Visualizing Activity of Programming Language Communities on GitHub

Haeyon Kim, Frances Tso, Sharon Yu,
Jake Zimmerman, Puja Jena
February 20, 2017

Goals

Software development is an intensely social activity, though not in the conventional usage of the word (like "social networking sites"). In order to get things done, developers frequently collaborate with others, doing things like reporting bugs, discussing future changes, and reviewing code for quality. A common online platform where software development happens is GitHub. GitHub groups software development into source repositories, or "repos," which then belong to individual users or entire organizations.

For each repository on GitHub, it's possible to identify the primary programming language in use. As with any community, programming language communities develop shared habits and customs, and even sometimes develop associated stereotypes. For example, people sometimes think of JavaScript or Ruby programmers as "hip" and C or C++ programmers as "old-school." These are gross simplifications, but many such characterizations ring surprisingly true.

For our project, we will explore this question:

> **How does social activity on GitHub vary by programming language community?**

We will use usage data from GitHub made available publicly to see what trends, habits, and correlations emerge.

Existing Work

GitHub has been studied extensively in the past. Some aspects of GitHub which have been explored include:

- which projects are the most "popular" or "active," using various metrics for popularity
- which languages are frequently used in the same project
- using "interest graphs" to gauge what kinds of topics people are

generally interested in on GitHub

In addition, many people study GitHub for reasons not related to understanding programming language communities. For example, people frequently analyze GitHub to better understand how to improve software development.

Approach

In our study, we'll be focusing more on the interpersonal and social aspects of GitHub's platform. GitHub has two major features where people interact socially: issues and pull requests. Issues resemble a type of open discussion forum. Pull requests are where individuals propose and discuss code to be added or modified in a project. Both issues and pull requests can be "open" (ongoing) or "closed" (resolved, whatever the reason).

Both of these features have "commenting" in common. For our study, we'll look at the types of discussions that occur on issues and pull requests. Some of the metrics we'll look at include:

- number of unique commenters per project
- number of code contributions from individuals outside a project's main organization
- average length of comment threads per project
- whether a project is characterized by a small number of highly active contributors, or perhaps by a large number of inactive contributors
- whether a project has above average issues open, issues closed, pull requests opened, or pull requests closed.

As we dive into the data, we anticipate finding other interesting metrics on which to compare programming language communities.

Audience

The audience of our analysis is anyone who has a vested interest in fostering positive community experiences on GitHub. This list is wide, but includes groups like the following:

- maintainers of large open source projects, trying to ensure a positive community for a large number of contributors

- hobbyist programmers working on small- to medium-sized projects, delivering a product for a specific group
- people looking for insights into existing social interactions in order to help grow a community they're involved in

From having participated on GitHub already, these groups already have some sense of what social interactions look like. For these people, our studies will confirm or deny their perceptions.

Additionally, our findings will be useful to newcomers to GitHub who are looking to get started in a particular community. Many people land internships through activities on GitHub, so a thorough understanding of how various communities communicate will set newcomers up with an expectation of how various programming language communities currently operate.

Risks

Visualizing data is inherently low-risk; the biggest "risk" is usually just that the dataset used didn't turn up any interesting insights. One way to get around this is to hypothesize what we expect to be the case, and see if the visualization confirms or denies those hypotheses. This way, we don't ever end up in a place where there are "uninteresting" results; instead, we get evidence that backs up a particular claim.

Another aspect where we'll have to take caution is in how we choose to visualize the data. What constitutes a good visualization has been extensively studied. Visualizing data ineffectively can cause entire claims to be invalidated. As long as we scrutinize our visualization practices, we should be able to avoid obvious pitfalls here.

Timeline & Deliverables

In order to make sure our final visualization is as good as possible, we'll break down the problem into a number of smaller goals. By the midterm, we'd like to:

- have a dataset representative of the format of the final dataset, but smaller for ease of iteration and prototyping.
- have produced some initial prototypes of our visualization graphics. These designs may not be connected to real data.

- have interviewed people to understand what visualization formats are the best to convey our findings.

By the final presentation, we would like to have one or more graphics visualizing our data and analysis. We'll base our analyses on as much data as we can manage, subject to cost, storage, and processing constraints.