I spent this past week learning the landscape of open source software and contributing to two open source projects with two pull requests. Overall, I feel like I have a solid enough foundation to continue working on open source projects in a technical manner. I am very much looking forward to being a consistent contributor in the future.

### Learnings, challenges, and opportunities in open source

Open source software has been fundamental to creating the digital infrastructure of our age. It's incredible how vast the ecosystem has grown. It has allowed for dramatic decreases in the costs of starting a technology company. Additionally, anyone with an Internet connection can now simply download the open source tools and resources necessary to teach themselves to code. This wasn't the case 15, 20 years ago. If done right, open source is easier to distribute, easier to customize, and highly empowering.

Now, for some challenges. Funding is a continuous, systemic issue. For example, the massive Heartbleed bug affecting OpenSSL, a major open source encryption tool used by two-thirds of all Web servers, was discovered in April 2014. At the time, OpenSSL only had funds to pay one full time developer, and the rest was managed by volunteers. Isn’t that insane? A tool that protects numerous banks, governments, and companies around the world being maintained full time by only one developer. Even after Heartbleed was publicized, OpenSSL only obtained enough funding from donations to hire 4 developers for 3 years. After that, the path for funding is unknown. The model of waiting for donations and/or a major flaw to be found isn't a sustainable or a safe way to prop up open source tools that power and protect the technologies that we use.

The common assumption is that open source projects are well funded when in reality many lack significant resources. This leads to the free rider problem where the vast majority of software products and infrastructure rely on open source, saving those companies millions of dollars in developer costs, yet there isn't the same level of investment back into maintaining and contributing to open source software. These companies are already using these libraries and tools for free, so they see no strong incentive to significantly fund these projects besides for technical recruitment and brand purposes.

Despite the number of websites and guides that I browsed through to help me get started, I found that many of them weren't up to date with postings of the latest issues that I could fix. I ended up searching through many open source projects on Github to find ones that I could complete without having to ramp up significantly. This search process took awhile as well as the tags associated with each issue weren't consistent. For example, the "easy" tag wasn't standardized, so many "easy" or "good-for-beginner" issues were not actually easy. In the future, I will target a specific open source project in a specific space to go in depth on and contribute to. One potential project is Scikit-learn, a popular machine learning library.

A great resource I did find on the technical details of contributing to open source were these series of videos on Egghead. It was a clear step by step explanation to selecting an open source project and making a pull request.

There is a lack of non-technical contributors - developer evangelism, project management, documentation, marketing, etc. Xavier estimates that 99% of open source consists of developers solely focus on coding. However, to create a successful open source project, e.g. React, you need the components that you would normally find in a company. It almost seems like to get started with open source at all, you need to contribute to the code base first because that's usually what you see first. This can be intimidating for people who are non-technical as they browse the READMEs and contributing documents. The guides for people exploring open source all gear towards contributing your first pull request.

Cyberbullying in open source can be an issue. Blake told me a story about how a prominent JavaScript open source contributor publicly and very negatively criticized code that Blake had contributed to a project by posting it on Twitter, which caused Blake to avoid that person and the company associated with him.

## Terminology

Open source software
Free software (distinct from above)
Frameworks
Libraries
Languages
Web and application servers
README
Contributing.md
Clone
Fork
Pull request
Merge
Upstream
Continuous integration
Code coverage
TravisCI
GitUp

## Quick thoughts on improvements for future sprints

- Plan out what I'm going to tackle on the first day Sunday evening and plan each day the evening before. It's important that I try to scope out the topic as quickly as I can to see what I can accomplish on a weekly basis.
- Every day, write a stream of consciousness and quick thoughts about what I learned that day on the topic. That way, I don't have to try to remember everything at the end of the week/2 weeks. It would be a matter of aggregating those thoughts instead.
- Try to start on the technical aspect as early as possible. For this sprint, I didn't do most of the coding until the last day on Friday.
- Document how I specifically spent my time during the sprint. For example, on Monday, I spent 2 hours reading X, Y, Z papers, 3 hours watching X, Y, Z videos, and 3 hours working on code to implement X thing. I didn't document this for open source and so forgot exactly how long I spent on each section of the content. I could put this in the "Week 1" and "Week 2" sections of my framework document or just write it into my calendar
- Try to have one or two people guide you throughout the process like Blake did for my open source sprint.
- Refine the sprint details in the framework document more.