A Study of External Community Contribution to Open-Source Projects on GitHub

Rohan Padhye
IBM Research India
ropadhye@in.ibm.com

Senthil Mani
IBM Research India
sentmani@in.ibm.com

Vibha Singhal Sinha
IBM Research India
vibha.sinha@in.ibm.com

ABSTRACT

Open-source software projects are primarily driven by community contribution. However, commit access to such projects' software repositories is often strictly controlled. These projects prefer to solicit external participation in the form of patches or pull requests. In this paper, we analyze a set of 89 top-starred GitHub projects and their forks in order to explore the nature and distribution of such community contribution. We first classify commits (and developers) into three categories: CORE, EXTERNAL and MUTANT, and study the relative sizes of each of these classes through a ring-based visualization. We observe that projects written in mainstream scripting languages such as JavaScript and Python tend to include more external participation than projects written in upcoming languages such as Scala. We also visualize the geographic spread of these communities via geocoding. Finally, we classify the types of pull requests submitted based on their labels and observe that bug fixes are more likely to be merged into the main projects as compared to feature enhancements.

Categories and Subject Descriptors

K.4.3 [**Computers and Society**]: Organizational Impacts—*Computer-supported cooperative work*; D.2.8 [**Software Engineering**]: Distribution, Maintenance and Enhancement—*Version control*

General Terms

Human Factors, Languages

Keywords

Open-source software, core committers, external contribution, pull requests, community participation, mining software repositories

1. INTRODUCTION

The success of open-source software projects depends heavily on active community participation in all its phases including planning, development, maintenance and documentation. Participants often collaborate using one or more shared software repositories for storing and managing source code, documentation and bug reports.

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597113

Most projects encourage any and all users to file a bug report when they spot some unexpected behavior, and many projects even maintain their documentation as public wikis. However, write access to the project's source code is almost always strictly controlled. A chosen set of developers, often referred to as *core committers*, have the ability to make changes to the project's code-base and all other community participation is solicited through distinct mediums such as patches or pull requests. In fact, continued submission of high-quality patches is an important requirement for an external contributor to be promoted to the level of a core committer [7].

In this paper, we analyze a data set of 89 popular projects hosted on GitHub and their 108,000+ forks [5] in order to study the levels of participation from different communities of the root projects, which we term as CORE, EXTERNAL or MUTANT. These categories are defined as follows:

- A CORE committer is a developer who has write access to a project's repository, and all commits directly associated with the main repository are considered CORE commits.
- An EXTERNAL commit is one which is incorporated into a project indirectly through patches or pull requests that need to be accepted by a CORE committer; the author of such a commit is called an EXTERNAL committer.
- A MUTANT is a modification to the code-base of a project which is not incorporated back into the main repository. This may occur either if the change was rejected by CORE committers or if the author made this change for personal uses only. Developers who have only made MUTANT changes belong to the MUTANT community.

Note that each of these categories are defined on a per-project basis, so a CORE-committer in one project may be an EXTERNAL-committer in another project. We formulate our study as a set of four research questions:

- **RQ1**: What is the distribution of relative sizes of communities (CORE, EXTERNAL and MUTANT) in the root projects considering (1) number of users in each community and (2) number of commits contributed by each community?
- **RQ2**: Is the distribution of relative sizes of communities impacted by the main programming or scripting language used by the root project?
- **RQ3**: Are the communities that are contributing to these open-source projects geographically diverse or concentrated?
- **RQ4**: What is the nature of external contribution (e.g. bug fix, feature enhancement or documentation) and do the maintainers of root projects have a preference of either type in deciding whether or not to incorporate such external contribution or to reject it?

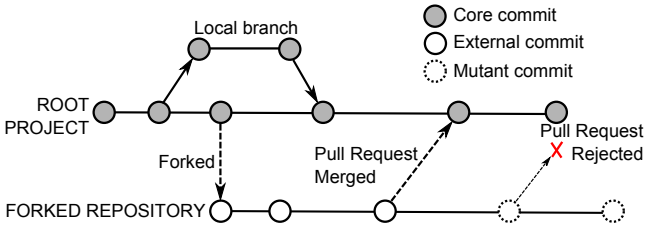

Figure 1: Simplified example time-line of a repository and its fork showing classification of CORE, EXTERNAL and MUTANT commits.

We first explain the *fork-and-pull* model of GitHub and how it is critical in enabling this study in Section 2. We describe our analysis of the data-set and address the four research questions in Section 3, followed by a conclusion in Section 4.

2. THE FORK-AND-PULL MODEL

Decentralized version-control systems such as Git allow users to clone entire repositories, giving every user write access to their own local clones, and thus facilitate the management of community driven participation [4]. In Git, commits are simply a set of changes (deltas) applied on some state of the system, referred to as the parent commit(s). This allows a very lightweight implementation of branching and merging. Another implication of such an architecture is that commits made on one repository can be seamlessly transferred to another repository as long as there is a common baseline (parents) between the two repositories. Developers can *push* commits to a remote repository, if they have write access, or ask the maintainers of a remote repository to *pull* in their commits.

GitHub [1] is a Web-based project-hosting service which is based on Git. The platform allows users to leverage the decentralized features of Git while maintaining a centrally hosted canonical repository and provides other features such as an integrated issue management system. A distinguishing feature of GitHub is the *fork-and-pull* model, which implements the aforementioned mechanism for seamlessly exchanging commits across clones of the same project. All users, regardless of whether they have write access to a project, can *fork* its repository with a single-click, just as if they were creating a lightweight branch. The fork is a remote-hosted clone of the repository completely owned by such users and they can modify its code and even add other users in order to collaboratively extend the code-base. If the developers wish to contribute these extensions back to the original project, they can open-up a *pull request*, which is implemented simply as an item in the issue tracker. Thus, there is a central platform for a discussion between the root project's maintainers and the external contributors. This discussion can be used as a code review and the external contributors may even be asked to make further modifications (such as write the proper documentation or test-cases to validate their code). Any such changes can be appended to the same pull request easily. Finally, a pull-request may be merged into the original project or be rejected and closed.

Figure 1 shows an example time-line of commits in a root project and its fork. CORE-commits are those which are logged directly on the root repository. EXTERNAL commits are all commits which are logged on forks of the root project, but which flow into the root repository through accepted pull requests. All commits made on the forks which are not incorporated in the upstream root project are classified as MUTANTs.

Our study of such communities is only possible using data from a platform like GitHub in which all commit activity, including that on forks, can be easily traced. In traditional open-source development with centralized version control systems such as CVS or Subversion, external community contribution occurs through the exchange of patches, which are often managed in distinct channels such as mailing lists, and which do not have associated change history. Further, it is almost impossible to measure the activity of developers who extend such projects' source code for personal uses only without submitting patches to the original repository.

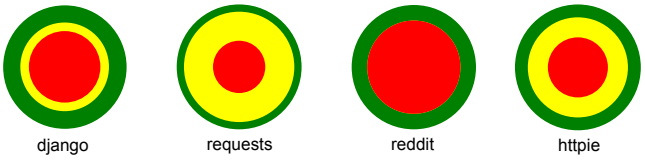

Figure 2: Ring visualization for relative community sizes by developer count for four Python projects.

3. DATA ANALYSIS

The data set [5] used for our study contains, among other things, commit history and pull requests of 89 top-starred GitHub projects written in different languages and their forks[1].

Of the 108,629 forks in this data-set, only 18,343 had at least one commit, indicating that a majority of the forks are just stubs. The root projects and forks combined account for a total of 548,299 commits by 23,237 distinct users, which we analyzed and classified, with respect to each root project, as CORE, EXTERNAL or MUTANT. Section 3.1 describes our ring-based visualization for studying the relative sizes of these communities, while Section 3.2 analyzes the impact of programming language on these communities.

12,635 users (54.37%) had entered some location data in their profiles. We resolved these arbitrary location strings to latitude and longitude coordinates using the Google GeoCoding API [2], and used the results to study the geographic distribution of different communities for each project, which is presented in Section 3.3.

Of the 18,343 forks which had activity, 18,278 forks (99.65%) had initiated at least one pull request. This shows that developers who make changes to forks of popular projects almost always consider contributing their changes back to the original project, as opposed to exclusively making custom changes for personal use. However, only 6,952 of these forks (38%) had at least one of their pull requests accepted. Section 3.4 investigates the merge ratios for different types of pull requests.

3.1 Project Communities

For each root project, we developed a ring-based visualization in order to compare the sizes of different communities as shown in Figure 2. The area of the innermost circle represents the size of the CORE committers, who have commit access to the root project, the area of the middle ring corresponds to the size of the EXTERNAL committers, whose commits have been incorporated in the root project via pull requests, and the outermost ring represents the MUTANT committers, who have extended forks of the root project but whose work has not made it upstream. The diagrams have been normalized such that the diameter of the outer ring is the same for all projects, and hence the rings represent *relative* community sizes.

The figure shows rings for four projects written in Python. Inter-

[1]Although the original data-set lists 90 root projects, we found that one project, `xphere-forks/symfony`, was a fork of another root project in the list, and hence we excluded it from our study.

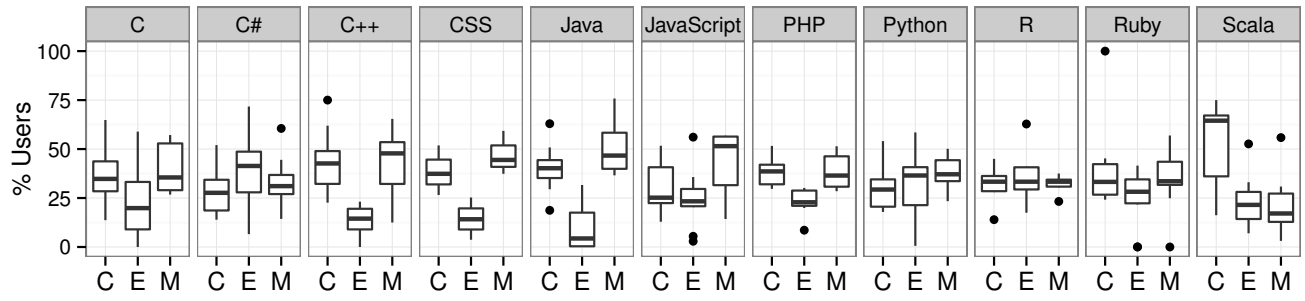

Figure 3: Distribution of relative community sizes – CORE (C), EXTERNAL (E) and MUTANT (M) – by number of developers, grouped by language.

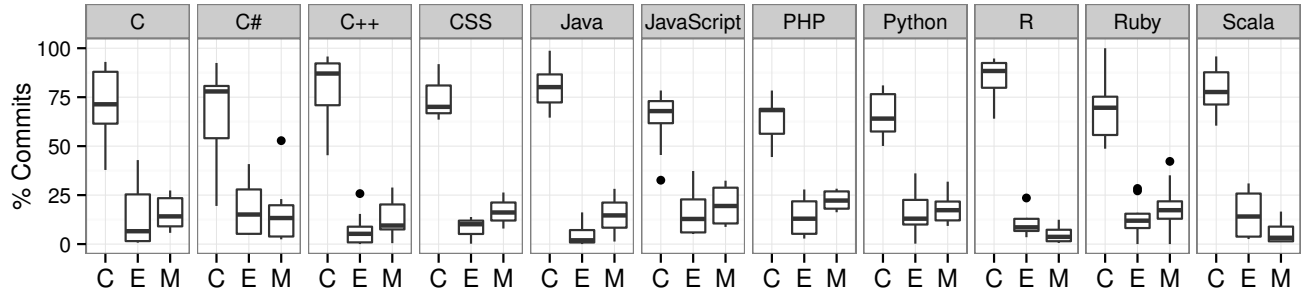

Figure 4: Distribution of relative community sizes – CORE (C), EXTERNAL (E) and MUTANT (M) – by commit count, grouped by language.

esting observations can be made from this visualization. For example, `httpie` has roughly an equal number of CORE-COMMITTERS as well as EXTERNAL contributors, and a small number of MUTANT-COMMITTERS as well. The `django` community is made up primarly of CORE committers, with some EXTERNAL participation as well. On the other hand, the `requests` project has a much larger community of EXTERNAL contributors compared to its CORE base; in fact the relatively smaller size of the MUTANT community indicates that almost all users who initiated pull requests had at least one of them merged. The `reddit` project does not seem to incorporate a lot of external contribution, having only one user in that community, although many more have made extensions to forks and hence lie in the MUTANT community. Similar visualizations may be studied for commit counts as well (instead of number of committers). The visualizations for the entire data-set of 89 projects (both by commits and by users) have been made available on the Web[2].

3.2 Programming Language

Figures 3 and 4 are box-plots of relative community sizes (summing up to 100%) for each project, grouped by language, for number of developers and commit counts respectively.

From Figure 3 we can see that different languages show different characteristics with respect to relative community sizes. The sizes of EXTERNAL communities for popular scripting languages such as JavaScript, PHP, Python and Ruby are comparable to the sizes of their CORE communities. Projects written in C# had the highest median value of relative size of the EXTERNAL community. On the other hand, all Scala projects had an unusually large proportion of CORE committers. Projects written in C and Java seem to have larger MUTANT communities than EXTERNAL; perhaps the policies for accepting pull requests may be more strict for such projects.

Figure 4 tells a more uniform story. Regardless of language, the number of commits by CORE-COMMITTERS always dominated the number of commits by any other community. This is intuitive as CORE-COMMITTERS are usually the only ones involved at the beginning of projects (before they are frequently forked) and they would naturally be more inclined to continue maintaining the project even after EXTERNAL participation has started appearing.

3.3 Geographic Distribution

Figure 5 shows the geographic distribution of CORE committers for the `django` project, which has been plotted using the Google Maps API [3]. Each marker in the map is labeled with a count of the number of users in that region. The exact locations of users within that region can be examined in more detail by *zooming-in*. An interactive demo of these maps for all projects and all communities has also been made available on the Web[2].

Table 1 lists some summary statistics derived from our study of geographic distribution of communities. It was clear that over all, the United States dominated the geographic composition of the CORE ($\geq 50\%$ committers in 5 projects) and EXTERNAL ($\geq 50\%$ in 6 projects) communities. No other country had more than even 20% of the CORE committer community for any project, though 10 countries had between $10 - 20\%$ of the CORE community. Germany is a distant but clear second in the top countries for both the CORE as well as EXTERNAL communities. It appears that there is more EXTERNAL participation from non-US countries as opposed to CORE community membership.

[2] `http://code.comprehend.in:8080/msr14`

Figure 5: Geographic spread of CORE-COMMITTERS for the `django` project.

Table 1: Summary stats for geographic distribution of CORE and EXTERNAL communities for the 89 root projects.

Community	Members	United States	Others (# Projects)
Core	$\geq 10\%$	81 projects	Germany (5), UK (3), Sweden (3), France (3), Australia (2), China (2), Switzerland (2), Brazil (1), South Korea (1), Russia (1)
	$\geq 20\%$	51 projects	(none)
	$\geq 50\%$	5 projects	(none)
External	$\geq 10\%$	72 projects	Germany (10), UK (8), Japan (4), France (4), Switzerland (2), Netherlands (2), Canada (2), 11 other countries (1)
	$\geq 20\%$	44 projects	Germany (3), Japan (2), France (1), Canada (1), UK (1), Belarus (1)
	$\geq 50\%$	6 projects	(none)

3.4 Nature of Pull Requests

Of the 75,526 closed pull requests in the data-set, only 34,125 (45.18%) were merged into root projects. We wanted to explore some of the factors that may impact whether or not a pull request is accepted by maintainers of a root project.

We found that 7,529 closed pull requests in the data-set (about 10%) had labels associated with them. We analyzed these labels and classified the pull requests as a bug fix, feature enhancement or documentation contribution and observed the merge ratio for each of these categories.

The 7,529 labelled pull requests accounted for a total of 274 distinct labels, which ranged from meaningful terms such as "feature" and "bug" to project-specific terminology such as "oracle" and "v0.10". We manually classified these labels into four tags – Bug Fix, Enhancement, Documentation or Unknown. The number of closed pull requests that we could successfully tag was 2,117. Of these, 1,140 (53.85%) were merged into root projects.

Figure 6 is a bar chart showing the total number of closed pull requests and the fraction of them that were merged, for different categories. As the figure shows, 66.84% of pull requests tagged as Bug Fix were successfully merged into root projects, while 46.97% of Feature Enhancements were merged. Also, about 60% of pull requests tagged Documentation were merged.

From this data, we may infer that core project maintainers are more open to accept external contributions that can immediately be seen to fix a known bug or those that update documentation-only, rather than incorporate new functionality into their project. However, the validity of this inference may be threatened by the fact that a relatively small proportion (about 10%) of pull requests were actually labelled.

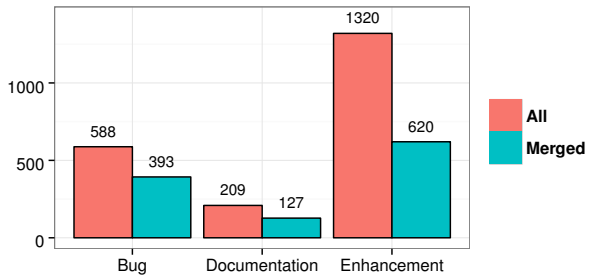

Figure 6: Number of pull requests merged, classified by types derived from labels.

Gousios et al. [6] conducted a similar study on a much larger data-set, and included an analysis of natural language comments, from which they concluded that the reasons for closing a pull request without merging were more often than not of a non-technical nature (e.g. duplicate, obsolete, not following standards, etc). Our study explored a different aspect, that of the *kind* of pull requests that are being merged or rejected.

4. CONCLUSION

We have leveraged the *fork-and-pull* model of GitHub to study community contribution for 89 popular open-source projects. We observed that a large number of developers, from diverse geographic backgrounds, consider participating in community-driven development of open-source projects, regardless of whether or not they have write access to a project's main repository. We also noticed that not all such community contribution is incorporated back into the original project. Core developers seem relatively open to accept bug fixes and documentation changes from the external community, but may be apprehensive about proposed feature enhancements.

5. REFERENCES

[1] GitHub. `https://github.com`. Accessed: Feb 2014.

[2] Google Geocoding API. `https://developers.google.com/maps/documentation/geocoding`. Accessed: Feb 2014.

[3] Google Maps API. `https://developers.google.com/maps/`. Accessed: Feb 2014.

[4] B. de Alwis and J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, CHASE '09, pages 36–39, Washington, DC, USA, 2009. IEEE Computer Society.

[5] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR'13, pages 233–236, 2013.

[6] G. Gousios, M. Pinzger, and A. van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE '14, 2014. To appear.

[7] V. S. Sinha, S. Mani, and S. Sinha. Entering the circle of trust: Developer initiation as committers in open-source projects. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 133–142, New York, NY, USA, 2011. ACM.