Life and Death of Great Open Source Projects

An exploratory analysis on the activity patterns of GitHub repositories

Jianchao Yang yang.jianc@husky.neu.edu
Zexi Han hhan.ze@husky.neu.edu

2017-04-28

Contents

1 Introduction

With its unique and user-friendly social coding features such as issues tracking, forks, pull requests, commit comments and wikis, GitHub has deservedly become the most popular source code hosting service in the world. Many open source developers use GitHub not only for source code management, but also to collaborate with fellow developers, share knowledge, or simply showcase their personal work. The vibrant and all-encompassing online community of GitHub makes its data a prime window on the social dynamics of open source development.

This project is inspired by the magnitude and heterogeneity of GitHub's project activity data. Our objective is to identify activity patterns for different types of open source projects and pinpoint indicators and determining factors that are most directly related to these patterns. We will do this by first exploring activity patterns using a Shiny app, then split projects into 4 groups based on codebase size (tiny, small, medium, large), and examine for each group how did their community interest (measured by changes in number of stars), maintainer commitment (number of commits) and community involvement (number of issues and issue comments) unfold over time.

2 Methods

2.1 Sampling and Settings

Among the 57 million repositories GitHub is hosting [1], a large portion of them are forks or small personal projects with almost no outside visibility. In order to make the data relevant and analyzable, we shall pick only repositories of adequate community interests and values. And to make sure we still cover all different kinds of repositories, we are selecting the top 1% most starred original repositories (i.e., those were not forks) of each programming language, using GHTorrnet MySQL database dumps exported on April 1, 2017. Languages with less than 100 repositories were ignored, leaving us with 36,068 repositories in 228 languages.

2.2 Measurements of Repository Activity

There are three major type of repository activities: repository starring, code commits and issues. For issues, there are also issue events and issue comments. Other types of activities also include commit comments, release downloads, milestones, etc. For simplicity, we collected only the major acticities: starring, commit counts, and issues (including issue events and comments).

Most of the activities can be aggregated to weekly counts. By looking at the changes of these counts over time, we get a rough picture of how much effort the maintainers have committed to the project, how fast a project gained interests from the community, and how deep the community was involved. These measurements are the most important activities that can happen for a repository on GitHub.

In addition to weekly counts of various activity events, we have also built two performance metrics for evaluating maitainer commitment.

2.2.1 Maintainer Commitment

Number of commits

A commit is an update to the files in the codebase. Number of weekly commits is a direct way of measuring how much efforts the contributors have put into the repository. It will be optimal if we can distinguish commits from core maintainers and the general public. Since that involves scraping one more dataset and this was not planned when we started the project, we are not going to make such separation of authors.

Issues response time

Issues response time is the average time needed for a repository maitainer to take action on an issue opened by someone else. The faster the response, the more commited the maitainer is.

Issues close time

Issues close time is the duration between when the issues were created and when they were last closed, i.e., how fast an issue gets resolved. The faster the close time, the more responsive the maitainers are.

2.2.2 Community Engagement

Issues count

Issues can be reported by anyone. For open source projects on GitHub, issues are an essential channel where the community communicate with the authors: filing bugs, asking feature requests/suggestions, getting support in using the code, and so on.

Issue comments count

[1] As of data on April 19, 2017

Higher number of comments on a single issue indicates more heated debate, which can be simply because of the complexity of the problem, but may also indicate the popularity of a project.

Pull requests count

Pull requests are contributions from the community. More pull requests means more outside people are involved in the development of the projects, often making them more resilient against faded maitainer interest.

2.2.3 Community Interests

Number of stargazers

GitHub officially admits number of stars as "an approximate level of interest" [2]. We have the possibility to know when a user starred a project. Aggregating a weekly count for the increase in stargazers can be used as an indication of how community interest changes over time.

2.3 Data Collection

The GHTorrent data were used only for generating this initial seed of potentially "great" projects, then all other data were scraped directly from GitHub API. Since repositories change owner and names from time to time, and GitHub sometimes handle such case with redirections, we have inevitably encountered a few 404s as well as scraped some duplicate data. To reduce the impact of repository renaming and ownership transfers to minimal, we used the seed only for scraping repository details, then used the up to date owner logins and repository names from the repository details to scrape all other data. [3]

With data at such a large scale, we put our scraping program on a cloud server (AWS EC2 instance) so it can run overnight. To maximize performance and efficiency, we use MySQL to store the scraped data and are saving data on the fly while scraping. In addition to that, each time a data point was successfully saved, an near empty text file will be created in the file system and it will later be used to skip scraped data in case of servie interruptions and system fault.

At the beginning, we tried to write scraped data into the database on the fly, and didn't find a way to efficiently check whether some data has been scraped or not. Then we decide to srape all data to local files then import then to the database. For the 34,779 repositories that were still alive, we scraped 8.9G of issues data, 4.6G issue events, and 14G issue comments data (more than 30G data in total), which took us about 2 days to scrape and 20 hours to import into MySQL and build the indexes (not including time of debugging).

Out of extensive struggles and testing, we found that the most practical AWS configuration with a bearable speed is to use a c4.large (4 cores CPU, 7.5G RAM) EC2 instance type while scraping and a 100GB provisioned SSD Elastic Block Storage with at least 1,500 iops while importing data.

The scraping process has been redesigned since then–we returned back to write data on the fly, but also used small files to check whether a data point has been scraped or not and would not add database indexes afte all data are scraped.

2.4 Data Exporation

To better understand the data we collected, we created a Shiny dashboard, where a user can conveniently explore the activity timeline of a repository, including how number of issues, stargazers and commits (by different authors) evovled over time.

[2] https://developer.github.com/v3/activity/starring/

[3] The actual ramifications were more nuanced than this. We didn't realize how serious this problem was before we have scraped all data, which means we had to run several SQL queries to identify those who changed names and scrape them again.

If time permits, we may add more features to the dashboard, such as issues timeline: response rate, response time, resolve time, community contribution rate (commits of non-core-maintainers) etc; and aggregated measurements by different repository groups: organization, programming language, lanuage of users, region, repository objective (library, framework, cheatsheat, references) etc.

But with this simple combined activity timeline, we can already see a lot of different patterns for different projects–basically no two projects are the same. Some showed clear regular development cycles (Figure. 1 twbs/bootstrap); some got their initial fame very fast, but also died fast–this mostly happens to "cheat-sheet" type of repositories (Figure. 2 vhf/free-programming-books); some are seeing very steady acticities as community interests keep growing (Figure 3. golang/go).

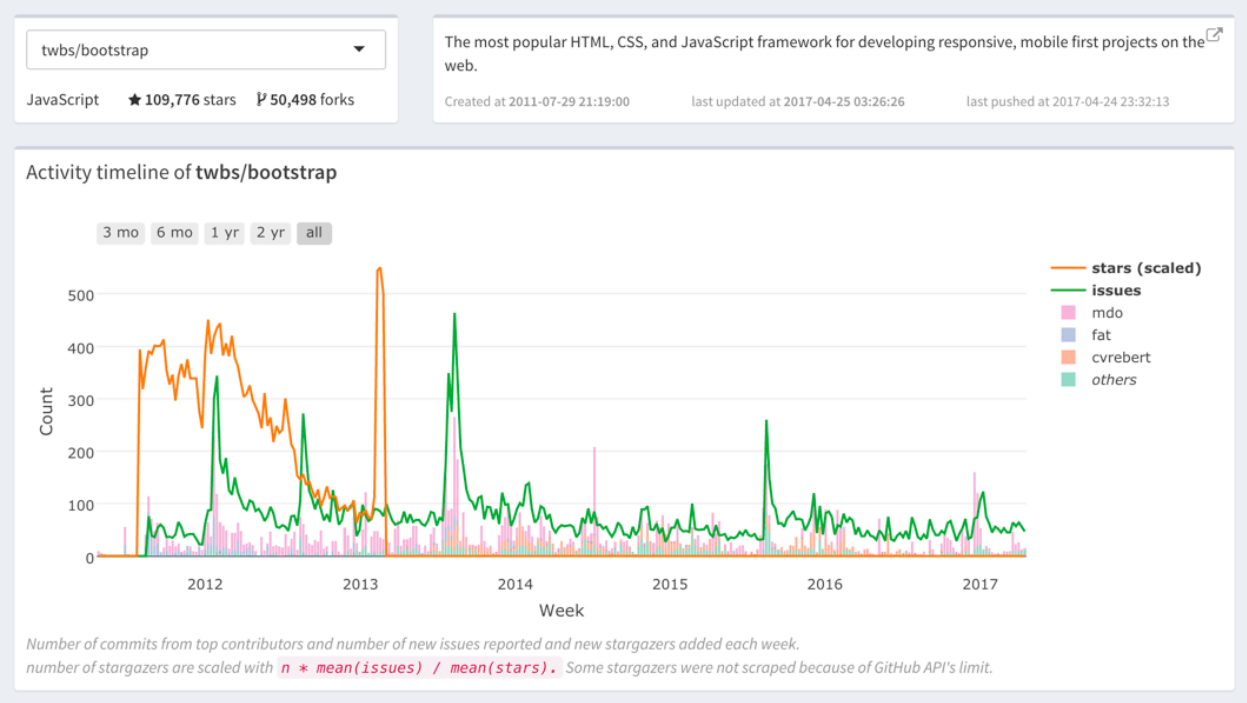


Figure 1: twbs/bootstrap

Our goal would be to detect these patterns and build a typology to describe them.

2.5 Grouping and Classification

To understand the implications of these activity patterns and the correlations between different metrics, we are splitting repositories into groups and comparing how various metrics behave differently over groups.

We sampled 2,500 repositories out of 34,779, and split them into 4 equal-sized groups based on their codebase sizes. This gives us approxiamtely 620 repositories each group, and each across the whole spectrum of programming languages and community interest (stargazers count).

As we are mainly looking at temporal data, it would be more analyzable if every repository had the same time length of data. So the first thing we did is to filter out repositories younger than **one year** and only look at the first year data for each remaining repositories. This time range is long enough for us to discover patterns, but also avoids dropping too many repositories. [4].

[4]One more caveat is that repositories created at different years might have a totally different pattern, as GitHub was getting more and more popular and its user base changes. But for simplicity reason, we are not ignoring such effects for now.

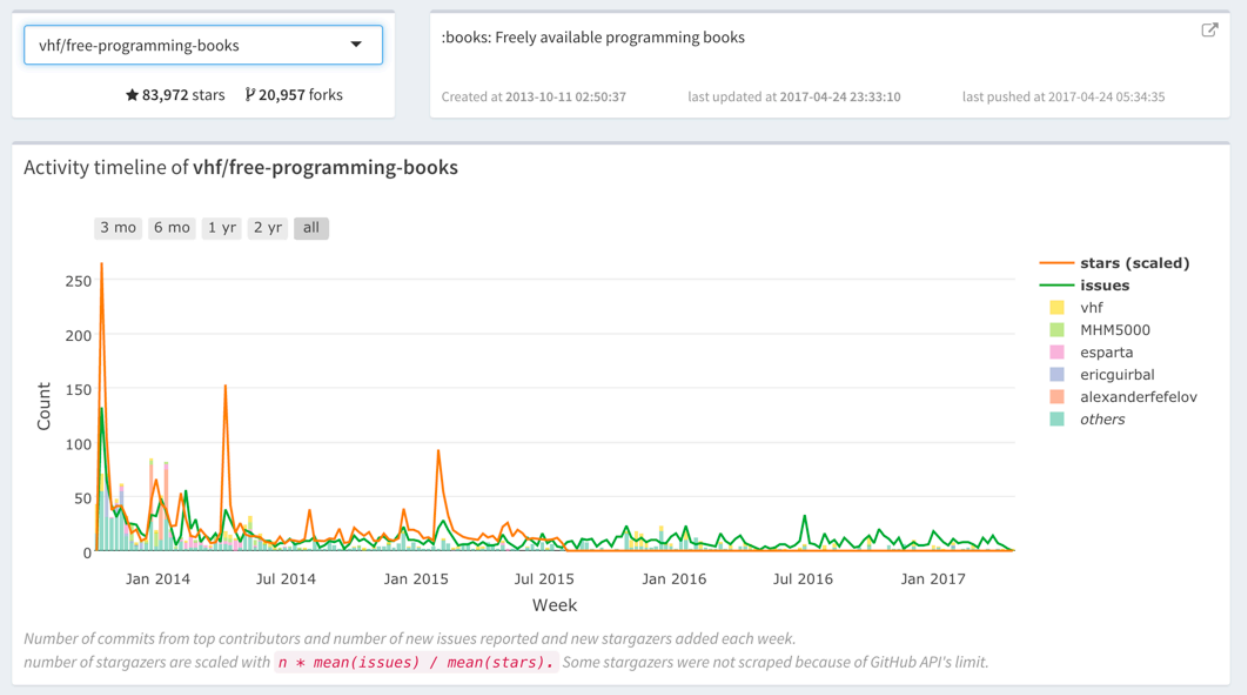


Figure 2: vhf/free-programming-books

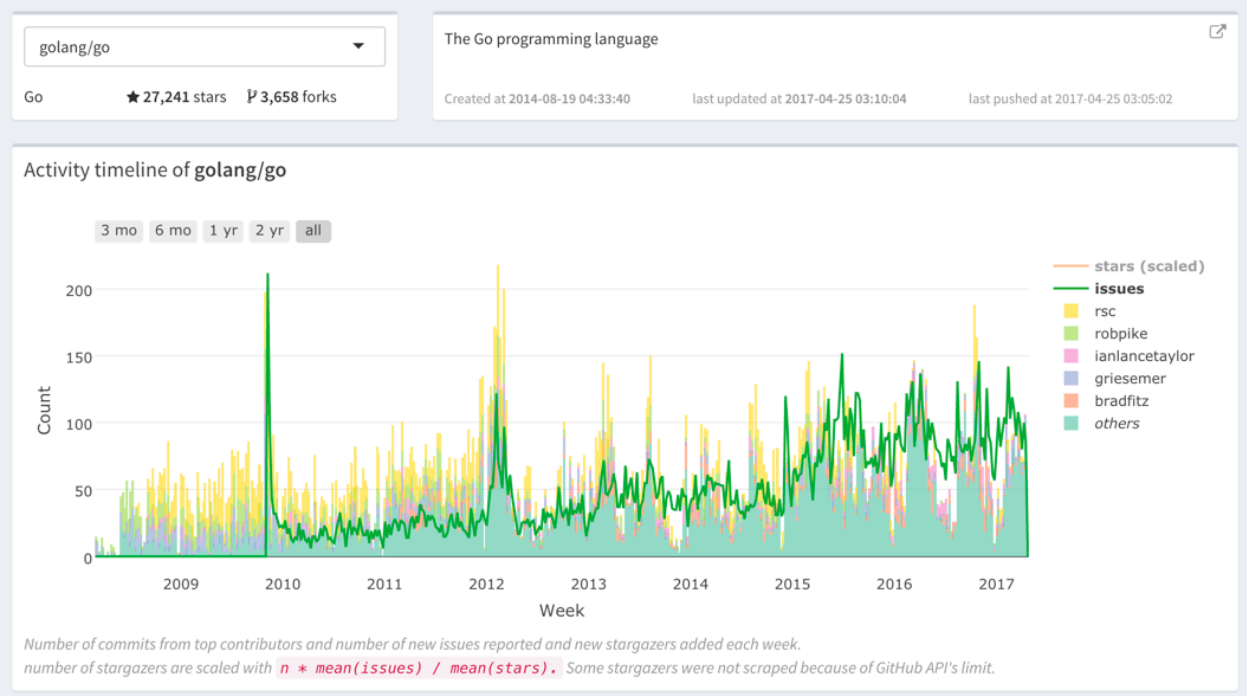


Figure 3: golang/go

3 Data Analysis

3.1 Overview

Among the 2,500 repositories we sampled, nearly half had less then 5MB of codebase.

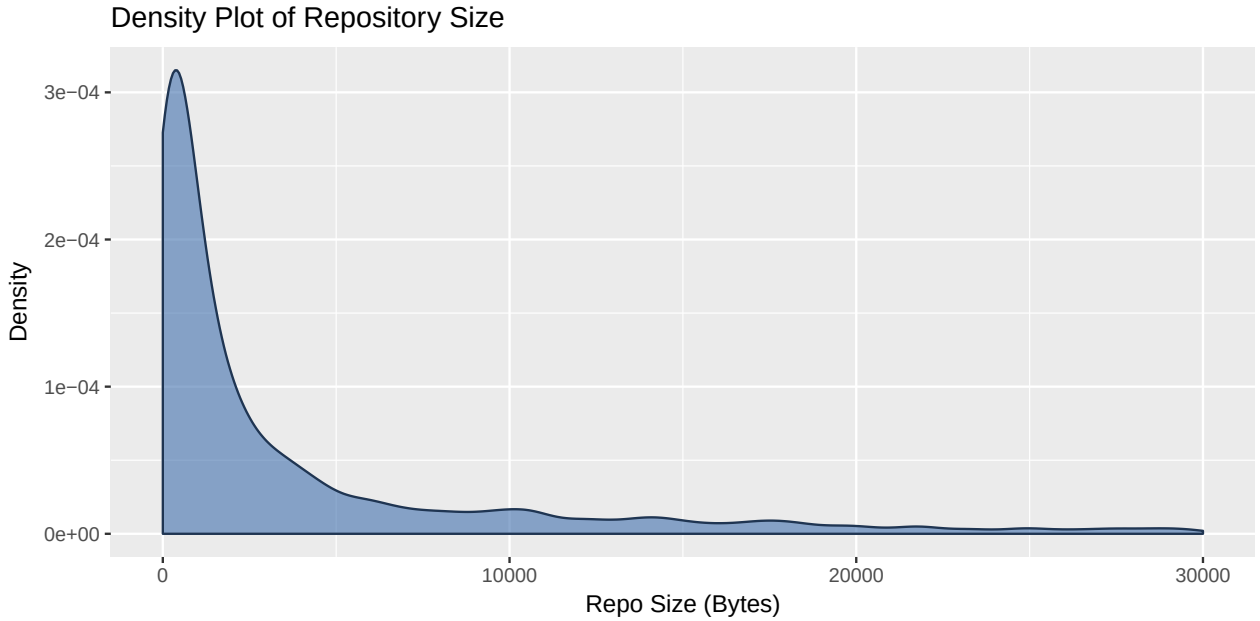
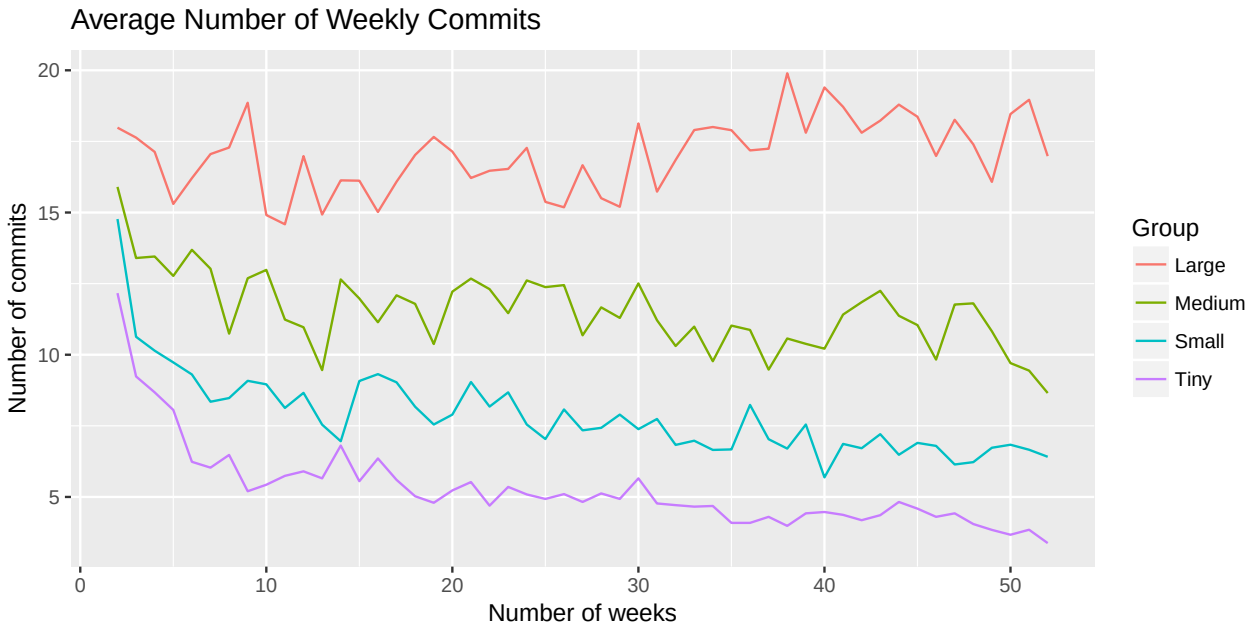


Table 1: Repo size quantiles

0%	25%	50%	75%	100%
80	1523.25	5722.5	26813.25	2492131

3.2 Commits



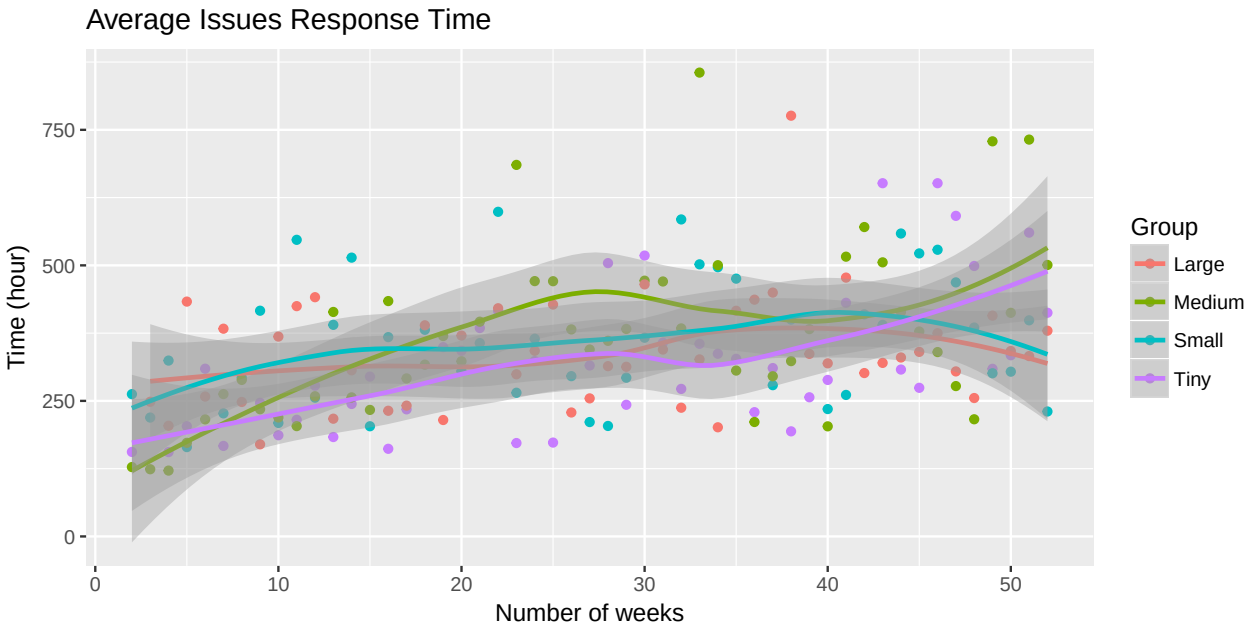
Smaller projects naturally have less commits, but what is more interesting is that almost all groups had a decline in number of commits since the creation, but bigger projects tend to have a more steady stream of code changes, and the number of weekly commits would actually increase as time goes by.

3.3 Issues

3.3.1 Issues response time

Almost for all repositories, issue reponse time increases over time, regardless of the codebase size. One explaination is that maintainers were more active when their projects were first made public, or in the early stage of development, they are using issues to track working progress, which implies more frequent updates to the issues.

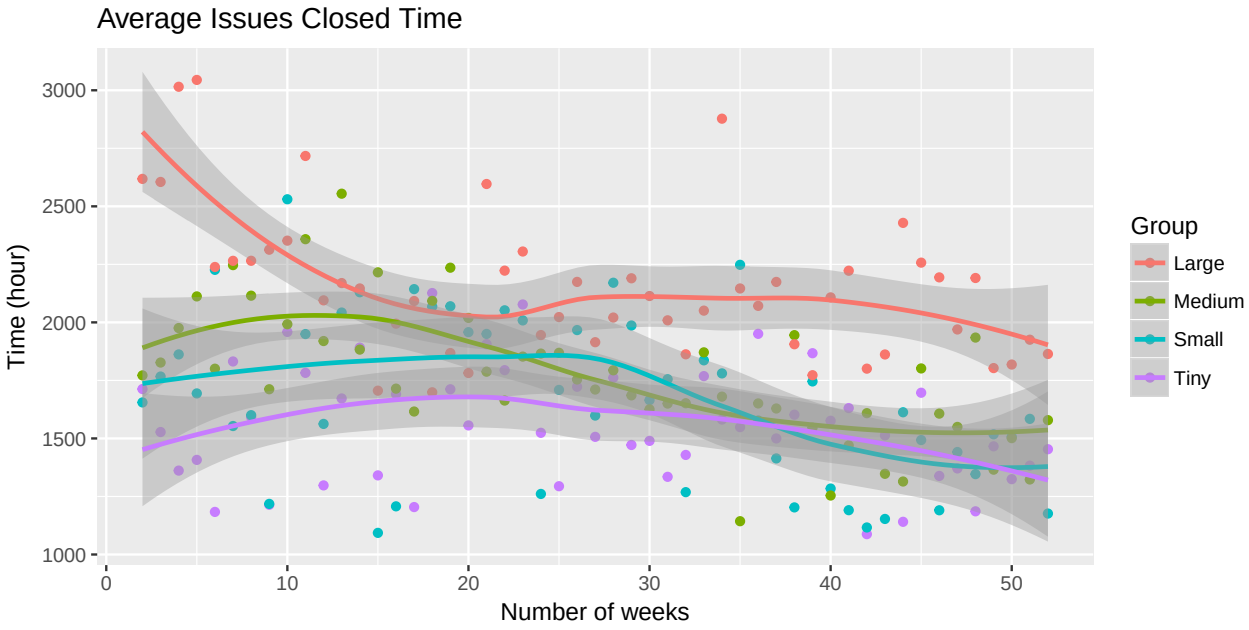
For tiny projects, the increase in reponse time is especially worse, indicating a sense of abandonment. On the contrary, issues response time for large repositories turned more stable in the long run, and even dipped till the end.



3.3.2 Issues close time

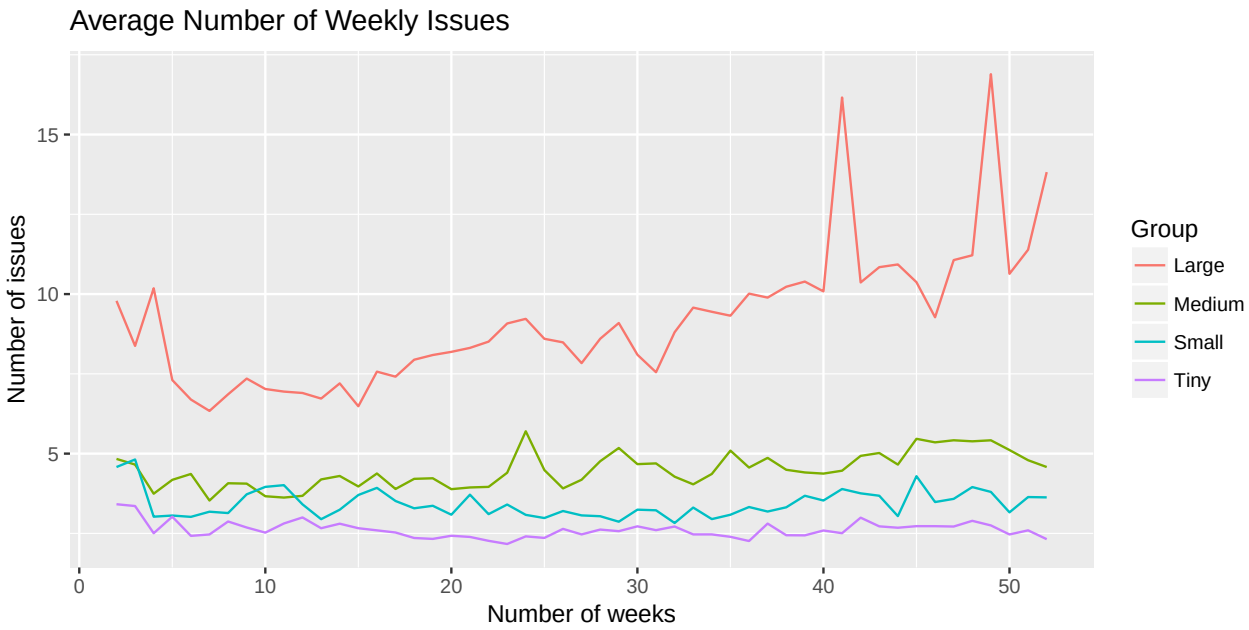
Issue close time have downward trend for most projects, which might be because of different issue types as of different stages a repository is in. At the begining, more feature request type of isssues are created, which in general requires mroe time to resolve; while on later stages, more issues would be bug reports or general questions, which requires relatively little efforts.

In addition, the bigger the codebase size is, the longer it is required to resolve an issue, which might be related to the complexity of the code.



3.3.3 Issues count

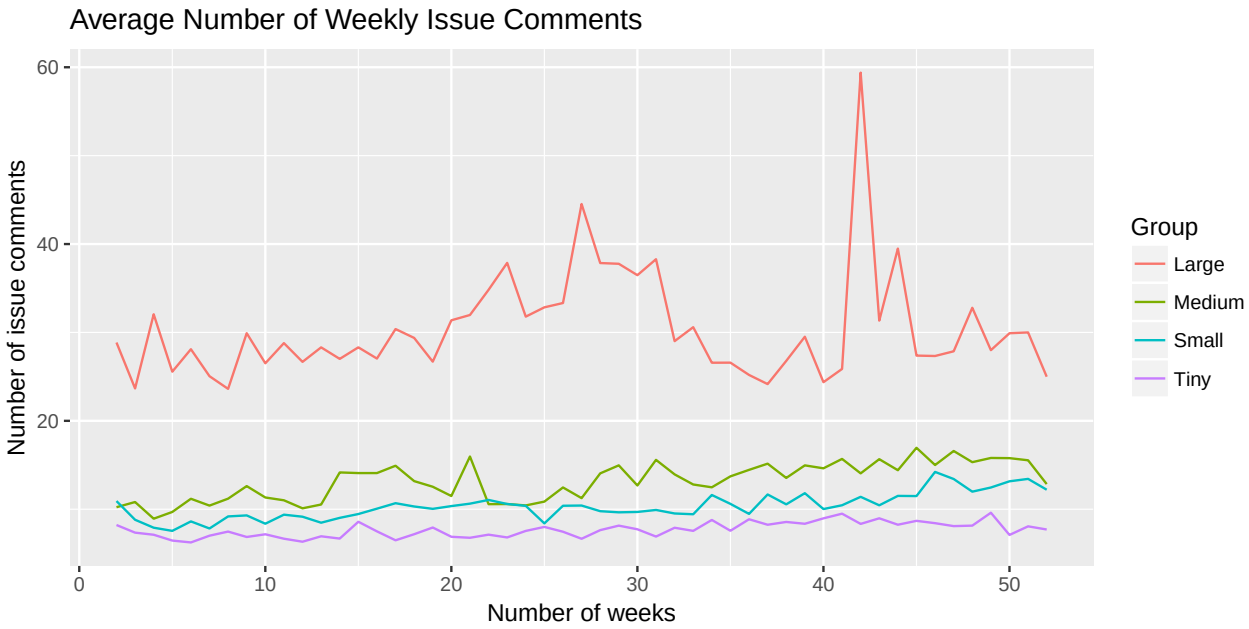
Issues created by non-maintainers indicate involvements of the community, with pull requests especially so.



Large repositories have a very obvious upward trend in terms of new issues created every week, while repositories of other sizes stay flat most of the time. There are two spikes in this graph, but we are not too worry about them since we have a relatively small sample size, and the noise can get very obvious.

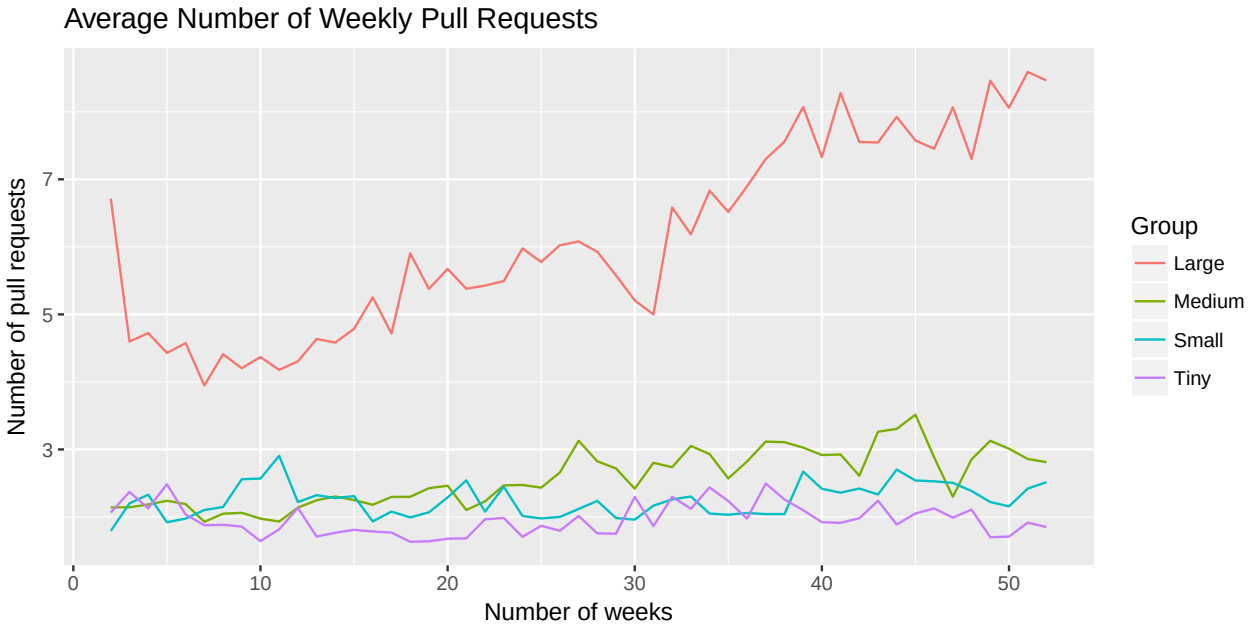
3.3.4 Issue comments

Issue comments are very in line with new issues count. They even showed the same spikes caused by noise.



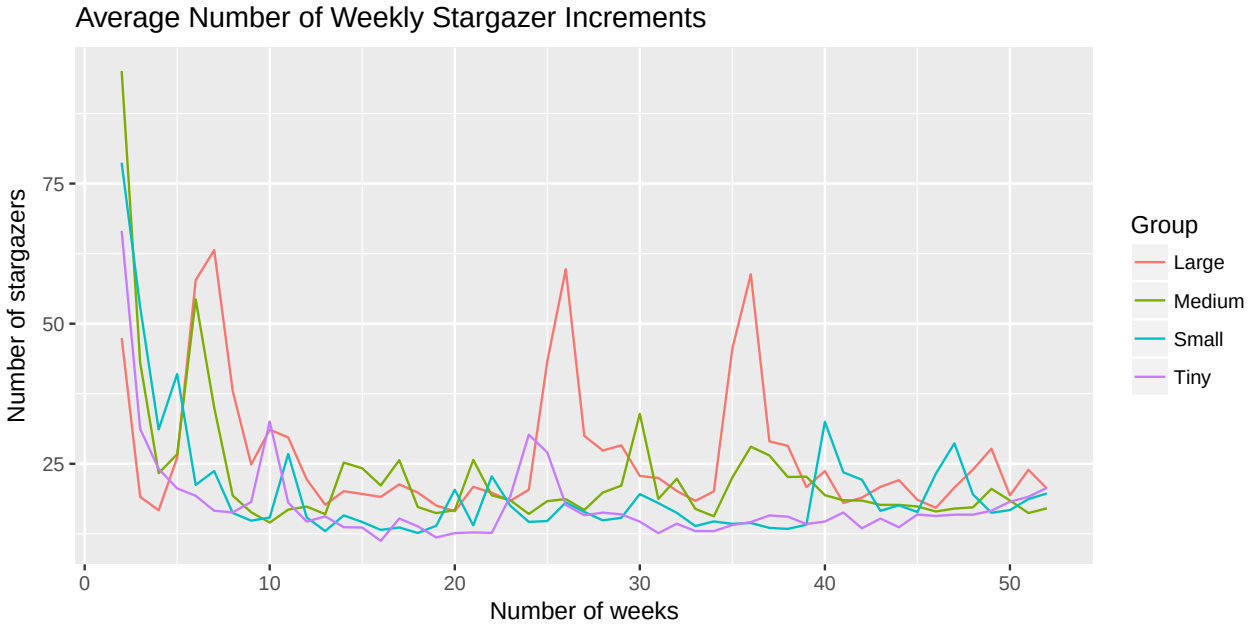
3.3.5 Pull requests

Large repository can attracts more contributions from the community to assist its development. Comparing to general issues, pull requests are showing a more obvious staircase pattern. We might need to investigate why is the case in the future work.



3.4 Stargazers

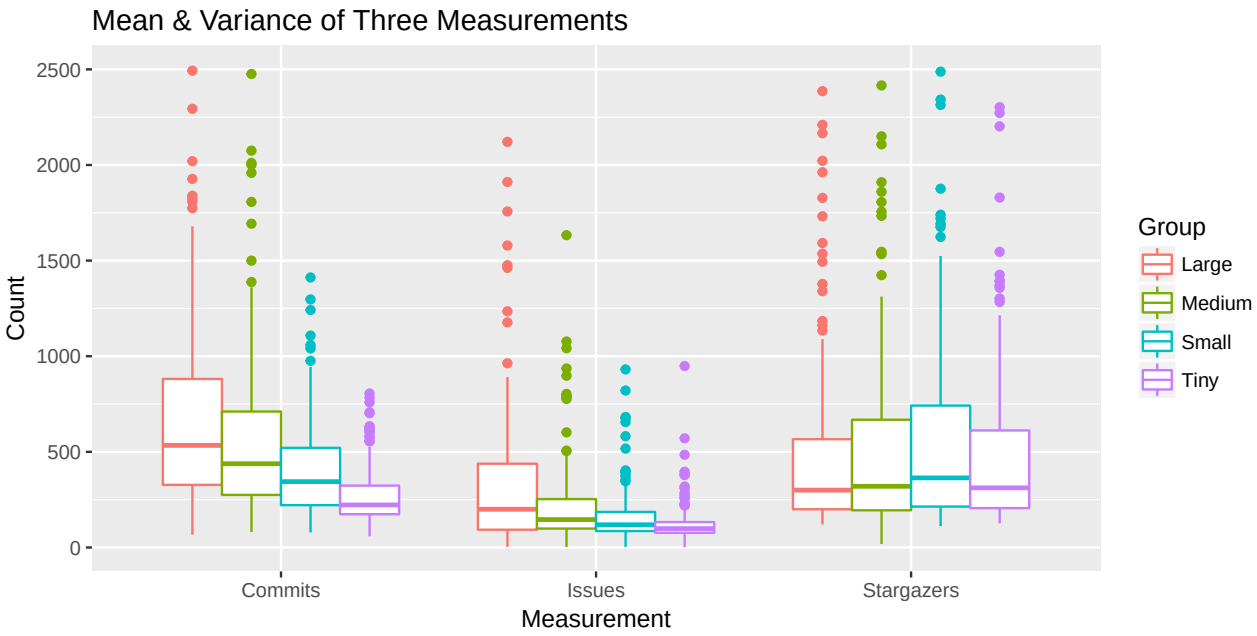
Changes in number of stars are less correlated with repository sizes. It seems that projects tend to gain more attraction when they first released, as all groups showed more stars during the first few weeks.



3.5 Aggregated Measurements

Besides time series exploration, it would be also helpful to compare the total and final state aggregated measurements for each group.

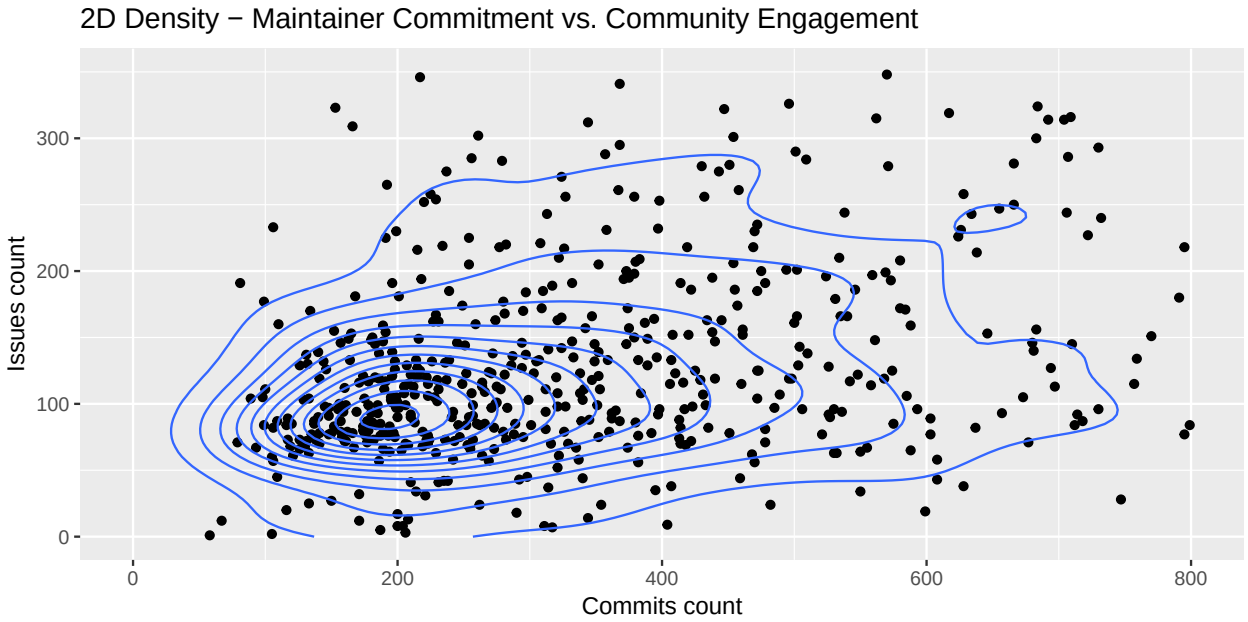
3.5.1 Mean & Variance of Measurements



In the aggregated analysis, we simply select number of commits, issues and stargazers to represent the three measurements. The above box plot clearly shows the mean and variance of three measurements. We dropped many outliers in this plot to zoom in the boxes. For commits and issues, both mean and variance of count increase with the increase of codebase size, which indicats that more variations exist in the development pattern of larger projects. While for stargazers, there is no obvious realation between the number of stargazers and codebase size.

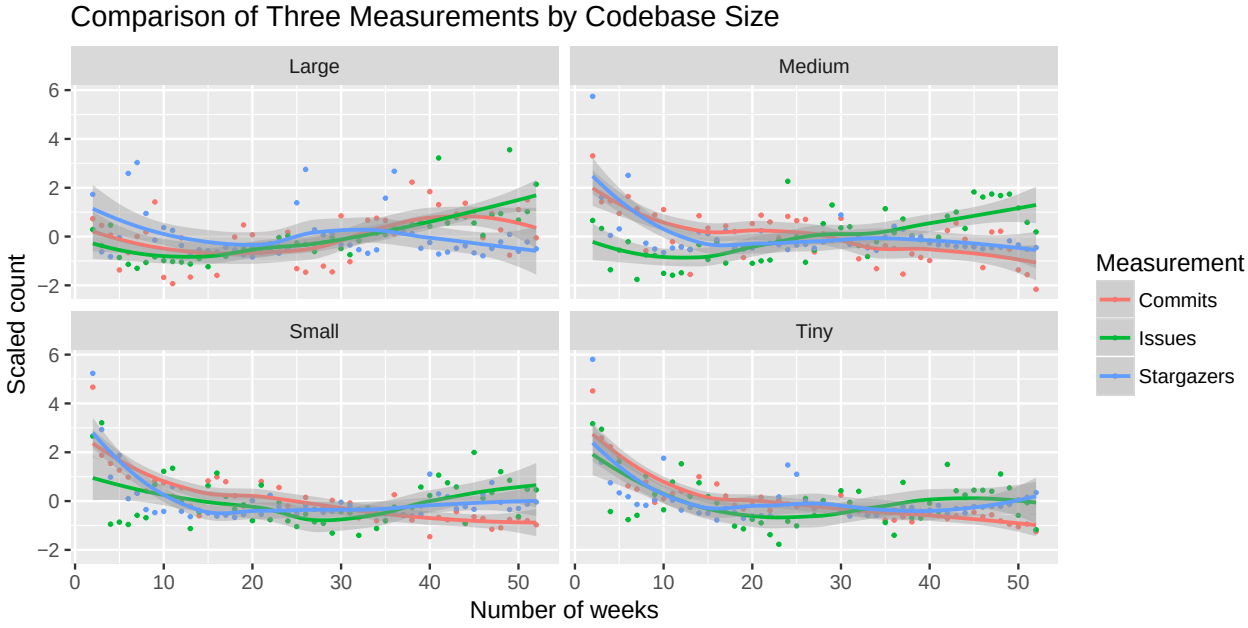
3.5.2 Maintainer Commitment vs. Community Engagement

As mentioned before, commits and issues reflect maitainer commitment and community engagement respectively. We want to examine how these two correlate with each other, using the current total numbers of these two types of events for each repository.



We put them in above 2D density plot. Each data point in the graph is a repository, and x,y axises represent its total commits and issues count in its life. We can see that the ratio of commits count and issues count is approximately 2:1. Besides, most of the repositories clustered at the contour center of 200 commits and 100 issues. In addition, the distance between same-density contour line becomes wider as both the count of commits and issues increase, which cound be interpreted as that the project with more contributions made by its maintainers can attract more supports from the community.

3.5.3 Three measurements in one graph



To better compare the correlated (or not correlated) trends of three different measures, we scaled the data and plotted them in one group. Looking at the trends for different codebase size group, it seems that all groups and metrics presented a pattern of starting at a high volume, decreases for a period, then stablize at some level.

The case for large repositories was more special. The number of commits and issues would actually started growing at some later stages, indicating a continued or even enhanced support from the maintainers and the community.

4 Discussions and next steps

Our main focus for this project was collecting and exploring the implications of the repository activity data on GitHub. The main finding is that no two repositories are the same. Almost all repositories showed different patterns and it is hard to categorically say which pattern group a repository belongs to. Nevertheless, by splitting repositories into groups based on codebase sizes, we were able to identify a few nuances in between different repository types.

Future works may include finding a more robust way to classify the repositories, such as in project objectives and scales. We certainly do not want to compare a pear with an apple. To dive into a subgroup and identify patterns for each group would be more meaningful work to do.

Parallel scraping was made possible with the `future` package. It can be very efficient, yet relatively hard to debug. Storing large scale data is a challenge, especially when writing into MySQL directly via R. Shiny and Plot.ly are great tools for dynamica explorations. Aggregating things in one place is a tremendously helpful way to uncover data insights.

We plan to open source this project and add more features in the summer.

Please refer to the README document on GitHub [5] for how the code are structured and which improvements were planned exactly.

[5] https://github.com/ktmud/github-life