Project Proposal - SENG 480B

Derrick Chow, Chase Churla, Guillaume Claude, and Leo Guan

October 16th, 2022

Do Small Code Changes Merge Faster?

Team members

The group consists of four team members: Derrick Chow, Chase Churla, Guillaume Claude, and Leo Guan.

Motivation

We are attempting to replicate the 2022 study *Do Small Code Changes Merge Faster? A Multi-Language Empirical Investigation*, which we will refer to as *the study* in this proposal.[1] This study attempts to investigate the factors which influence *time-to-merge*. Interest in this domain is based on the fact that *code velocity* has been identified in previous studies to be a strong indicator of development performance and overall code health. Code velocity can be roughly thought of as the amount of modification a codebase undergoes over a controlled period of time. It has been found that code velocity is largely constrained by *time-to-merge*. Time-to-merge is the amount of time taken between the introduction of a code change in the code review process, and the change being merged into the codebase. The study attempted to determine if there is an optimal size for a code change. The study measures the size of a code change in *Source Lines of Code* (SLOC). Presently, there is little industry standard which determines the size code changes ought to be, some are single lines of code, some are thousands. The study seeks to find if there is a code change size which, given a set of factors, optimizes code velocity.

The study extracts data from Github pull requests, extracting every pull request from ten popular repositories for each of the ten most popular programming languages. The study attempts to answer three related research questions. Of course, the ultimate goal was to determine if the SLOC change size had a significant impact on *time-to-merge.* The study calculated a Spearman Correlation Coefficient Samples were displayed on a scatter plot with SLOC as the *x* axis and time-to-merge on the *y* axis. The data was analyzed for any particular relationship that may have an impact on time-to-merge. The study then uses pull request metadata in order to determine the effects of contextual factors on *time-to-merge*. This includes data based on potential confounding factors, such as day of the week, programming language used, and industry affiliation.

Ultimately the study finds that there does not appear to be a strong correlation between SLOC and time-to-merge.

Our replication of this study will be similar to the original, though we wish to use a different methodology to extract data. Selecting from popular repositories, as done in the study, biases towards projects who have more efficient processes and more contributors. Many of the most popular github repositories also have corporate backing which indelibly improves code velocity. Instead, we will

randomly select from repositories to provide an 'average case' for time to merge, though we still will separate our datasets by programming language.

How will we obtain the data

We intend to use the same dataset that was used by the study, which would be extracting metadata regarding code changes directly from the Github API. The data will have to be sourced and cleaned by us. Thus far, we have run preliminary tests using *cURL* in order to get relevant data. All tests have been successful. Data is received from the GitHub API as a *JSON* object, we intend to turn the data into samples of each pull request, as a single CSV document.

Using the GitHub API we can source the ten most popular repositories for any particular language using a call to https://api.github.com/search/repositories?q=language:'$lang'&stars:>0&sort=forks&per_page=10. Then using the repositories found there we can use https://api.github.com/search/issues?q=repo:owner/repo+is:pr+is:merged to get the data for each particular pull request. This is a rather *raw* way of extracting the data, which is why we will have to invest time structuring and cleaning the dataset we receive. As well, there is an issue with rate limiting while using the Github API. The API allows for 1,000 requests per hour on the public API. Large API requests are paginated, thus a particular repository may require hundreds of requests to collect all of the data. We will have to pull from at least 100 repositories. We will need to pull the data over time rather than all at once, as we would quickly exceed the rate limit.

Tools

In testing, we extracted data from the first thousand pull requests in the python/cpython repository and found that the average amount of data within a pull request is 6KB. The original paper claims that their dataset included roughly 800,000 pull requests. Assuming we acquire 1,000,000 pull requests, this would be approximately 6GB of data. This value is well within the range of our available data resources, of which we have a 500GB SSD. In addition, we will not require all of the data from the pull requests, the vast majority can be removed. Thus 6KB per pull request and 6GB of total data are firm upper bounds.

Because this project relies heavily on the Github API, we intend to use the Python GithubAPI library to fetch the data we need. We have begun investing some time into learning the API.

For data science itself, the team has found that internally there is more experience with using Python than R. As well, Python will already be used for data collection. Using Python for the analysis will simplify the process. R and its tools are also causing compilation errors on certain devices (M1 macs), whereas Python operates without any workarounds.

Looking at the paper's methodology, the data science performed does not appear to be computationally expensive. There are no models to train, or demanding statistical work involving GPUs. Occasionally the amount of analysis computation required means that the data must be run on high power devices with GPUs and fast processors. Yet we are confident that a personal laptop will be enough given the low cost computations involved and the relatively meager amount of data. For instance, the Spearman correlation calculation, used several times in the original paper, is *O(n)*, a calculation linear to our input size.

Research goal or question

The research goal is to replicate the study with a slight modification to the way we collect the data. The researcher's choice to use popular repositories may form a bias in the sample data to favour time to merge as popular repositories have better funding, and support, and are more active. Instead chose to collect data randomly from repositories that contain more than 1000 pull requests to get more of an average case for the analysis. Random selection also has the added benefit of making data collection more accessible, by allowing cumulative data collection. This ensures a wider sampling size to better the statistical power when examining the data in distribution models. The study is to examine whether the factors of pull size and compositions can have an impact on code velocity. To achieve this we ask four research questions:

RQ1: What characterizes pull request size, composition, and time-to-merge?

The purpose of the first question is to get a better insight into the characteristics and distribution of the data. This understanding of the data can have an impact on the following questions.

RQ2: What is the relationship between pull request size and composition to time-to-merge?;

RQ3: Does context influence the relationship of pull request size and composition to time-to-merge?

Research Strategies

This research will consist of multiple steps outlined here:

In order to better develop our research project, we will need to read relevant background papers. This will be done collectively throughout but mainly headed by Derrick. As mentioned before, we will need to collect and clean the data before analysis can be performed. This will consist of collecting source code from the Github API which will be done primarily by Chase. Next, we will need to write analysis code. This will be done in python as mentioned above and be headed by Leo. Running the analysis will then be collectively done by all members, especially Leo and Chase who will have the best understanding of the dataset and analysis code. Once the analysis is done, Guillaume will create the project video or

presentation to present to the class. Finally, writing the paper will be done by the entire group each bringing on their accumulated information regarding the project together to create a summary of the project.

We will use Github as a source to get the data needed to perform our study. We will filter through the data for data out of the scope of the project. The methodology for their study was to use the 100 most popular projects from 10 languages on GitHub. As the report mentions, “large patch sets are difficult to review and require a lot of time to read, thus this may delay the acceptance of the patches, " meaning the larger code would take more time to merge. The different types of code do not change merge time. They create the table and record the factors that influence pull request time-to-merge. Later they use the diffstat tool to analyze the pull request and extract the number of modifications of each type. In order to calculate the time-to-merge, they pull the request attributes available from GitHub API, which is **closed_at** and **merged_at**. The plot demonstrates how time-to-merge is not normally distributed, and the spearman correlation between pull request size and time-to-merge has weak relations.

Potential Limitations

Our main potential limitation is the pull request issue. Unfortunately, GitHub has a limit to the rate at which you can make pull requests. Right now the limit is 1000 requests per hour per repository. This means that we may have less flexibility in choosing a data set and will need to commit to collecting from a data set collectively for a couple days. Additionally, it may be more difficult to collect the desired amount of data from these data sets. Currently, we are developing a script that will automatically perform pull requests on a desired repository. Assuming the script works and all team members can run it, for a couple days, the data collection should be of adequate quality and size.

A secondary potential limitation is our data storage size. At the moment we have one dedicated hard drive of 500 Gb to store our data. If really necessary, more storage could be acquired to store data but we estimate that that will not be needed.

A final potential limitation is time constraint. Since it took our group some time to get started on the low level work for this project, time may be a factor regarding the depth of our analysis. This is depending on how well we can manage the rate limiting issue. As a mitigation, a scope adjustment could help us focus on a simpler research project. That being said, we are now confident that our group is on the right foot to move forward with this project.

Expected Results

We are replicating the study with a different dataset, thus we expect similar results to the study. This means that we are expecting pull request size and its composition to not influence time-to-merge. This should be the case no matter how we partition the data: day of the week the pull request was created, affiliation to industry, and programming language. If the findings of the initial study are found, then we

can confirm this result in an “average case” repository. If we do not find the same results, then we may have found some issues with biassed repository selection in the initial study.

Core References

[1] Kudrjavets, G., Nagappen, N. and Rastogi, A., 2022. *Do small code changes merge faster? A Multi-language empirical investigation*. [ebook] Available at: <https://arxiv.org/abs/2203.05045> [Accessed 14 October 2022].