Role of a Software Engineer

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the role of a software engineer by discussing:

1. A BROAD VIEW
2. DIFFERENT LANGUAGES, ALGORITHMS, AND SERVERS
3. DAILY WORK OF A SOFTWARE ENGINEER
4. CODE REVIEWS AND PAIR PROGRAMMING

Software Engineer

Hi, I'm Monique. I'm the Software Engineer at Poodle Jumper and I've been working here for nine months. Before I joined tech, I taught French and Spanish to high school students. I loved teaching but I needed a career with more flexibility. I enrolled at Hackbright Academy, a Software Engineering bootcamp for women based in San Francisco. I'm not going to lie, the bootcamp was challenging. For 12 weeks we started lectures at 10:00 am and spent the entire day learning, practicing, and creating projects. Making a career pivot later in life was intimidating, but I love the community of talented engineers I joined.

The schedule flexibility I have as an engineer makes it possible to raise a young family and have a career that challenges me. I've been able to leverage my teaching skills to mentor other engineers who are just starting out.

1. A BROAD VIEW

There are many niches of software engineers; one you've already learned about iOS Engineering and Web Development. The term 'Software Engineer' is broader than those roles. The low-level problems solved are very similar, but the high-level problems are different. Software Engineers have additional responsibilities to understand how computer memory works, creating logic or algorithms, and managing how software interacts across the different layers of technology. You may hear these systems referred to as the backend. Backend technology is the servers, applications, and databases that work behind the scenes to power the interface users interact with.

Software engineering is a unique discipline that combines features typically exclusive to mathematic, scientific, and creative fields. Like mathematicians, they use specialized languages to denote ideas. Like scientists, they must understand the rules that govern how code runs on computers. Like designers and engineers, they create solutions by assembling individual components into systems and evaluate tradeoffs among alternatives.

You don't have to love math, be a trained scientist, or a skilled designer to excel as a software engineer. The most important skill for a software engineer is problem-solving—the ability to formulate problems, devise creative solutions for them, and communicate those solutions effectively. The types of problems that Software Engineers encounter can vary, but the way they approach building solutions to those problems is similar.

1. Develop a general solution.

2. Communicate the solution to a computer, in a way that the computer understands.

3. Use the computer to automate the execution of that solution.

At their essence, a Software Engineer writes code to solve problems and create computer systems or applications that do something useful.

2. DIFFERENT LANGUAGES, ALGORITHMS, AND SERVERS

While Ruben focuses on the web application and Camilla focuses on the iOS apps, my role is to make the code that ties it all together, keeps it in sync, and enforces the business logic.

My experience with multiple languages has been really helpful for learning the syntax or structure of different programming languages. In my role, I work with multiple different programming languages depending on what I'm trying to do. This can be a tricky part of starting in software engineering because [brackets], {curly braces}, and 'single quotes' do different things in different languages, but like most things–it gets easier the more you work with it.

This image shows a bubble sort algorithm in three different languages.

Bubble sort is the first sorting algorithm most developers learn because it is simple and easy for small data sets, but it is notoriously inefficient in real-world applications. It's so bad that it's become a joke in the tech community. The left section of code is in Python, the middle is in JavaScript, and the right is in Rust. Don't worry about being able to understand the code, but you can see differences in the structure, formatting, and syntax.

The backend systems I manage include powerful computers, also known as servers, that send information, or data, to code that displays for users. Ten years ago, most companies had to buy and manage expensive servers on-site, but now, most companies leverage servers from a cloud technology provider such as Microsoft, Google, or Amazon.

When someone says "It's in the cloud," what they mean is that it is in a server somewhere else. The "cloud" is a really large network of computers that talk to each other, just like the internet. As the engineer, I know where those servers and data are physically located. I deploy, or release, the updates to the code on the servers and make sure our code and data are backed up regularly in case of a service outage.

I'm responsible for managing the server's performance and memory. I set up monitoring with alerts to tell me when something is out of the ordinary. This can happen at any time, day or night, so the team and I have an on-call rotation to make sure someone is always available to solve an issue that might come up. This is a lot of responsibility. When I'm on-call, I keep my computer close and my notifications on.

BIG IDEA

Are you someone who hates math, and you are afraid that it may get in the way of your ability to learn how to code? A **recent study published in Scientific Reports https://www.nature.com/articles/s41598-020-60661-8** found that language aptitude accounted for a higher variance in learning outcomes than numeracy. Don't let stereotypes get in the way of your career options. We need more engineers with diverse backgrounds and perspectives building technology.

3. DAILY WORK OF A SOFTWARE ENGINEER

The code that I'm responsible for includes the servers and databases mentioned in the last section as well as applications that control the logic for how things work.

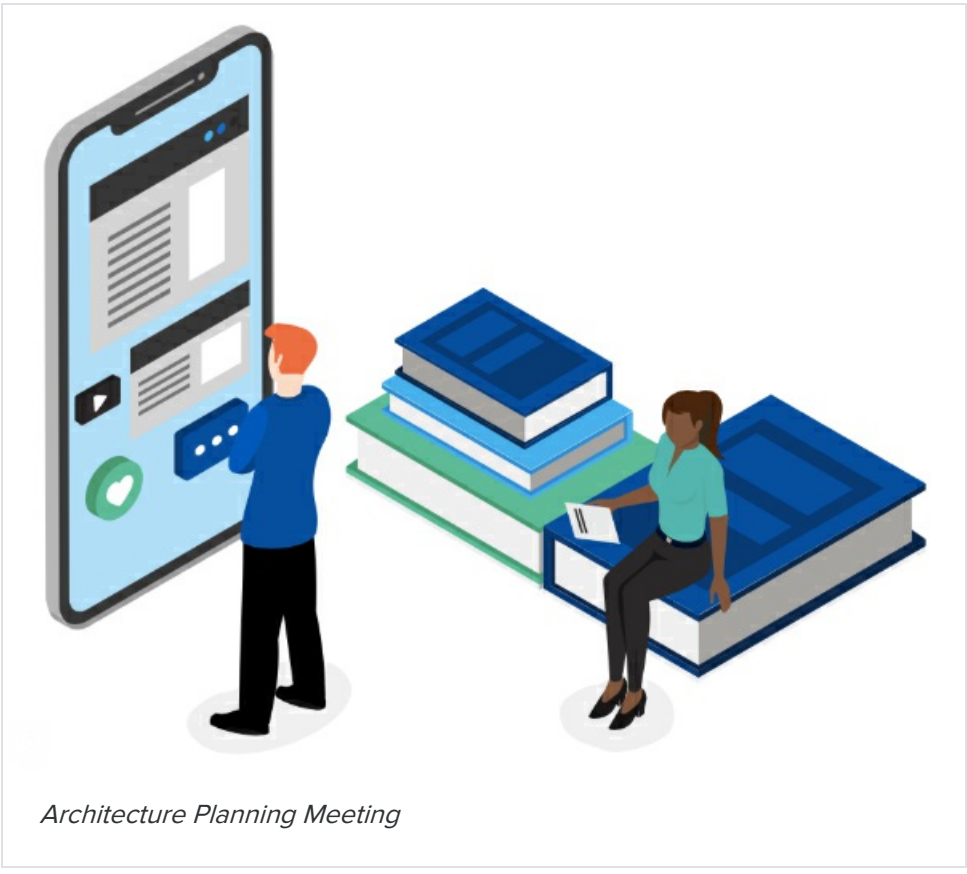
In the last iteration, I created a recommendation engine to help customers view the service providers based

on their reviews and the reputation of the service provider. I create integrations between different parts of our technology stack, but also integrations to third parties. When Camilla was creating the iOS app, I helped create an integration to send notifications to our users through Apple's Push Notification service.

In addition to the code I write, I'm responsible for collaborating with the dev team. My coworkers told you about some of the regularly scheduled meetings we have.

I'm going to show you more about engineering collaboration that happens. You might think that engineers spend a lot of time working alone based on the stereotypes, but that isn't the case for most engineers. We spend a significant amount of our time in meetings and collaborating. Collaborating with your peers feels more like conversations and less like traditional meetings because they happen as needed.

The first collaboration point is an architecture planning discussion. This is where the developers and engineers get together to diagram or whiteboard, how different systems will integrate. We identified dependencies, requirements, and possible solutions. We focus on creating a solution that is flexible so it can change as needed, scaleable so it can grow to support a large number of users without significant cost, modular so parts of the code can be reused in other places, and reliable so we can count on it working. We plan who is going to complete the different development tasks and make sure that we've documented the design.



Architecture Planning Meeting

4. CODE REVIEWS AND PAIR PROGRAMMING

Code reviews are the most common way engineers and developers collaborate. A code review is a process where your peers review the code you've written before it is accepted and merged into the main code

repository. This allows us to check for mistakes with a fresh pair of eyes and ensure we have the same understanding of the requirements. Because everyone has their own way of writing code, I need to make it presentable and readable, so it is easy to understand.

Feedback from code reviews can be contentious if you aren't self- and socially aware. Critiques can be painful, especially if you haven't built a productive relationship with your peers. When I receive criticism for my code, I try to understand where they are coming from and look for ways to improve my skills. Reviewing my teammate's code allows me the opportunity to share my experience and expertise with them as well. This creates a great culture of learning.

When engineers find a problem particularly tricky to solve, we like to leverage pair programming. A collaborative technique where two individuals program on one computer. This allows us to talk through the smallest details and complexities together. One person types while the other person describes what the code should be doing at a high level. Then, they switch roles. It can be awkward at first to vocalize the thoughts you have when coding solo, but practicing explaining complicated ideas clearly and concisely improves your communication skill and helps you write better code.

Another way we collaborate is with written documentation. Our development team keeps a wiki to capture important decisions, requirements, architecture designs, standards, and best practices. Everyone is responsible for contributing and maintaining the wiki.

When writing code in the wiki, you can notate or comment right in-line with the code to explain why you built something a specific way. There have been times where I've forgotten details of my code, so documentation isn't just for everyone else–it helps me be efficient as well.

Other documentation we maintain includes release notes and FAQs for the support team. This helps them understand the expected behavior and how to troubleshoot simple issues that may come up. When an issue happens that our support team can't solve, they escalate it to our development team. I'll walk you through that process more in the next section.