit much from the exercise; although my arms might have grown from carrying around books of that size.

I don’t want you to make the same mistakes I did, and if you already have, I **want to show you a better way.**
LEARNING HOW TO LEARN QUICKLY

Before we get into the specifics about learning technical skills, I think it’s worth taking a second to talk about learning anything quickly and teaching yourself in general.

We’ll go much more in-depth about the topic of teaching yourself in an upcoming chapter in this section, but I want to go over the basics here and talk about a methodology I use to learn anything quickly.

As I mentioned, I spent a large amount of time both learning and teaching various technologies. I learned whole programming languages in a matter of weeks and then turned around and taught courses on them.

During that process, I developed a reliable system for learning just about anything I needed to learn.

This wasn’t so much a conscious effort as it was a necessity. I was trying to learn at such a rapid rate that I had to come up with efficient ways of doing things, and naturally, patterns of learning developed which helped me to become faster and faster.

I’m just going to cover the basics here, since you can find a whole course I put together on the subject at “10 Steps to Learn Anything Quickly” or in a few chapters in my Soft Skills book.

THE BASIC PROCESS

The basic idea is pretty simple.

Essentially, you want to first get a good idea of what you are learning and what the scope of it is.

You need to get enough information about your subject to understand the big picture and narrow the subject down to a small enough scope that you can actually tackle it and wrap your head around it in a realistic amount of time.

Then, you need a goal. You need to establish what it is you are trying to learn and why and, most importantly, what metric you will use to know that you’ve learned it.
Far too many people set out to learn something but have no way to measure whether they have succeeded or not.

Equipped with that starting point, you can start to **gather some resources for learning**.

I recommend not just reading one book cover-to-cover but to instead gather multiple resources, which may include books, blogs, podcasts, magazines, video courses and tutorials, expert opinions, etc.

Then, you are going to use some of those resources to **create an actual plan for learning**.

You can do this by using the resources you’ve gathered to create a systematic, sequential set of steps for you to learn what you want to learn.

For example, you could utilize the table of contents of one of the books you found to help figure out what order you should try to learn things in and what is important.

You are basically going to figure out in what order to learn everything you need to know about your topic.

**Then, you dive in.** From your learning plan, start with each module you are going to learn about your subject. For each module, **learn enough to get started, play around** for a bit, and then go back and **answer any questions you had** while playing around.

You are basically going to focus on **learning by doing**, which we’ll talk about in a second.

The key here is to not learn too much up front. Instead, utilize natural curiosity to drive your learning as you play around on your own. Then, go back and actually read the text or consume the content about your topic, with questions in your head and some experience which will **naturally guide you to seek out what is actually important**.

A big problem we face when learning by consuming a bunch of material is that we don’t actually know what is important. By playing around first and forming your own questions, you solve that problem and what you learn actually sticks.

Finally, you **take what you learned and you teach it to someone else**.

It doesn’t really matter the format and it doesn’t matter who you teach it to. You could talk to your dog or the squirrels in your yard if you like. Doesn’t really matter.
What matters is that you somehow reorganize the thoughts in your head in a way that communicates them to the outside world.

This is the place where learning changes from knowledge to understanding.

And that’s it.

What we have here is a basic formula you can apply to just about anything you want to learn quickly.

If you’d like to see a more detailed example, complete with a workbook and videos that can help you master this process, you can find it here: “10 Steps to Learn Anything Quickly”.

Now, let’s talk more specifically about learning and developing technical skills.

**LEARN BY DOING**

I believe we all learn best by doing, but when it comes to technical skills, this is paramount.

It is just not possible to learn most technical skills by simply reading a book or even watching a video tutorial.

You may get an idea of what is possible using a particular technology, programming language, or tool, but until you’ve actually used it yourself, or solved problems with it, you are only going to have a surface level understanding.

Just about everything I talked about in the last chapter is going to require more than book knowledge in order to gain any real level of competency.

This might be obvious for programming languages, but can you really learn how to use source control from just reading about the syntax?

If you’ve never made the mistake of merging a file into the wrong branch or checking out the wrong version of the source code, and you’ve never actually used a version history to figure out where a bug got introduced, you aren’t really going to know how to use source control—you’ll just think you do.

(Don’t worry if you don’t understand any of that yet.)
But didn’t I promise to teach you all of those technical skills in a later section of this book?

Aren’t you reading a book right now, hoping to learn something?

Yes, but the key is that the learning doesn’t stop there.

You can read my words and gain a cursory understanding of a topic I’m talking about here, but at some point you are going to need to put down this book and actually take some real action to actively learn what you are reading about by doing (at least for the technical skills we are talking about here).

**HOW TO LEARN BY DOING**

At the risk of regurgitating some information that might seem obvious, I’m going to tell you about how to actually learn by doing—consider this as a reminder for something you already know.

Whenever you are going to try and learn a technical skill, start by figuring out what it is going to help you do.

If you don’t have an immediate need for the skill, you might even question whether you need to learn it at all. A large amount of time is wasted by learning technical skills we are never actually going to use in the real world. Believe me, I’m so guilty of doing this that it’s not even funny.

You’ll have a much easier time learning something if you have an immediate application for it—a real reason to learn it.

I guarantee you will be learning like you’ve never learned before if you are being instructed on how to skydive right before you fly up into the sky and jump from a plane.

But what if you don’t have a pressing need? What if you are learning a technical skill because you want to be able to get a job where you’ll need to use it?

In that case you need to manufacture a reason to use that skill. Create a goal.
AN EXAMPLE OF LEARNING BY DOING

Let’s look at a real example.

Suppose you wanted to learn about relational databases and how to use them.

You could just try and read about a database and run some queries against it to play around with—and that might be somewhat effective.

What if instead your goal involved creating a database to store a collection of movies you owned?

What if your goal was to query this database, insert new movies, delete movies, update the titles, etc?

What if you wanted to create a simple application to let you access the database and do all this?

Now, you have a purpose and a way to learn by doing.

Now, you have something to do.

How do you approach learning about relational databases?

You crack open that book or you watch that video tutorial, looking for specific information you need to know to solve your actual problems.

Then, you actually create and use a database and not just as an exercise. You have a real goal.

Think about how much more information you’ll retain when you work and learn in this way.

And won’t it also be so much more fun?

HOW I TEACH TECHNICAL SKILLS

As I mentioned earlier, I have taught quite a few technical skills in a pretty large variety of technologies in my day.

Therefore, I thought you would benefit from seeing how I teach technical skills to make them easier to learn. That way, you can just apply this concept when teaching them to yourself.
Make sense?

When I teach technical skills, I want to give people the biggest bang for their buck, and I don’t want to bore them with a bunch of stuff they don’t really need to know or could learn on their own when they actually need to learn it.

Instead, I focus on teaching what will be immediately valuable and giving students the resources they need to practice what I call “just in time learning” when they need to go deeper on a topic.

There are three main things I try to teach someone when I am teaching a technical skill:

- The big picture: what can you do with the technology?
- How to get started.
- The 20% you need to know to be the most effective.

Let’s break each of these things down.

**THE BIG PICTURE: WHAT CAN YOU DO WITH THE TECHNOLOGY?**

I always start with the big picture.

I believe in the power of Google to solve most of your problems, but you can’t Google something if you don’t know what it is.

Therefore, I first try to teach my students how big a particular technology is and an overview of what it can do.

This is at a very surface level. I’m not showing them or you how to do everything in a technology; I’m just giving you a quick tour and overview of all the points of interest on a map.

For a programming language, for instance, I might talk a bit about the history of the language and what it is mostly used for.

Then, I might jump in and show you all the constructs of the language and the language features—especially the unique ones.
Finally, I might introduce you to the various libraries that are a standard part of the language and give you an idea of what you can do with them and what they cover.

The idea here is to give you the complete lay of the land without going into the details.

You can always look up the details on your own for the things you are interested in.

At this step, I want to eliminate the unknown unknowns. I want to make sure that you establish what you don’t have any knowledge about, so when you need to learn about it, you can know where to look.

My goal is that you don’t say, “Oh, I didn’t know X could do that,” but instead say, “I know X can do it. I’m not sure how, but I can figure that out later.”

Imagine trying to learn woodworking without knowing that such a thing as a dremel or router existed.

You don’t have to know how to use those tools, but without knowing they exist, you would be severely handicapped.

**HOW TO GET STARTED**

The next thing I like to teach students is how to get started.

This is often the most difficult part of learning a technology—and it’s the precursor to “doing”—so I try to make this as painless as possible.

I want to show a student how to download whatever they need to download, get it installed, create their first project, and compile their code.

Once a person is able to overcome this obstacle, they can start playing around and actually build things or work with the technology.

If this barrier to entry seems too high, someone is likely to just read a book or watch a tutorial and never actually get their hands dirty.

You can utilize this in your own studies by making sure you focus on discovering how to get started with a particular technical skill early on in your learning process.
Look for specific tutorials or guides that will show you how to get started, then you can take it from there.

**YOU NEED TO KNOW 20% TO BE THE MOST EFFECTIVE**

Finally, I try to teach students 20% of the information concerning the technology that they’ll use 80% or more of the time.

Almost everything in life falls into what is called the Pareto principle, which basically states that 20% of something produces 80% of the results.

The key to learning a technical skill is figuring out what the 20% is.

What 20% can you learn that will be used in 80% of the work you do using that technical skill?

This is where it is really going to be critical to be doing rather than just reading.

Many books and even tutorials are written like reference manuals, not giving particular emphasis to the 20% of the technology that is most important to know.

If you are actively working with a technology, you’ll quickly discover what you use the most because it will be extremely painful to not know it.

Let’s look at the relational database example again.

Most likely, if you learn about relational databases, you’ll find that writing select statements is pretty squarely in that 20% area.

If you just read a book on SQL, you might find equal weight given to selecting, inserting, updating, deleting, indexing, and various other database functions.

However, if you actually try and create a database and use it, you are going to be doing a whole lot of select statements. You’ll also quickly realize that you need to learn how to join tables.

Instead of wasting your time trying to learn everything there is to know about relational databases, you’ll focus your efforts on learning how to write select statements, join tables, and other common operations that make up that critical 20%.

This is why doing is so important.
It also can help to shadow an expert and watch them work, or even assist them in an apprentice-like situation.

By just seeing someone who uses the technical skill you are developing and what their 20% is, you can learn what you need to know quickly.

On the job training especially can be extremely effective.

**READ WHAT EXPERTS ARE WRITING**

I’ll leave you with one final idea to help you develop technical skills.

**Become an avid reader of what experts who already possess that technical skill are writing.**

When I was learning my craft, I would spend about 30 minutes each day reading various blogs related to subjects I was learning about.

When I really wanted to get an in-depth knowledge of C++, I devoured Scott Meyers’ *Effective C++* books.

Oftentimes, just hearing an expert’s opinion on a subject can grant you deep insights you wouldn’t be able to gather on your own.

It’s one thing to understand the syntax of a programming language or how to use a framework; it is another thing to understand the idiomatic usage of either.

Study how experts are applying the skills in the real world that you are trying to learn.

Read about the problems and arguments experts have regarding the intricacies of a technical skill, and your understanding will deepen.

**PRACTICE, PRACTICE, PRACTICE**

Hopefully, I’ve given you a few good tips to learn technical skills and to develop them further.

It should be pretty clear to you now that learning by doing is critical—especially when dealing with technical skills.
It should also be pretty clear to you that you need to have an actual plan to learn and a clear goal of what it is exactly that you want to learn.

I want to leave you with one final point.

**Practice.**

It’s going to take time to develop any technical skill. In order to get good at something, you are going to have to practice a lot.

Try not to get frustrated by how long it may seem to take, especially when you feel like you aren’t making progress.

If you put in the time, the skills will come as long as you are following a solid plan with a clear goal.

Just keep at it and trust the process.