WHAT DOES A CAREER IN CODING LOOK LIKE?

By Jay Wengrow

What Does a Career in Coding Look Like?

You've probably heard that coding makes for a great career. It's regularly featured in the media as one of the best jobs out there, such as in CNN, U.S. News, and Glassdoor. You've heard that coders make great money, have flexible work options, and have an extremely low unemployment rate. And from what you've seen in TV and movies, coding looks like a lot of fun and seems to have lots of interesting perks, like scooters and nap pods.

But let's dig into the facts. Here are some of the key questions we'll address in this book:

- What does a coder actually do at work every day?
- What skills and attributes does a coder need to be successful?
- What kind of companies do coders work for?
- What does the career trajectory of a coder look like?
- Is coding a career for you?

So, let's get started!

ACTUALIZE

What Do Coders Actually Do?

Coders go by many names. Software Engineers, Software Developers, and Computer Programmers are three of the most common terms used to describe professional coders, and all titles basically refer to the same position, as there's no substantive difference between the three. The term programmer is slightly dated, so you'll see more mention of software engineers and developers these days. In this book, we'll use all of these terms interchangeably.

In a sentence, software developers are the people who create software. Now, software includes many things. Desktop applications, mobile apps, and websites are the most popularly known types of software. However, software exists in many other places too - such as in your car! These days, cars (even the ones that aren't self-driving) have lots of software embedded within them that tells the various parts of the car how to respond to different situations. It's software developers who create all of this software. To the user, computer programs and cars just seem to work - but it's the coders in the background who actually bring them to life.

Even websites are a kind of software. When you visit an online store

like Amazon - a lot of programming has been done to make it that you can add a product to your shopping cart and then purchase it. One can't simply create an image of a shopping cart, add it to a web page, and automatically be able to accept orders. The code behind the website dictates all the nitty gritty details - how someone selects an item, how they add it to the cart, how payments are processed, how to notify the store that they need to ship a new order, and many other details. These processes appear seamless to the user (on a good website, anyway), but there's actually a lot of complexity under the hood!

To make software, developers write computer code - which we'll just call code. To understand this in greater detail, we need to understand a little bit about how computers work. Despite recent movies that make computers and robots seem to have minds of their own - they don't. Computers are just pieces of metal connected to other pieces of metal. They don't understand English. And they don't understand Spanish or Mandarin either. What they do understand is binary - which is just a bunch of ones and zeroes.

In fact, even simple text like "HELLO THERE!" is really 01001000 01000101 01001100 01001100 01001111 00100000 01010100 01001000 01000101 01010010 01000101 00100001 to the computer.

Technically, we can write software with just ones and zeroes, but that's a bit difficult. Because of this, computer scientists have created what are known as computer programming languages which resemble human language a bit more. These computer scientists also wrote software that break the programming language down into ones and zeroes so that the computer can understand it. Thus, we can communicate with the computer by writing code using a programming language - and then another piece of software that interprets that code into ones and zeroes so that the computer can understand what we're telling it.

In the end, the bulk of a developer's day is writing code using a programming language like we've described. Examples of languages that you may have heard of include HTML, JavaScript, Ruby, and Python. In fact, there are hundreds of programming languages in existence, and each one specializes in something unique. For example, HTML is used for websites, and SQL is used for databases. At times, multiple languages are used together. In fact, most websites include a combination of HTML, CSS, and JavaScript in addition to other languages as well.

In some companies, developers may write code for nearly 100% of the time, while in other companies the day may also contain meetings of various sorts. These may include meetings of developers amongst themselves, developers with business managers, or developers with outside clients. These meetings are often great vehicles for allowing developers to understand exactly what they're being asked to create.

What Skills Does it Take to Code?

There have been interesting discussions about what skills coding actually entails. Is learning a programming language like learning a new language - or is it more similar to learning math? There have even been studies that measure which parts of the brain are activated when coding.

The truth of the matter - based on my personal experience as a developer and educator - is that learning to code is not really like learning a foreign language, nor is it like learning math. It's really a self contained subject that boils down to the ability to do one thing: Communicate with a hunk of metal that has no intuition.

Before computers existed, the ability to communicate with mindless metal was a completely useless (and impractical) skill. "Computer Science" only became a thing when computers were first conceptualized.

Because talking to metal is an independent skillset, that results in both bad news and good news.

The bad news first: No one is born with an innate ability to learn to code. It requires lots and lots of practice.

The good news, however, is that anyone can learn to code. You can be bad at math and foreign languages (like me!), and still become a software engineer.

Learning to code is really about learning a new way of thinking: How can I get this piece of metal to do what I want? And this is very different than the communication you're used to - which is the communication of human to human.

ACTUALIZE

If someone called you on the phone and asked how you made that delicious stew you served last night, you might say: "Oh, it was easy! I just threw in a bunch of potatoes, ground beef, and spices - I'll send you the list of spices in a text."

Now, let's imagine that we were to describe to a computer how to make that same stew. (Let's even assume the computer had robotic arms so that it can actually make the stew itself!)

We'll begin with the potatoes. To the human, you said "I just threw in a bunch of potatoes." Now, a computer has absolutely no idea what to do with that instruction. Why? Here are just a few reasons:

- Since it doesn't know how to estimate, it doesn't know how many potatoes to put in. You need to give it an exact number.
- You never mentioned that you're placing the potatoes into water! Without specifying that, the computer will skip that step.
- You also never mentioned peeling the potatoes - so the computer will add them unpeeled. Eww!
- It might literally throw the potatoes, causing them to smash into bits.

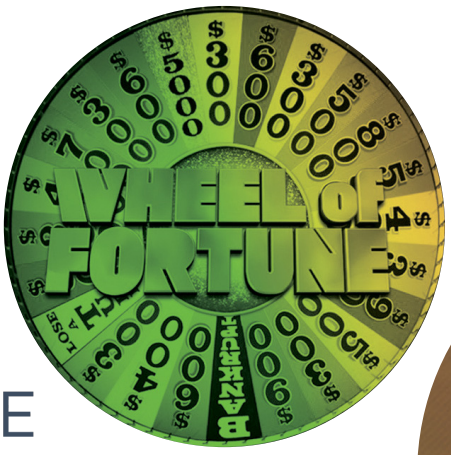
Let's discuss a more realistic example. Imagine that you're coding a Wheel of Fortune app, where a user can play that game show against computerized contestants. Building the game so that there's a wheel, letters, and puzzles is the easy part. But how can you make it so that the computer plays against the human user in a realistic way? If the computer does not truly have a brain of its own, how can you make the users feel like they're playing against other intelligent human beings?

If we analyze the situation, we're faced with a very challenging set of constraints:

- The computer doesn't have any real world knowledge of phrases, sayings, or concepts.
- The computer doesn't know which letters form a word. To it, "asdfads" is a word just like "pizza."
- The computer already knows the true answer to each puzzle! If it's loading the puzzle in the first place, and can check whether the user's guesses are correct, by definition it already knows the puzzle's solution.

Now, there are some things that a computer can do that we may be able to take advantage of:

- A computer can be given any set of arbitrary rules that we provide it.
- A computer can randomly select an item from among a set of stuff.
- A computer can count and make mathematical calculations.



ACTUALIZE

Software engineering boils down to figuring out how to use the computer's tools to accomplish a particular task despite the computer's constraints. How can we utilize the computer's abilities to make a human-like Wheel of Fortune player?

Here's one approach, that although simplistic, may be good enough that a human might feel like they're playing against another real human player:

We're going to write code that gives the computer the following set of rules:

- We're going to divide consonants into three categories: The **common** letters are [T N S R H L]. The **semi-common** letters are [D C M F P G]. The **uncommon** letters are [Y B V K X J Q Z]. (I just Googled "most common letters of alphabet" and found this info.)

- When it's the computer's turn, it will first randomly select one of the **common** letters. When all those letters are exhausted, it will then randomly choose a letter from the **semi-common** category. Finally, when all of those letters are exhausted, it should randomly pick one of the **uncommon** letters.

- The computer should buy a random vowel on its every third turn if it has money to do so.

- If 85% or more of the letters of the puzzle have already been revealed, the computer should just "solve" the puzzle. Really, it knew the solution the entire time. But when the 85% of the letters are revealed, it just reveals the rest of the letters and wins the round.

We just worked with the computer's capabilities to mimic human intelligence!

Learning to code boils down to one thing: ***It's learning to give a computer a set of very specific instructions so that it can perform a particular task.***

Is this a skillset that people have naturally? Absolutely not. But anyone can learn it. I've seen people of all backgrounds go through Actualize and succeed in the field of software engineering. I've seen construction workers, taxi drivers, social workers - people whose backgrounds have nothing to do with coding - all successfully launch careers as software engineers.

While learning to code is primarily learning how to communicate with a computer, there are a number of character traits that are also important to be a successful software developer:

PERSISTENCE: No day goes by in the life of a developer where you aren't working through tough problems. Some coding is routine, but most of it is solving a challenge you've never encountered before. In fact, many of these problems are challenges that no one has ever solved before. If you're looking for easy street - or a job where you do the same thing every day - coding is not for you. However, if you're ready to wake up every day and tenaciously tackle new problems - some of them which may take hours to solve - then coding might be right up your alley.

CREATIVITY: Most people associate creativity with the arts. However, software engineering is truly a creative process that makes you use every part of your brain. Here's a little puzzle: Imagine that you are locked in a room whose walls are made of wood, and you have a match, a thimble, a glass bottle, and a rope. How can you escape? I have no idea what the answer is, but you might already start to feel your creative juices flowing when asked such a question. Coding is the art of getting things done despite constraints. It's like solving a puzzle: You're given a mindless computer, that has limited but powerful abilities - now go figure out how to achieve X. There are no wrong approaches - you can use all the computer's capabilities to get the job done. How do you do it? If that excites you, software development may be fun enough for you that it may not even seem like a job.

ACTUALIZE

ATTENTION TO DETAIL: If you're missing even a single comma in a file containing thousands of lines of code, it's possible that your program won't even be able to run. If that "s" is lowercase when it should be uppercase, that might also make your code flop. Now, the good news is that when you have a bug in your code, an error message will let you know which line of code is problematic and what about that line is faulty - so you don't have to go searching for a needle in a haystack. However, you do need to care about the details. Detail oriented people make for very good software engineers.

ALWAYS BE LEARNING: Technology is always evolving, and the stuff you know how to do now might change significantly in a year or two. If you want a job in which you learn a set of skills and then stick to those skills for the rest of your life, coding may not be your cup of tea. A developer always needs to be learning new things to become more effective and to stay relevant. If you love to learn, and want to always keep your brain in full gear - then being a software engineer might be your thing!

Where Do Coders Work?

Coders work for companies of all sizes and in every industry. While it's easy to understand why a developer might work at Microsoft - a company that develops software - it's perhaps less intuitive why a developer might work for Home Depot.

ACTUALIZE

The explanation, though, is simple. Every company relies on technology in some way. A hardware store may have a website and tens of software tools for internal management. From inventory trackers, automated product ordering software, and human resource management tools, a retail store may have a lot more software running its business than you might imagine. And the same applies for every organization in every sector.

The truth is that a developer's day-to-day is affected much more by the employer's size rather than industry. Working at a small, five person Silicon Valley startup is a very different experience than working for IBM. Let's talk about the different scenarios.

Startups: Startups can have anywhere between 1 and 100 employees. I once worked at a ten person startup in which I was the only developer. Other startups have a whole team of developers. Working at a startup is usually exciting, but also comes with greater risk and sometimes less work-life balance. Some find working at a startup thrilling since their impact on the company is very large and noticeable, and the company itself often seeks to make a large impact on its given industry.

Midsize companies: For the purpose of this book, I'm defining a midsize company as one with 100 to 500 employees. The truth is that some midsize companies have only a few developers, while others may have dozens of them. In a company with more than ten developers, they are typically divided into teams. You'd work with your team on a single project at a time, splitting up the work among your teammates.

Large companies: Large companies usually come with large company culture - which tends to be a bit more bureaucratic - and you may be one of hundreds of developers. In many large companies, there may be many teams of developers, and you may not even know all the other developers in the company. As with a midsize company, large companies divide their developers onto various teams that each tackle a different project.

TV and movies often depict the perks of working as a software developer. You may have seen pictures of Google's crazy offices, where lunch is served every day and dry cleaning is taken care of in-house. While it is true that these perks do exist at some companies, it's relatively rare and everything depends on the company you're working for. If you're a developer for a typical large company, you will likely not have many perks beyond what every other employee of the company gets. Deciding to become a developer because of the perks is not a great idea.

What you do get as a software developer is a career that's enjoyable (for the right person) and stable, and the ability to command a high salary that has the potential to keep growing.

We'll talk more about salaries in the next section, but I wanted to focus a bit on career stability. One of the most promising things about becoming a software engineer is that our world is become ever more dependent on technology. High-tech - by definition - is something that can never go away. While robots may be taking the jobs of people who perform certain tasks, the world will always need people who can program those robots.

According to Code.org, based on the Bureau of Labor Statistics, there are over 500,000 open jobs in computing. (This includes software engineering and some other related fields as well). However, less than 43,000 people graduated in computer science last year. In other words, computing jobs will be in demand for as long as the eye can see. Additionally, ComputerWorld reports that tech jobs are projected to rise on the whole by 22% through 2020. Software developers in particular can expect a 32% percent increase by that time.

There's an entire industry around trying to get a developer from one company to switch to another company. Since it's difficult to find developers who are out of a job, technical recruiters are always bombarding developers with enticements to join the company that the recruiter represents. Recruiters get paid when they successfully nab a developer for the company. I personally receive emails from recruiters every week who are looking to hire me as a developer - just because my LinkedIn profile says that I have development experience. The fact that my LinkedIn profile also says that I run my own business doesn't seem to matter. That's how desperate they are to find developers!

In short, a career in software engineering is a career with longevity. While many other jobs may be going away, coding is here to stay. (The rhyme was entirely accidental, I promise.)

How Much Do Software Engineers Make?

The salaries of software engineers depend primarily on two factors: Experience and location. Let's explore experience first.

Let's look at how experience determines salary in the city of Chicago. The following is typical, but your mileage may vary:

0 - 1 years experience:
$50,000 - $65,000

1 - 2 years experience:
$65,000 - $90,000

3 - 5 years experience:
$90,000 - $120,000

We'll talk about 5+ years in a moment, but first let's talk about location.

ACTUALIZE

The salary range for each level of experience varies by location. Generally, software developers in major cities command higher salaries than those in smaller towns. However, there's an important tradeoff to consider: The cost of living in major cities is significantly higher than that in smaller locations.

Silicon Valley, which consists of San Francisco and its surrounding environs, is well known for paying extremely high salaries for software engineers. For example, entry level developers in San Francisco can be paid as much as $100,000! However, the cost of rent in San Francisco can easily be double that in other cities.

After 5 to 10 years of coding, there are several tracks available to software engineers regarding the next step in their career path.

Tech Lead: Some developers never want to stop being in the thick of code. They simply love building things and using their creativity to solve tough problems. These developers often become the leaders of a team of developers, and go by various titles such as Tech Lead or Principal Developer.

Software Architect: Software Architects make many of the high level technical decisions regarding how the software should be built. For example, they may select which programming language to use, or what type of information should be stored in the database. They set the course for the software engineers, but usually don't actually jump into the code itself.

Engineering Manager: Sometimes known as Director of Engineering, an Engineering Manager manages the software developers. Among other things, the Engineering Manager ensures that the developers are making progress on their project, and does what she or he can to help ensure their success. Most of what Engineering Managers do is people-work, and can be very rewarding.

CTO: The Chief Technical Officer, sometimes known as the Chief Information Officer (CIO) or VP of Engineering, is essentially in charge of the entire software development department of a company. The CTO is very much a business person, making executive decisions about the company's technology and tech projects, and collaborates with the CEO and other leadership about many business decisions. This is a level up from Engineering Manager, as Engineering Managers usually report to the CTO.

Consultant: Many software developers go out on their own, and work for themselves by working short or long term contracts with companies in which they do a variety of technical work. This work may range from actual coding to software architecting to managing software engineers. Consultants make whatever they decide to charge, which can be a lot of money. Consultants with a lot of experience have a lot of leverage in this regard.

There's no single career path for a software developer. A software developer may become a Software Architect and then an Engineering Manager, or they might have the opportunity to become an Engineering Manager immediately. An Engineering Manager may eventually become a CTO, or can become a consultant. Even regular software developers can go straight into consulting if they wish. In addition, many companies have all sorts of positions and roles with various titles, making it impossible to include all of them here.

To complete the salary discussion, let's return to our Chicago example: Tech Leads, Software Architects, and Engineering Managers can make salaries that range between $120,000 and $150,000, and CTOs can make even more. And as we mentioned, consultants can charge whatever they want if they know how to market themselves.

Of course, some people decide to create their own tech companies, and then the sky's the limit!

What About You?

Deciding whether software engineering is a career to pursue is a major decision, but I'll present some factors and advice that can help you decide.

Firstly, let's sum up some of the major points to consider.

- Software developers make a very good salary with a clear career trajectory that keeps their salaries increasing at high rates.
- Software development is a very stable career.
- Software development is a very creative process.
- Software development requires a special kind of ingenuity - which can be acquired.
- Software development requires focus, continued learning, and the willingness to tackle problem after problem.

How do you know if you'd like coding as a career?

One fortunate thing is that there are many free resources that can give you a taste of what coding is like. Here are some:

- Anyone Can Learn To Code
- Codecademy
- Free Code Camp

While these and other resources can provide a sense of what coding is all about, it's not a substitute for seeing firsthand what working with code professionally is like. To do that, I recommend that you reach out to local companies that employ developers and simply ask if you can get a tour and the opportunity to talk to the developers. Tell them that you're considering a career in software development, and you want to see firsthand what it's like. You'll be surprised by how many companies would be happy to show you around.

Finally, we at Actualize would be happy to tell you more. Our Advisors can learn about your career goals and see if coding might be the right option for you. If it is, we can help you chart your path to get there.

If you have any more questions, please don't hesitate to reach us at: hello@actualize.co!

ACTUALIZE