I have been a software engineer for a little bit over 5 years now. My dad became a software engineer in the 1990s, and although I never learned to code when I was a child, I grew up watching him endlessly tinker with various computers and digital tools. This is what drove me to take C++ and Introduction to HTML/CSS modules while studying Physics at university. I greatly enjoyed it, and, after my degree, I transitioned to software engineering and haven’t looked back.

Over the years, I have helped a few of my friends make the switch to software engineering. In this guide, I want to share the same advice I gave them – not just about the technical skills, but also about the mindset, resources, and strategies to succeed in this field.

Whether you're starting from scratch, transitioning from another career, or simply curious about what it takes to become a software engineer, this guide is for you. I’ll walk you through the essential skills, learning paths, and practical steps that can turn your interest in technology into a fulfilling career. Along the way, I’ll also share some of my personal experiences and lessons to help make your journey smoother and less daunting! And if, at the end of this article, you are serious about the industry, enroll in our Software Engineering Principles in Python course.

Understanding the Role of a Software Engineer
Let’s break down the role of a software engineer.

What is software engineering?
Software engineering is the process of designing, developing, testing, and maintaining software systems to meet user needs and solve real-world problems.

It’s a discipline that combines creativity, analytical thinking, and technical expertise to create reliable, efficient, and scalable software solutions. At its core, software engineering is about writing code, yes, but it's also about understanding complex systems and delivering value through technology.

It’s important to note that while the terms software engineer, software developer, and programmer are often used interchangeably, they have subtle distinctions in practice:

Software engineers: Focus on the bigger picture of software systems. They consider architecture, scalability, and maintainability, often working across the entire software development lifecycle to ensure the final product meets both technical and business requirements.
Software developers: Specialize in writing code to build specific features or applications. While they may also contribute to design and testing, their primary role is turning ideas and specifications into functioning software.
Programmers: Typically emphasize the act of coding itself. They are skilled at writing efficient code in specific programming languages but may not always engage in the broader design or system-level considerations.
Each role plays a part in bringing software projects to life, and depending on the company or project, these roles may overlap or be distinct.

Key responsibilities of a software engineer
Software engineers are responsible for designing, building, and maintaining software systems that solve problems and deliver value to users. While the specifics of the role may vary depending on the industry or company, several things are common:

Coding and development
Writing clean, efficient, and maintainable code is at the heart of software engineering. Engineers often work with multiple programming languages, like Python, JavaScript, Java, and C++, depending on the project’s needs.

Python: Popular for data science, automation, and web development.
JavaScript: Essential for creating dynamic and interactive web applications.
Java: Widely used for Android app development and enterprise-level backend

systems.
C++: Often employed for performance-critical applications like gaming, embedded systems, and real-time systems.
Learning multiple languages is important to stay versatile and adaptable to different challenges and industries, but focus on one if you are a beginner. When I started coding,I made the mistake of trying to learn Python, Javascript and Java at the same time and ended up spreading myself way too thin!

System analysis
Understanding user requirements and breaking down complex problems into manageable parts is a very important skill, and perhaps the one that takes the longest to master. The goal here is to analyze how software systems will interact with users and other systems to ensure they meet performance and functionality requirements.

Debugging and maintenance
Software engineers must troubleshoot issues, fix bugs, and update software to adapt to changing needs or technologies.

Things You Should Know as a Software Engineer
Of course there are many programming concepts, languages and design patterns you should know as a software engineer, but there are also broader concepts that will help you land your first job.

Software development lifecycle (SDLC)
The software development lifecycle is a structured framework that guides the creation and maintenance of software systems. This helps ensure efficiency, consistency, and quality. The key stages of the SDLC are the following:

Planning
In this initial stage, teams identify project goals, gather requirements, and evaluate feasibility. Planning sets the foundation by defining timelines, budgets, and the scope of the project.

Design
Engineers create detailed software architectures and system designs based on the requirements. This stage often involves selecting technologies, defining data flows, and outlining how components will interact.

Development
The actual coding takes place here. Engineers implement the design, writing and testing of individual components of the software. This is often the most time-intensive phase.

Testing
Rigorous testing ensures that the software is functional, secure, and free of bugs. Techniques like unit testing, integration testing, and user acceptance testing are used to verify that the system meets its requirements.

Deployment
Once tested, the software is released to users. This stage may include setting up environments, configuring servers, or distributing the application through app stores or websites.

Maintenance
Post-deployment, the software requires ongoing updates and bug fixes to adapt to user feedback and evolving needs.

Understanding and adhering to the SDLC ensures that projects are delivered on time and meet quality standards. It also helps engineers work more effectively in teams, manage risks, and adapt to changes during development.

The SDLC framework can be implemented using different methodologies, each tailored to specific project needs. You might have already heard of Agile and

Waterfall methodologies, or subsets of the former like Scrum or Kanban. We won’t dive into them in this guide, but I would recommend reading about them a little bit if you have time. They will teach you about the different ways of working you can encounter in an engineering team and allow you to hit the ground running when you join a company without getting bogged down by the processes.

Version control

Version control is an essential tool in software development, and allows teams to collaborate effectively and manage changes to codebases. It ensures that multiple developers can work on the same project simultaneously without overwriting each other's work.

The most widely used version control system today is called Git. It can take some time to get the hang of it, so I would really recommend you start using it as soon as possible. Most engineering managers I know look at the version control skills of a junior engineer before hiring them, because work being frequently deleted or overwritten by mistake can significantly slow down a team!

If you are new to Git, I would recommend this GitHub Foundations Certification. It is a beginner-friendly course where you’ll learn Git and GitHub basics, and will reassure prospective employers that you can collaborate with your team from the get go.

Problem-solving and analytical skills

Problem-solving and analytical thinking are the most important skills to have in software engineering. Picture debugging a tricky issue, optimizing code for better performance, or designing a complex system. The idea is to approach each problem methodically: define the problem, brainstorm solutions, evaluate trade-offs, and test thoroughly.

If that isn’t something that currently comes naturally to you, don’t panic. There are many ways to practice this, and like most things in life, it will become easier the more you do it.

Coding challenges are one of the best ways to develop your problem-solving mindset. The best platforms to practice these are CodeWars, HackerRank and LeetCode. They offer a wide range of problems, from beginner to advanced levels, and cover key topics like algorithms and data structures. And as an added bonus, you will encounter these same coding challenges in interviews. So, by practicing your analytical skills, you will also be practicing the same coding challenges that will make you pass your interview!

Educational Pathway to Becoming a Software Engineer

There are many ways to become a software engineer, and the right one for you depends on your unique situation and factors like your ability to self-motivate and be consistent, the amount of time you want to spend learning, and the amount of money you have available.

The traditional path: Computer science degree

Best if: You want an academic foundation and have the time and resources.

A computer science degree has long been considered the classic entry point into software engineering.

A degree provides a structured education in:

Programming fundamentals: Courses in languages like Python, Java, or C++.
Data structures and algorithms: Core concepts for solving computational problems efficiently.
Systems knowledge: Modules on operating systems, networks, and databases.
Software engineering principles: Team projects that simulate real-world development.
Some companies still require their employees to have a formal Computer Science degree. In most countries, like the USA or the UK, the culture has shifted and

you can become a software engineer without a degree.

Pros	Cons
Provides a deep understanding of computer science fundamentals	Time-consuming (3-4 years)
Opens doors to internships and entry-level roles with ease	Expensive, depending on location and institution

The condensed path: Bootcamp
Best if: You want a quick transition into the industry with structured guidance.

A bootcamp usually lasts 3 to 6 months, full-time.

Depending on the focus of the bootcamp, you might learn mobile or web development but the curriculum will always include topics like coding, debugging and deploying applications.

Most of them are beginner-friendly and the price can vary greatly. I went through a bootcamp in the UK, just after university and was lucky enough to get a full scholarship, so make sure you check if you’re eligible for this type of funding.

Bootcamps often include career services, such as portfolio-building projects to showcase skills, mock interviews and resume reviews, and networking opportunities with hiring partners. This means you have good chances of finding your first job quickly. Some bootcamps even have guarantees in place to make sure you get placed within 3 months of graduating.

Pros	Cons
Fast track	High cost
Hands-on learning	Limited depth

The self-directed path: Online courses and certifications
Best if: You are an independent learner who values flexibility and cost-efficiency.

Self-directed learning is the most flexible and affordable route to software engineering. The fact that you can learn at your own pace allows you to condense all your learning over a short period, or learn while keeping your current job or juggle other responsibilities.

DataCamp offers many great courses at a fraction of the cost of a bootcamp or university degree. For instance, our Software Engineering Principles in Python course covers the basics of working with Python and is super affordable. Our Case Study: Building Software in Python is also very helpful.

If you decide to be self-taught and know a software engineer, I would highly recommend you ask them to mentor you. They can help you with general guidance, career advice and provide code feedback amongst other things.

The biggest issue when being self-taught is that you don’t have the credibility associated with bootcamps or degrees, and you might struggle more to get your first job. This is where networking and building your portfolio with your own projects or open-source contributions can really make a difference.

Pros	Cons
Cost-effective	Can be difficult to get a job
Very flexible	No personalized structure or guidance

Gaining Practical Experience
No matter the path you choose to become a software engineer, you need to have a little bit of practical experience to demonstrate your skills, build confidence, and understand that you know how theoretical concepts apply to real-world problems.

Portfolio projects
Building a portfolio is one of the most effective ways to showcase your skills

as a software engineer. A strong portfolio not only demonstrates your technical abilities but also highlights your problem-solving mindset and creativity. Employers and clients often value seeing real examples of what you can build over just reading a resume, especially when applying to your first software engineering job.

You could, for instance, build:

Web applications: Build a functional web app, such as a to-do list manager, e-commerce site, or social media dashboard. This demonstrates your front-end and back-end development skills.
Data visualization tools: Create an interactive dashboard or a visual representation of complex data. This is particularly valuable if you're interested in data science or analytics.
Automation scripts: Develop scripts that solve everyday problems, such as scraping web data, automating repetitive tasks, or setting up custom notifications.
Properly documenting your projects is just as important as building them. Try to include:

Problem statement: Clearly define the problem your project addresses and the goals you aimed to achieve.

Code samples: Share your code on platforms like GitHub, with comments explaining key sections. Use a README file to outline your project and its features.

Outcomes: Highlight the results, such as how the project improved efficiency, solved a specific issue, or provided other insights.

Pro Tip: Start small, but aim for projects that reflect your personal and career interests. For example, if you’re drawn to mobile app development, try building a basic app using React Native or Swift.

Internships and freelance work
Both internships and freelance work offer excellent entry points to gain practical experience, build professional networks, and enhance your employability. You’ll get to work on real-world projects and develop industry-relevant skills.

Internships are structured and often short-term roles that give you a glimpse into how software development works within an organization. You’ll have a chance to learn a lot in real-life conditions, with professionals that are there to help you do just that. You’ll expand your network and you might even get a job offer from it! I usually advise to accept unpaid internships if they offer substantial learning opportunities and align with your career goals. However, make sure you prioritize roles that respect your time and effort.

If you already feel comfortable, you can take on independent projects for clients as a freelancer. It’s ideal for those who want to explore different industries or work remotely. Common freelance platforms include Upwork, Fiverr, and Topta, but don’t overlook the power of networking and reaching out directly to startups or small businesses. I now work full-time as a freelance consultant and have found almost all of my clients through word-of-mouth recommendations!

Certifications and continuous learning
In software engineering, your ability to learn quickly and adapt is often more valuable than the specific skills you currently possess. Because the tech industry is constantly changing, continuous learning is essential to stay competitive and relevant.

I would recommend subscribing to tech newsletters, blogs, substacks, reddits and stack overflow groups. You’ll get updated on the latest technologies, security practices, or engineering soft skills. And you will also get to see the amazing projects that people share there!

Certifications are also useful, as they can help showcase your expertise in specific tools, languages, or methodologies. There are a lot of certifications for various areas of expertise, like:

AWS Certified Developer – Associate: Ideal for those interested in cloud computing, this certification demonstrates knowledge of core AWS services, architecture, and deployment.
Certified Scrum Master (CSM): For engineers looking to step into project management or Agile environments, this certification validates your understanding of Scrum principles and practices.
SQL Associate Certification: To demonstrate that you are capable of using SQL to extract appropriate data from a database, and use it to answer common data questions.
They are not essential for landing your first job, but might come in handy afterwards!

Building soft skills
Soft skills complement technical expertise, making you a well-rounded engineer. Remember, coding may get you through the door, but soft skills will keep you in the room and help you climb higher.

Some of the key soft skills for software engineers are:

Teamwork: The ability to work harmoniously with others to achieve shared goals. Agile methodologies, for example, require close collaboration.
Adaptability: The tech landscape evolves rapidly; being open to change and learning new tools or approaches is a crucial asset.
Time management: Balancing multiple tasks, meeting deadlines, and prioritizing work.
Empathy: Understanding the perspectives of users and team members leads to better designs, solutions, and relationships.
The stereotype of the software engineer locked in a dark room drinking cans of energy drinks is just that: a stereotype. Good engineers with great soft skills consistently do a much better job than amazing developers who cannot or do not want to interact with the rest of the team!

Applying for Software Engineering Jobs
Landing your first software engineering job can feel a bit scary, but with the right preparation and strategy, you can stand out from the crowd.

Prepare a good resume
Highlight technical skills, portfolio projects, certifications, and any relevant experience. Include a concise technical summary that showcases your expertise.

Focus on achievements rather than responsibilities. For instance, instead of “Built a web app,” write “Developed a web application that increased user retention by 15%”.

I haven’t really seen cover letters in the industry in the past few years. If you have to write one, address specific qualifications in the job posting and explain how your skills align with the company’s goals. Use this as an opportunity to showcase passion and enthusiasm for the job.

Find jobs
Platforms like LinkedIn, Wellfound, and Stack Overflow Jobs are great for tech-specific roles, but there are dozens of other platforms that are just as popular!

Interview preparation
Practice coding challenges on platforms like LeetCode and HackerRank, and brush up on algorithms, data structures, and system design principles. Make sure you prepare for behavioral questions too, like "Describe a time you overcame a technical challenge."

If you have any in your network, reach out to experienced engineers for career advice, project feedback, and interview preparation tips.

Try not to feel disheartened if it takes time. It took one of my friends 9 months of learning and 6 months of applications and interviews before getting his first job, but 3 years on he is a very happy mid-level engineer at a reputable company!

How Long Does It Take to Become a Software Engineer?

Becoming a software engineer is a highly individual journey, and timelines can vary widely depending on your chosen path, prior experience, and learning approach.

Your timeline to becoming a software engineer is influenced by several factors. For instance, if you have prior experience in STEM or programming, you may find yourself progressing at a quicker pace than someone starting from scratch.

The amount of time you can dedicate to learning also plays a significant role. Full-time learners often complete their journey faster, whereas part-time learners who balance other responsibilities might take longer.

The resources you use also make a difference. High-quality tools like interactive courses, mentorship programs, and coding platforms can really accelerate your progress.

Regardless of how long it takes, it’s essential to focus on your own journey. Every skill you acquire and every project you complete is a meaningful step forward. Avoid comparing your progress to others, as everyone’s path is unique. Celebrate your achievements, no matter how small they may seem, and stay motivated by breaking your journey into manageable milestones. And don’t forget to reward yourself for reaching those milestones to maintain momentum!

Conclusion

Becoming a software engineer is an exciting and rewarding journey, but it requires time, dedication, and a willingness to continuously learn and adapt. Whether you take the formal route through a degree, enroll in a bootcamp, or teach yourself through online resources, the most important thing is to keep moving forward. Build your skills with hands-on projects, seek out practical experience, and never stop learning.

Remember, every successful software engineer started somewhere – often with a simple curiosity and the determination to turn that curiosity into knowledge. So take the first step today, and embrace the challenges and opportunities that lie ahead. You’ve got this!

If you want some extra motivation, have a look at some inspiring journeys into software engineering with these Datacamp employee spotlights:

Hettie McConnell, Software Engineer
Forging a Non-Traditional Path From Fashion to Software Engineering
Also, finally, make sure to tune in to the DataFramed podcast, which has many great episodes, including ones focused on software engineering, like this one: The Unreasonable Effectiveness of AI in Software Development. This episode is with Eran Yahav, the CTO of Tabnine, who will also be speaking at DataCamp's Radar: Skills Edition coming up on March 27, 2025. So make sure to register!