A Project Evaluation Activity

1 Summary

Given that one of your major objectives is to make a meaningful contribution to a free and open source software (*FOSS*) project, it is crucial that you find a project for which this is possible in the time frame that you have allocated for this task. Your objective might even be more restrictive if you want to make a contribution to a "humanitarian" FOSS project, called an *HFOSS* project, of which there are much fewer from which to choose. The number of FOSS projects is so large and varied in terms of their approachability, size, programming language base, level of activity, complexity of code, community friendliness, domain knowledge prerequisites, and complexity, that you will need some means other than an exhaustive search for choosing projects that are the optimal choices for you. This implies that

1. you have identified realistic goals for what you would like to achieve in a semester-long class in open source software development, possibly with a plan to continue after the class ends,
2. you have identified your skill set and your interests,
3. you have created a set of criteria for evaluating projects to which you would like to contribute, and
4. you have a decision-making procedure for determining how well a project matches your requirements.

The objective of the project evaluation activity that accompanies these notes is to give you some experience in selecting projects in this way, using a combination of qualitative and quantitative measures.

2 Background

Though open source software existed before the *World-Wide-Web* was created, the Web and Internet connectivity have been essential for the rapid expansion of FOSS in recent years. FOSS projects need to be available on the Web in order for anybody in the world to be able to contribute to them. There are several websites that provide a home and visibility for FOSS projects (although many of the biggest projects are hosted on their own servers and websites). Some of these websites are ***source forges***, such as *SourceForge*, and some are ***full-featured development environments***, such as *GitHub* and *GitLab*. In addition, there are online directories of open source projects, such as *OpenHub*, which provide statistics about projects, such as the number of contributors, software metrics, and commit activity.

In short, different projects are hosted on different platforms. As of August 2019, *GitHub* hosted about 100 million repositories, with more than 31 million users and contributors[1] and *SourceForge* hosted about 430,000 projects with 3.7 million registered users[2]. *GitLab* is much smaller, with about 2000 contributors and 550,000 projects[3]. *Wikipedia* has a comparison of source code hosting sites at

https://en.wikipedia.org/wiki/Comparison_of_source_code_hosting_facilities.

The point is that there are many different places to look for projects. In this activity, we will apply criteria for project evaluation to a few different projects.

2.1 Where to Look

Sometimes you need to look for the answers to the questions in the project's code repository, and sometimes you need to start on the project's home page. In general it is a good idea to start on the home page, to get a sense of what the project is about and learn as much background as you can, before diving into the details of its code repository.

[1] Source: https://github.com/about
[2] Source: https://sourceforge.net/
[3] Source:https://en.wikipedia.org/wiki/Comparison_of_source_code_hosting_facilities

3 The Anatomy of an Open Source Project

Before we can evaluate an open source project, we have to know its anatomy, i.e., its components, its organization, its structure, and how they are interrelated.

3.1 The People

Who are the people in the project and what roles do they play?

- **Author(s)**: These are the people or the organization that created the project.
- **Owner(s)/Leaders**: These are the people who have administrative ownership over the organization or repository, which might be a board and may or may not include the original author.
- **Maintainers (Core Contributors)**: These are people who are responsible for driving the vision and managing the organizational aspects of the project. They are the ones who accept pull requests and manage the code base.
- **Contributors**: This is everyone who has contributed something back to the project.
- **Community Members**: These are the people who use the project. They might be active in conversations or express their opinion on the project's direction.

In general, you can find who is involved with a project on its website, in a "team" or "about us" page, or in the repository for governance documentation.

3.2 The Important Documents

Aside from the code itself, every project has a set of documents, usually found with the code, that are important for anyone planning to use the project, to contribute to it, or to modify it and redistribute it.

- **LICENSE**: Every open source project must have an open source license, otherwise it is not open source. If the project does not have a license, it is not open source. The content of the license determines what can be done with the code.
- **README**: The README is the gateway to the guidance and help for the project. It is the very first document that anyone should read. This is why it is named README. It welcomes new community members to the project and it explains the project's purpose and what someone should do to get started using or working on the project.
- **CONTRIBUTING**: A contributing document is intended only for those people who intend to contribute to the project. It explains what types of contributions are needed, what protocol is used for contributing, what form contributions should have, and in general, how the contribution process works. It might include links to coding style guidelines for example or other documentation that a contributor needs to read before working on the project. If a project does not have a CONTRIBUTING file, it is a sign that the project is not so welcoming.
- **CODE_OF_CONDUCT**: The code of conduct sets down the rules for the behavior of the community and helps to encourage a friendly, welcoming environment. Not all projects have a CODE_OF_CONDUCT file, but those that do are projects that recognize that this is an important part of the project and they are usually welcoming projects to which you can contribute.

In addition to these core documents, projects might have tutorials and governance documents.

3.3 The Tools

Every open source project uses a variety of tools. These include the tools for tracking issues, for managing pull requests, and so on. In general, a project's tools include

- **Issue Tracker**: This is a tool that keeps track of all reported issues, including their descriptions, dates when they were first reported, whether someone has been assigned to work on them, what their status is, and so on.
- **Pull Request Manager**: This is similar to an issue tracker but it keeps track of pull requests and their status.
- **Code Hosting Site**: This is the website that hosts the code. Its structure and design greatly influence how a contributor interacts with the project and works on it. Some projects host their code on their own servers and have a custom interface, whereas others may use a hosting site such as *GitHub*.
- **Programming Language(s)**: The set of languages used to write the code of the project.
- **Communication Channels**: These might be chat channels (such as Slack or IRC) for casual conversation, collaboration, and quick exchanges.
- **Discussion Forums/Mailing Lists**: Some projects may use these for conversational topics such as how one solves a problem, rather than using bug reports or feature requests.
- **Development Environment**: This is the set of tools needed to work on the project and install it. Very often one may need to download libraries and development tools to work on the project code.

4 Project Evaluation Criteria

Let us use the term "***appropriateness***" to mean how well a project fits your goal of being able to contribute to it meaningfully in the time frame of a couple of months. The appropriateness of a project is determined, for the most part, by the answers to a set of important questions. Some of these questions are more important than others in terms of whether the project is a good choice or not. Some researchers in this field have even suggested that the questions that should be asked fall into three categories related to the project: *viability*, *approachability*, and *suitability* [1], and that some should be tagged as *critical* whereas others are not as critical and are tagged as *secondary*. I will not categorize them as such. I view this as an optimization problem, in that different projects will satisfy some criteria more than others, so that there is no single metric that can be used to assess projects. It will reduce to a question of which criteria become more important in the end.

The general types of questions that you need to answer about a project under consideration follow. ***These are not the specific questions that you should answer as part of this activity; those will follow***.

- Is it open source? This is obviously the first question to answer. A negative answer for this one stops further evaluation.
- What is its license? Does the license allow for forks and modifications?
- How active is this project? Are people actively submitting issues and are people closing issues in a timely manner?
- How welcoming does the community seem? Are people friendly in the issue discussions, the discussion forum, and chat?
- How easy is it to find information about contributing to the project? Are there clear guides and documentation that can help someone who wants to contribute? Are there written guides about rules of conduct, for example? In short, is it an inviting project?
- What programming languages are used in the project?
- What is the development environment, and how hard is it to download and install it?
- How hard is it to understand the project code? Is it a large code base?
- Does the project depend on external additional software modules such as database or graphics libraries?

- How much does one need to know about the domain of the application in order to understand the code? For example, if it is a health-related application, how much medical or biological background would a contributor need?
- How complicated is the code and how large is it?
- How mature is the project? Is it very new, or has it been around for a long time? Does it have a stable release yet?
- Is the project interesting and/or exciting to you?

It is unlikely that you will find a project for which the answers to all questions are the "good" ones. You will probably have to make some hard decisions. But

- if the community is not friendly,
- if finding the answers to questions is hard,
- if the project is not very active,
- if the code is hard to understand, or
- if it is very difficult to download and install the development environment,

then it is not a good first project.

So how do you answer these questions? How can you evaluate a project based on these criteria? The next section gives you specific questions for which you should find the answers for any project.

5 Specific Questions

The material in this section is partly based on the GitHub Open Source Guide and is used under the CC-BY-4.0 license. These are the specific questions that you should try to answer for each project that you are asked to evaluate. For some, finding the answer is hard. On GitHub, the ***Insights*** tab is very useful.

5.1 Finding the Project License

1. What is the project's license?

   There might be more than one. If so, what are some of the licenses that you found? On most source forges there will be a file named `LICENSE` or something similar in the root level of the repository. Find the license and if it is something you do not understand, try looking for a simple summary of what it allows and does not allow.

5.2 Assessing the Code Base

From a technical perspective, it is important that you feel confident and comfortable in using the tools.

2. What programming languages are used in the project? What is the major language?
3. What is the development environment, and how hard is it to download and install it? Is it easy to find information about this?
4. Does the project depend on external additional software modules such as database or graphics libraries?
5. How complicated is the code and how large is it (you might have to look elsewhere for this)?
6. How mature is the project? Is it very new, or has it been around for a long time? Does it have a stable release yet?

5.3 Assessing the Activity Level

Find the issue tracker, the pull request tracker, and if there are discussion boards or wikis, find them as well. The kinds of things you need to know about a project include:

- the length of time issues stay open (responsiveness)
- the length of time pull requests remain unaccepted
- the frequency of issues reported
- the frequency of pull requests
- the trajectory over time of the above parameters; for example, are issues being reported less often now than they were a few years ago?

Specific questions to ask are

7. How recent are the last ten commits: hours, days, weeks, months?
8. How many people are maintaining the project?

   This can be hard to find. You have to look at the conversations or who is doing the merges.
9. How many contributors has the project had in the past year, roughly?
10. How frequently do people commit: hours, days, weeks, etc?

On *GitHub* you can find this information in the repository's root directory. For example, you can find commit activity by clicking "`Commits`" in the top bar. Look at the `Insights` tab. For projects that have their own sites, you have to find the repository and dig around. You can also look on *OpenHub* for this information.

11. How many issues are currently open?
12. How long do issues stay open?

    Look at the list of closed issues. Take the twenty most recently closed issues and look at when each was first reported. Compute the number of days that each was open and take the average.
13. Is there active discussion on the issues?

    In GitHub, in the rightmost column in the Issues view, you can see how many comments were made per issue. Browse them and report a range of comments, like five to ten.
14. Are issues tagged as easy, hard, etc.? Are they triaged?
15. How many issues were closed in the past six months?

On *GitHub*, a project might have an "`Issues`" tab and you can view them. You can click the "`closed`" tab on the Issues page to see closed issues, for example. Some projects maintain their own issue trackers and you will have to find them by "following your nose" on their websites (e.g., *Mozilla*). Projects listed on *SourceForge* usually have repositories hosted on some other site, such as *GitHub*, where they will have their issue tracker.

16. How many open pull requests are there?
17. Do maintainers respond quickly to pull requests when they are opened? I.e., How long are these pull requests sitting there un-answered?

    Look at the closed pull requests to see how long they stayed open. Take the ten most recently closed ones and look at when each was first reported. Compute the number of days that each was open and take the average.

18. Is there active discussion on the pull requests?

    Use the same method as you did for the issues.

19. How many pull requests were opened within the past month?

20. How recently were any pull requests accepted and merged?

On *GitHub*, you can click on the "closed" tab on the `Pull Requests` page to see closed pull requests.

5.4 Assessing the Welcomeness

21. Is there a `CONTRIBUTING` document?

    Read the first few paragraphs and see if it is easy to understand.

22. Is there a `CODE OF CONDUCT` document?

    Read the first few paragraphs and see if it is easy to understand.

23. Do the maintainers respond helpfully to questions in issues? Are responses generally constructive?

24. Are people friendly in the issues, discussion forum, and chat?

25. Do pull requests get reviewed quickly?

26. Do maintainers thank people for their contributions?

References

[1] Heidi J.C. Ellis, Michelle Purcell, and Gregory W. Hislop. An approach for evaluating foss projects for student participation. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 415–420, New York, NY, USA, 2012. ACM.

