Programming languages and collaboration communities on GitHub

Corey Ford

April 22, 2013

Introduction

The website GitHub facilitates collaborative software development and hosts the source code repositories, as well as other development activities, of numerous open-source software projects, many of them widely-known.

To what extent does the diversity of programming language fragment the software development community? We anticipate that the frequency of collaboration between projects using the same programming language (or software framework, especially) will result in communities identified by a programming language.

We construct the one-mode network of repositories, identify important repositories in this network, and examine its community structure, comparing communities defined by programming languages to those determined by common community-detection algorithms.

Related Work

Some recent analyses have examined the programming languages associated with each user via their repositories to determine conditional probabilities [3] or correlations [11] relating to use of language pairs.

User communities based on primary programming languages and "follower" relationships were explored in [2, 13]. A visualization of the bipartite graph of organizations (groups of users) and languages [10] won third place in the 2012 GitHub Data Challenge.

Important nodes in the one-mode project-project and user-user networks, weighted simply by number of common links, were identified in [12].

Methodology

Data was obtained using SQL queries against the Google BigQuery web interface. Further analysis was performed using Python 2.7 and version 0.6.5 of the igraph library, and visualizations were created using Gephi 0.8.2 and sigma.js. All SQL and Python code, along with data files, is available at https://github.com/coyotebush/github-network-analysis.

Data Collection

We queried the GitHub Archive dataset[5] to obtain data on GitHub repositories that had received at least 1000 "stars" (a way for users to bookmark interesting repositories) as of the end of 2012, including the URL, primary programming language, and number of stars for each repository.

Using a second query, we computed a set of weighted edges between these repositories based on two types of events: *pushes*, where a user adds new changes to a repository, and *pull-request merges*, where a user without push access proposes changes that are then reviewed and added to the repository by another user who does.[1] We first computed contribution weights C_{ui} for user-repository pairs as the number of times the user has either pushed to or had a pull-request accepted to the repository during 2012.[2]

Weights between repositories were then computed by examining users who have contributed to each of a pair of repositories, and defining the weight of the edge between repositories i and j as the sum over all users who have contributed to both of the geometric mean of the number of times they have contributed to each. The geometric mean was chosen because it is high when the counts of a user's contributions to both repositories are high, but low when either is low (and elegantly zero when they have not contributed at all to either), and the impact of increasing either is diminishing. That is, the the weights are defined by $W_{ij} \equiv \sum_u \sqrt{C_{ui} \cdot C_{uj}}$, where C is the contribution weight defined previously and u ranges over all GitHub users.[3]

[1] Accounting only for pushes gave a relatively disconnected graph, presumably because, as examined in [6], most users have push access to only a few related repositories. Including other types of events, or measures such as lines of code, might produce different results, but there is no definitive measure of developer productivity.

[2] This counts pull-request merges for both the user who submitted the request and the user who pushed the merge; this seems acceptable, since it accounts for the work of reviewing a pull-request and better identifies repository maintainers.

[3] This is similar to the approach of [7], which effectively used $\sqrt{min(C_{ui}, C_{uj})}$; [8, 9] provide additional insight into projections of such two-mode networks.

The resulting graph contains 825 nodes and 3661 edges. However, for the remainder of the analysis we consider only the giant component of this graph, which contains 632 nodes and 3644 edges. This is justified by the lack of meaningful structure in the remainder of the graph—the remaining connected components comprise at most 3 nodes each—and reduces clutter both in the visualization (which would have many nodes floating at the edge) and the community detection (which would identify most of these nodes as their own communities).

Visualization

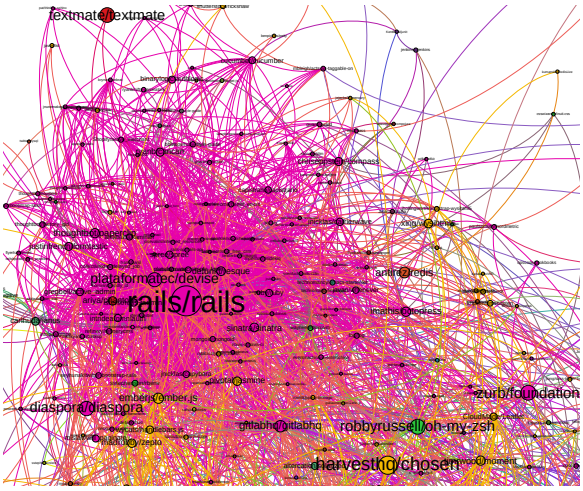
A visualization of the network using the ForceAtlas 2 layout algorithm, with nodes colored according to programming language and sized according to number of stars, is shown in figure 1. An interactive version is available at http://coyotebush.github.io/github-network-analysis/.

Figure 1: GitHub repository network.

Some language-based communities are already visually evident, including

- Ruby (pink), which is noticeably centered around the prominent Ruby on Rails project (figure 2).
- JavaScript (yellow), the most popular language on GitHub and fairly central to this network.
- Python (blue).
- Objective-C (teal), which is relatively sparse.

Figure 2: Ruby network.



Analysis[4]

Notable repositories

Central repositories, defined by degree and betweenness, both weighted and unweighted, are presented in table 1.

```
g.vs['degree']   = g.degree()
g.vs['degreew']  = g.strength(weights='weight')
g.vs['between']  = g.betweenness()
```

[4]This section was created using Pweave (http://mpastell.com/pweave/) and matrix2latex (http://code.google.com/p/matrix2latex/)

```
g.vs['betweenw'] = g.betweenness(weights='weight')

top_degree   = g.vs[reversed(np.argsort(g.vs['degree'])[-5:].tolist())]
top_degreew  = g.vs[reversed(np.argsort(g.vs['degreew'])[-5:].tolist())]
top_between  = g.vs[reversed(np.argsort(g.vs['between'])[-5:].tolist())]
top_betweenw = g.vs[reversed(np.argsort(g.vs['betweenw'])[-5:].tolist())]
```

Table 1: Top nodes (repositories) by various measures

(a) Unweighted degree

rails/rails	175
github/hubot-scripts	78
carlhuda/bundler	71
mperham/sidekiq	66
gitlabhq/gitlabhq	64

(b) Weighted degree

rails/rails	7937.6
symfony/symfony	5634.9
ajaxorg/cloud9	3442.6
ajaxorg/ace	3291.2
fabpot/Silex	3124.5

(c) Unweighted betweenness

rails/rails	39917.9
github/hubot-scripts	12037.5
gitlabhq/gitlabhq	10632.4
visionmedia/mocha	8075.6
mitchellh/vagrant	7590.4

(d) Weighted betweenness

robbyrussell/oh-my-zsh	21736.0
rails/rails	21157.2
github/hubot-scripts	16747.9
gitlabhq/gitlabhq	14675.2
mitchellh/vagrant	12776.6

Discussion

Clearly, 'rails/rails', the Ruby on Rails web application framework, is a very important project on GitHub. This makes sense considering the strong community of Rails plugins on GitHub, and that GitHub itself uses Rails.

Many of the other repositories in tables 1a, 1c, and 1d (excluding 'mperham/sidekiq' and 'visionmedia/mocha') serve as development tools moreso than as independent frameworks/applications, so their importance is likely due to their use by developers who primarily code in other languages.

Weighted degree (table 1b), which best represents the amount of collaboration surrounding a project, includes none of these. From the visualization, collaboration surrounding the two 'ajaxorg' projects appears to be primarily between the two, and the other three are web application frameworks.

Communities Defined by Language

We group repositories by language and first calculate the resulting modularity using both the weighted and unweighted algorithms implemented in igraph; the results are presented in table 2.

```
by_language = igraph.VertexClustering.FromAttribute(g, 'language')
mod_u = g.modularity(by_language)
mod_w = g.modularity(by_language, 'weight')
```

Table 2: Properties of clustering by language.

Unweighted modularity	0.290
Weighted modularity	0.469

We then examine the language-based clusters as independent subgraphs, presented in descending order of number of repositories. Measures of community strength are calculated below and presented in table 3: the density (fraction of possible edges that exist) and forms of the clustering coefficient (probability that any two neighbors of a node are themselves connected) for the entire subgraph, the average for each node, and a variant of the latter that accounts for edge weights.

```
langs = sorted(by_language.subgraphs(),
    key=lambda x: len(x.vs), reverse=True)
names = [s.vs[0]['language'].replace('#', '\#') for s in langs]
langs.append(g) # Include statistics on entire graph
names.append('OVERALL')

count_n = [len(s.vs)                   for s in langs]
count_e = [len(s.es)                   for s in langs]
density = [s.density()                 for s in langs]
tr      = [s.transitivity_undirected() for s in langs]
tr_avg  = [s.transitivity_avglocal_undirected(mode='zero')
    for s in langs]
tr_avgw = [s.transitivity_avglocal_undirected(mode='zero',
    weights='weight') for s in langs]
```

Discussion

The modularity resulting from clustering by language indicates significant community structure, especially in the weighted version, which considers amount of collaboration rather than merely its existence.

Table 3: Languages in the dataset, subgraph density and clustering coefficients (in averages, nodes with degree<2 treated as having clustering coefficient 0).

Language	Nodes	Edges	Density	Global	Local	Weighted
JavaScript	246	642	0.021	0.247	0.270	0.289
Ruby	149	1044	0.095	0.345	0.472	0.515
Objective-C	50	71	0.058	0.179	0.209	0.222
Python	37	127	0.191	0.556	0.486	0.516
PHP	31	79	0.170	0.686	0.479	0.522
Shell	18	5	0.033	0.000	0.000	0.000
Java	16	7	0.058	1.000	0.188	0.188
CoffeeScript	15	12	0.114	0.500	0.456	0.471
VimL	15	14	0.133	0.811	0.307	0.315
C++	13	1	0.013	-	0.000	0.000
(unknown)	12	1	0.015	-	0.000	0.000
C	10	2	0.044	0.000	0.000	0.000
C#	7	4	0.190	0.000	0.000	0.000
Scala	4	2	0.333	0.000	0.000	0.000
Clojure	2	1	1.000	-	0.000	0.000
Perl	1	0	-	-	0.000	0.000
Objective-J	1	0	-	-	0.000	0.000
ActionScript	1	0	-	-	0.000	0.000
Julia	1	0	-	-	0.000	0.000
Rust	1	0	-	-	0.000	0.000
Go	1	0	-	-	0.000	0.000
Emacs Lisp	1	0	-	-	0.000	0.000
OVERALL	632	3644	0.018	0.227	0.334	0.365

Among the top languages, the subgraph statistics for competing languages Ruby, Python and PHP are notable. Although there are also many popular projects using Objective-C and Java, their lower clustering coefficients reflect the perhaps less open culture associated with these languages. JavaScript projects are abundant but loosely connected. The Shell network is quite sparse; again, most of these are probably development tools.

Languages in Detected Communities

We apply two community-finding algorithms implemented in igraph, the Infomap method and the greedy optimization of modularity, to the graph. Tables 4 and 5 present the largest communities found by each of these algorithms, and the most common programming languages in each community.

```
info = g.community_infomap(edge_weights='weight')
info_mod = info.modularity
info_largest = sorted(info.subgraphs(), key=lambda x: len(x.vs),
        reverse=True)[0:5]
info_sizes = [len(s.vs) for s in info_largest]
info_langs = [collections.Counter(s.vs['language']).most_common(5)
        for s in info_largest]

greedy = g.community_fastgreedy(weights='weight').as_clustering()
greedy_mod = greedy.modularity
greedy_largest = sorted(greedy.subgraphs(), key=lambda x: len(x.vs),
        reverse=True)[0:5]
greedy_sizes = [len(s.vs) for s in greedy_largest]
greedy_langs = [collections.Counter(s.vs['language']).most_common(5)
        for s in greedy_largest]
```

Table 4: Largest communities, Infomap (weighted modularity: 0.714)

Size	Top languages
62	Ruby (48), JavaScript (7), C++ (2), C (1), C# (1)
22	Ruby (8), CoffeeScript (3), Shell (2), JavaScript (2), C (2)
19	PHP (11), JavaScript (4), (unknown) (1), Shell (1), C (1)
19	JavaScript (15), (unknown) (2), Python (1), Shell (1)
15	Ruby (9), Shell (2), JavaScript (1), VimL (1), C++ (1)

Table 5: Largest communities, greedy (weighted modularity: 0.760)

Size	Top languages
147	Ruby (104), JavaScript (18), C++ (6), Shell (3), Objective-C (3)
71	Objective-C (37), JavaScript (12), Ruby (7), PHP (5), Python (3)
70	JavaScript (50), (unknown) (4), Shell (3), C# (3), PHP (3)
69	Python (31), JavaScript (17), Objective-C (5), Ruby (4), Shell (2)
40	JavaScript (16), Ruby (8), Java (4), CoffeeScript (3), Shell (2)

Discussion

Both of these algorithms find a clustering with very high modularity. [5] While Infomap more cautiously finds smaller communities, in both cases most of the largest communities contain a majority of one of the top programming languages in the dataset. This suggests that collaboration communities do, in large part, form around languages. Some intra-language collaboration is also evident, especially related to JavaScript; this might be explained by JavaScript's use as a secondary, client-side language in web applications written in another language.

Conclusions

On analyzing the network of collaboration between top GitHub repositories, we find that programming language use is a reasonable indicator of community structure, and vice versa. Still, there is plenty of collaborative activity between projects using different languages.

A more complete picture might be obtained by lowering the threshold for including repositories, thereby including less popular side projects, or by changing the time range under consideration. Further work might also more carefully analyze the relationships between languages themselves, ideally incorporating data on non-primary languages used in a project.

[5] The Girvan-Newman edge-betweenness algorithm (`community_edge_betweenness`), in addition to being very computationally expensive, at best found a modularity of 0.290, or 0.347 for the weighted version.

Acknowledgements

This project is being developed both as the final project for a course in social network analysis [1], and as an entry in the GitHub Data Challenge [4].

Thanks to Peter Faiman and Corey Farwell for help regarding SQL and visualization, respectively.

References

[1] Lada Adamic. Social network analysis. https://www.coursera.org/course/sna, 2013.

[2] Franck Cuny. Github explorer. http://lumberjaph.net/graph/2010/03/25/github-explorer.html, March 2010.

[3] Brian Doll. Programming language correlation dataset. https://gist.github.com/briandoll/e0637fff9c8eec988528, April 2012.

[4] Brian Doll. The GitHub data challenge II. https://github.com/blog/1450, April 2013.

[5] GitHub Archive. http://www.githubarchive.org/.

[6] Nikhil Khadke, Ming Han Teh, and Sharon Tan. An analysis of GitHub's collaborative software network. December 2012. Course project report.

[7] Joseph Marrama and Tiffany Low. Social coding: Evaluating Github's network using weighted community detection. Course project report.

[8] Tore Opsahl. Projection. http://toreopsahl.com/tnet/two-mode-networks/projection/.

[9] Tore Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35, 2011.

[10] Eduarda Mendes Rodrigues. GitHub data. http://labs.sapo.pt/networds/2012/05/09/github-data/, May 2012.

[11] Akshay Shah. Language use on GitHub. http://datahackermd.com/2013/language-use-on-github/, February 2013.

[12] Ferdian Thung, David Lo, and Lingxiao Jiang. Network structure of social coding in GitHub. 17th European Conference on Software Maintenance and Reengineering (CSMR), 2013.

[13] Nicholas M. Weber. Combined methods, thick descriptions: Languages of collaboration on Github. *Proceedings of the American Society for Information Science and Technology*, 49(1):1–4, 2012.