Learning from Mistakes in Open Source Software Course

Olzhas Zhangeldinov

Nazarbayev University, Astana, Kazakhstan
olzhas.zhangeldinov@nu.edu.kz

Abstract. Open Source Software development includes many peculiarities, which may not be apparent at first sight. Committing to Open Source projects may be a difficult task without former knowledge. This paper describes the experience of a student that took a course on Open Source Software. A lot of attention was paid to mistakes that were made during the course, and a reflection on decisions was conducted.

Keywords: Open Source Software · Learning Experience · OSS Challenges.

1 Introduction

Open source projects are widely used in almost every software today since even proprietary products depend on open source compilers, libraries, or tools. A programmers' involvement in the open source community is an important part of their learning. This paper describes my experience gained attending the Open Source Software course taken in the Spring semester of 2020-2021 academic year in Nazarbayev University.

1.1 Course Description

The learning outcomes of the course belong to one of these two parts - theoretical and practical. Theoretical is understanding the concept of open source software and its aspects, while practical is learning how to participate in the development of open source software. The theoretical part included lectures about the history and current state of open source software. It was assessed by quizzes, a midterm exam, and a short paper about a specific topic related to the Open Source Software course. The contribution part implied that students would choose their role and project, commit to the project and submit deliverables about the work done. The deliverables included short weekly reports about commits and pull requests. Besides the weekly reports, the students had to deliver more detailed reports with their reflections about the work done and decisions.
This paper evolved from the final report written for the course. The work includes more details about the course and its contribution part. This part caused

serious struggles for me and other students, and the reasons for this will be discussed in this paper. A thorough self-assessment was made at the end of the course, and the main mistakes were identified. The paper will demonstrate the mistakes and describe lessons learned from these mistakes.

1.2 Background and Expectations

When the course started, my background in software engineering included the development of HTTP servers and web applications. The projects I had built heavily depended on open source packages. Therefore, I was already familiar with the packages' installation procedures and bug reporting. Additionally, I had worked on projects that involved several other people. Hence, I also had the experience of software development in a team before the course started.
Contributing to the open source community attracted me since I started using the open source packages. These packages make a great impact on software engineering, and being a part of such world-changing technologies was a very attractive perspective.
I had the skills needed for the project development, but I was completely new to contributing to the open source community. Therefore, I expected the course to help me understand and apply conventions specific to the development of Open Source projects. Learning these conventions would help me to commit to the projects later, which was a good motivation to take the course.

2 Project Selection

The selection of a project to commit to, despite my expectations, was a very challenging task. I did not complete this task as successfully as I wanted to. The reasons for the bad project choice will be discussed in later parts of the paper.
At the beginning of the university course, the lectures included generic rules about choosing a good project to contribute. They intended to select a project with an easy way to have the contribution accepted. Such projects would allow passing the practical part of the course, where students needed to report on their commits to their selected projects. Lectures informed students about several metrics that would help them to identify a project that needs intense development. The metrics included the number of commits, frequency of closing pull requests, number of contributors, and, finally, maturity of the project.
The project selection process consisted of two stages. For the first selection, we needed to identify five possible options, and for the second, after a week to weigh our decisions, we had to pick one as a final choice. The first decision took into account judgemental factors presented during lectures before the selection. After receiving additional information the following week, we could change our final decision. This two-staged process had a positive impact on the project selection experience. During the period between the stages, I could assess the choices more mindfully and resolve possible hesitations.

2.1 The First Selection

As the first stage of our selection, the students selected five projects as the first choice and four other possible options. I decided that I should include projects with different motivations, sizes, and familiarity. The reason for this was the initial uncertainty about the course. I planned to obtain more information about the project selection from lectures later and was not restrictive to myself in the project selection at the time. However, I tried to pick projects with the best numbers for the metrics given above.

Laravel As the first choice, I picked the project I wanted to commit to the most because I had already been its passive user and planned to contribute there. Thus, I chose the Laravel [3] project, which is a back-end framework for web servers. I was the most familiar with this project amongst other options. Moreover, I was interested in this project the most because of my wide usage of the package in several projects. Four other options were new projects on GitHub that I had not been familiar with before. I chose projects with an increasing level of difficulty. Therefore, I could find a balance between an interest in learning and a possibility of merging my commits.

NextCloud Nextcloud [1] is an application to store and manage files on the cloud. The project was written in already familiar for me PHP [14] and JavaScript [15]. This project would be a good option if I decided to conduct more research with comfortable programming languages.

Graphana As the next level of difficulty, I included Graphana [2]. This project specifies on visualization of large volumes of data. The main focus of this project is a web representation using TypeScript [16]. The server-side computations are made with Golang [17]. I learned TypeScript and had experience with Golang, so I considered this as a good option.

Zola I was eager to use Rust [18] language because I considered it a promising language, which includes many features I liked. However, writing projects in Rust is a challenging task because the language is low-level. Therefore, I chose a small project named Zola [4], which is a static website generator. I had some experience with Webpack, so I found this project not very difficult to understand. On the other side, I would need to learn Rust a lot. Therefore, I needed to consider the drawbacks of this option before choosing.

Nushell Another project written in Rust is Nushell [5]. It is the Linux shell interface with an improved user interface. This project was the most difficult to learn. Not only was I not proficient enough for the Rust language, but also I had never built a command line. I included this in case that there would not be enough things to learn in other projects.

2.2 Final Decision

The course lectures, besides other hints, advised choosing relatively new projects with enough contributors and active daily work. The projects were compared to each other, focusing on these details. The comparative table (Table 1) illustrates the differences between the projects. I tried to pick projects with as many activities as possible. However, projects with high activity are also very mature projects. Lectures warned about low activity in projects with a long lifetime, and firstly I considered Zola and Nushell as good options. Nevertheless, their frequency of closed pull requests was drastically lower than in old projects. For the practical part of the course, students had to write weekly reports about the work done, preferably commit every week, and interact with the community. I was afraid that the regularity of several PRs per week would not guarantee that my requests would be closed before the weekly report submission. Neither did it assure enough communication for the report. Therefore, I ended up excluding relatively new projects from my options.

Table 1. Comparison of open source projects from the first choice.

Project name	Number of commits	closed PR frequency	Maturity	Number of contributors
Laravel	32,000	7 per day	8 years	2600
NextCloud	60,000	9 per day	8 years	785
Graphana	32,000	13 per day	7 years	1590
Zola	1600	1 per week	4 years	265
Nushell	3500	3 per week	2 years	261

The most mature projects left were Laravel, NextCoud, and Graphana. Despite their distinctions in each metric, these projects were quite similar. All the numbers were far more than acceptable for each project. Therefore, such a change in these metrics would not affect a student's experience as a contributor. Finally, I decided to choose Laravel, because I was the most familiar with it. It would be easier for me to find tasks to complete since I was using the framework in real projects, while NextCloud and Graphana would require me to do more research about their usage.

3 Project Description

3.1 Governance

The leadership model of Laravel is "Benevolent Dictator" with Taylor Otwell [6] playing the most influential role in the project development. He decides on what features to include and the overall development path of the project.

There is also a group named The Laravel Framework [7], which includes core developers of the project. They have designated roles and more weight on their claims. Their work is not limited to the framework, and only 5 out of 30 the most active Laravel contributors belong to the group [8].

3.2 Community

The community is structured hierarchically so that the ideas flow from bottom layers (passive, active users) to higher (core developers and then the maintainer). Ideas are usually presented as pull requests because the community promotes the addition of some proof of concept along with the suggestions. The following feed of the pull request tracks discussions of an idea. Therefore, every step of the contribution is performed at the GitHub repository.

3.3 Licence

The project is licensed under MIT license. It is a common choice for vendored packages. The commercial use of the package is allowed, as well as the modification for any purpose, and sharing the package (distribution). These are the most common cases of usage, especially for web frameworks. These allowances let the package spread in the community freely.

4 Technical Aspects

4.1 Architecture

The architecture of the framework is justified by the fact that PHP is an Object-Oriented Language. Thus, the code is split based on classes, interfaces, and traits. The higher-level division is modular. Each module contains independent logic that can be used outside of the framework's context by any software. The modules also contain specific classes consumed by a kernel of the framework (service providers, contracts, console commands). The modules are exposed to a user via Facades for neat usage.

4.2 Related Modules

I have done my work within specific modules and modified one module at a time. The module choice was based on the complexity of the module. I tried to avoid working with modules that implied using many PHP extensions and drivers for other programs. For example, I decided not to modify the Eloquent module, written for building Database queries.
The work I have done is related to several modules, namely Routing, Foundation\Console, and Foundation\Auth.
The Routing module maps HTTP routes that come from a client to specific controllers. The routing logic often uses a "Route" Facade. The facades provide methods for registering route patterns and mapping them to PHP Callables.
The Foundation namespace includes many submodules that have the same names as most of the usual Laravel modules (Auth and Foundation\Auth). These submodules, in contrast to basic modules, do not perform significant actions for a request lifecycle. Instead, they shorten the time and amount of code a developer

needs to perform frequent actions for most back-end servers.
Foundation\Console module contains console commands for automatic configuration of the server. Most of them generate pre-defined stubs of code that contain basic structure of a class, which is consumed by the framework. For instance, it might be a fresh controller or a middleware.
Foundation\Auth module provides some shortcuts for users authentication and resolving their restrictions.

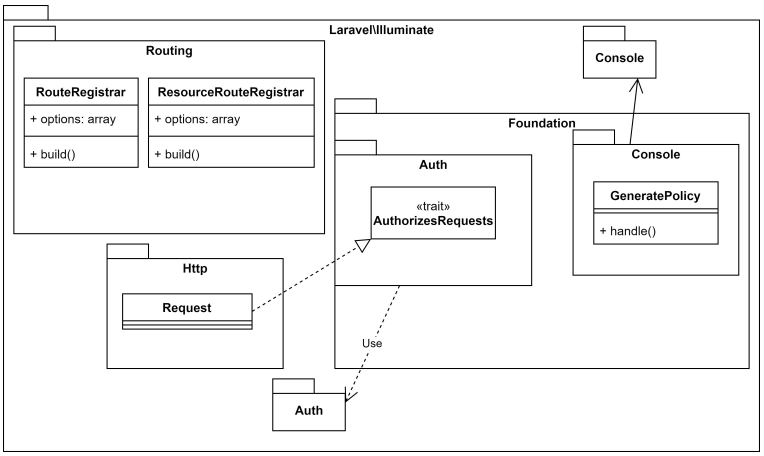

Fig. 1. Package Diagram of Related Modules

4.3 Project Status

At the time of my project selection, the project had entered a soft freeze mode. New features were rare to maintain the code base properly. This circumstance imposed additional challenges for a successful contribution to the project. I missed the project status from my attention during the project selection. Therefore, I was limited in the number of ways in which I could contribute to the project. The soft freeze lasted the whole semester, and I had to do the work in a way that I did not expect initially.
There were not many unsolved bugs either. Because developers focused their attention on them instead of features, bugs did not usually wait for a day before being solved. The persistent bugs were relevant to the interoperation with databases (Redis, PostgreSQL, and Microsoft MySQL). I attempted solving them, but the tasks were too complicated for being completed within one week.

5 Role and Work Done

Initially, I chose a Developer role. After the first pull request, I realized that the project paused accepting new features, and entered a soft-freeze mode. The maintainer declined most of the proposals to save the stability of pre-release versions. Due to this, I started to look for bugs and missing features that intuitively should be present. Therefore, at the end of the semester, I was playing the role of Developer and Tester.

5.1 The First Commit: Minor Feature to Group Routes by a Common Controller Class

For my first commit, I have implemented a minor feature. The feature was an additional option for grouping routes in the Routing module. The option lets a user to group routes by a common Controller class used. This solution was inspired by Ruby on Rails because I found it helpful in some cases. I forked the project, made a new branch, committed my changes to the branch, and posted a pull request [9]. The work was minor and consisted of 5 code lines. The maintainer declined the pull request because it was not important enough for the soft-freeze status of the project.

5.2 Testing Modules and Bug Fix

Then I decided to switch my priorities in the development from generating new ideas to testing the existing features. However, since the project has a large community and the last version was in the soft freeze for a while, major bugs were already fixed, and only bugs in specific cases could occur. I started with testing different modules. I decided to test the Policy feature because it was one of my favorite features. I found a bug in the code that generates a stub for a new policy. The code is in the Foundation\Console module. Because of the bug, duplicated lines in the stub did not collapse because regular expression did not match Windows' carriage return \r symbol in the line-feed. I attempted to make the matching of the line-feed platform-independent [10]. However, another contributor pointed out that my solution would break the behavior in projects that import Laravel package with a git *core.autocrlf* option set to *false*. I tested his proposed solution, copied the code, and made a pull request that was merged in a few hours [11].

5.3 Adding Compatibility between Old and New Features

I realized that there were not many known bugs left in the project. I scanned the recently merged pull request to find possible bugs there. I could not spot any, but I found a recently added Routing method *missing*. The function allows adding a custom callback, which is called when a model bound to the route is not found in the database (e.g. users/1 - binds User with id = 1). When I was

testing it, I decided that it should be compatible with *resource* method added a long time ago. The *resource* method automatically defines a set of CRUD routes for a model. I implemented a wrapper for the *missing* method used during the building of *resource* routes. The decision was not a mistake, and the pull request with this modification was merged in the latest version branch and released in the next minor release [12].

5.4 Interaction with Community

There was little interaction with the community during my contribution process. My first PR was declined by the maintainer a few hours after I posted it. In this timeframe, only one contributor asked me a question about route caching. I checked the functionality when they pointed it out and reported that the caching worked as expected.
The feature also became merged very fast. There was not any following discussion.
Most interactions with the community were during the submitting the bug fix. However, this was not a full-fledged discussion since the contributor only provided a better code than me.
Therefore, a fast activity in the project might imply that there will not be a space for communication. Probably, picking slower-paced development will result in a more quality experience received from contributing to open source.

6 Lessons Learned

There are some lessons about Open Source Software that I took from contributing to the project. The contribution to the Open Source has many aspects. They must be taken into account when trying to help to develop a project. The lessons fall into several categories that embrace a specific detail of interacting with the Open Source community. Most of the new information was from the mistakes that I made because of a lack of experience. These lessons might guide new students of the course to proper decisions. The decision might help them pass the practical part of the course.

6.1 Identifying a Good Project to Contribute

One of the main mistakes was choosing a project without a need for active help. I expected that I would commit changes that I needed personally for my projects. My thought was that they would be helpful for others. Adding new features for users sounded interesting to me. Especially when considering such a large project as Laravel, thousands of projects depend on it. Laravel is developed very fast and releases new major versions every several months. However, I was wrong when I supposed that it would not be a problem to propose new features to the project. I analyzed the number of contributors and the frequency of pull requests.
However, these factors were not enough for selecting the project in my case.

Despite the high frequency of merges and many contributors, committing to the project was a big challenge. I could observe the reasons could even before I selected the project. However, I have learned it too late and decided not to switch the project in the middle of the semester. My main mistake was the lack of attention to specific details about the project.
The first and the most significant detail is the status of the project. Since I started committing to the project in a soft-freeze mode, I had few chances of merging and being helpful as a developer. The status of the project plays a big role in choosing a project for contribution. The main reason is that if the project does not accept any major changes, you will not be helpful as a developer.
The status of the project can be identified by recently closed pull requests. Although I looked at the list of the pull request, I did not pay attention to its content every time. However, those PRs contained important information that can inform a reader about the project's status. Many of the pull requests that were proposed in the last month, were typically closed with the reply from Taylor Otwell that proposals of new features are not accepted because they want to save a good code maintainability [13].
With this experience of choosing the wrong project to contribute, I learned a lesson to investigate deeper about whether the project needs my help in the specific role or not.

6.2 Writing Pull Requests

After writing and reading some pull requests, I understood that it would be better if your pull requests obeyed some basic rules that would help other people in the community. One of the most important rules is the proportion of the number of words in PR and the amount of changed code.
My first pull request contained a lot of words and redundant examples, although the commit changed a very small portion of the source code. When I read other pull requests and compared them to mine, I understood that it is very convenient when you open a PR and can easily tell how many changes commits in the pull request brought. It might be too troublesome for developers to open the *Changes* section and scan all the code to get an estimate of the work done. Thus, it is better to follow a good proportion between PR length and the number of changes.
The proportion is specific for every community. Some projects encourage developers to write long PRs with all descriptions and comments, while Laravel usually follows the format of a short message, without many details and examples.

6.3 Finding Tasks

Since the project did not accept most of the changes, my main work was to search for any tasks that I could do. I was not an experienced developer in the project, thus I did not understand complex parts of the code. Therefore, I needed to find something simple and useful.

After some weeks of trying, I understood the concept that should assist in finding the tasks by yourself. Since no major changes were accepted, only minor fixes and additions should be considered. However, if there is no repository of pending tasks that should be completed, it might be even harder to understand what a project needs.
However, I managed to bring a commit that was merged to the main branch. The commit included the extension of a relatively new feature to an old method. In other words, it made an intuitive connection between two features. I learned from this that projects may often lack the intuitive connection between classes. Commits that allow several features to work together are a good contribution that does not conflict with project modes limiting the number of new features.
Therefore, I learned to adapt to the workflow that takes place in the project. I think contributors should follow some general trends in project development. This will make their commits more valuable, and get more chances to be merged.

7 Conclusion

Although I used to develop the project with help of the Open Source Software course, I had not contributed to any of them before I took the course. The course became a good motivation for taking the first steps in developing Open Source community.
I chose a project that I felt attracted to and started doing commits, hopefully being useful for the project. Some mistakes were made, which led to several difficulties with deciding further actions. Nevertheless, it is important to adapt to the environment and identify the feature that the project needed even in the soft-freeze mode. Thus, the mistakes that were made during the course became a good learning material for further contributions to Open Source Software.

References

1. Nextcloud Repository, https://github.com/nextcloud/server. Last accessed 5 Oct 2021
2. Graphana Repository, https://github.com/grafana/grafana. Last accessed 5 Oct 2021
3. Laravel Repository, https://github.com/laravel/framework. Last accessed 5 Oct 2021
4. Zola Repository, https://github.com/getzola/zola. Last accessed 5 Oct 2021
5. Nushell Repository, https://github.com/nushell/nushell. Last accessed 5 Oct 2021
6. Taylor Otwell GitHub page, https://github.com/taylorotwell. Last accessed 5 Oct 2021
7. Taylor Otwell GitHub page, https://github.com/laravel. Last accessed 5 Oct 2021
8. Laravel Contributors page, https://github.com/laravel/framework/graphs/contributors. Last accessed 15 Apr 2021
9. [8.x] Route group for same controller, https://github.com/laravel/framework/pull/36213. Last accessed 5 Oct 2021

10. [8.x] Make user policy command fix (Windows), https://github.com/laravel/framework/pull/36445. Last accessed 5 Oct 2021
11. [8.x] Make user policy command fix (Windows) - fixed version, https://github.com/laravel/framework/pull/36464. Last accessed 5 Oct 2021
12. [8.x] Add resource missing option, https://github.com/laravel/framework/pull/36562. Last accessed 5 Oct 2021
13. Response about soft-freeze, https://github.com/laravel/framework/pull/36213#issuecomment-776738560. Last accessed 5 Oct 2021
14. Bakken, S.S., Suraski Z, Schmid E.: PHP Manual: Volume 1. iUniverse, Incorporated (2000)
15. Flanagan D.: JavaScript: the definitive guide. O'Reilly Media, Inc. (2006)
16. Bierman, G., Abadi, M., Torgersen, M.: Understanding typescript. In: European Conference on Object-Oriented Programming, pp. 257—281. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44202-9_11
17. Meyerson, J.: The Go Programming Language. In: IEEE Software, vol. 31, no. 5, pp. 104–104, Sept.-Oct. (2014). https://doi.org/10.1109/MS.2014.127
18. Matsakis, N.D., Klock II, F.S.: The rust language. In: ACM SIGAda Ada Letters, vol. 4, no. 3, pp. 103-–104, (2014)