The GitHub Open Source Development Process

Kevin Peterson
kevin@kevinp.me

Abstract

Open Source Software (OSS) projects, or software projects with publicly available source code, are realizing ever more important roles in both personal and business computing. As such, shifts in the way in which OSS is developed could have impacts on both the quantity and the quality of OSS projects.

The development process by which these projects are produced is generally unstructured compared to commercial software, but many projects do exhibit general development patterns. GitHub, a popular OSS code hosting website, along with Git, the site's Source Code Management (SCM) tool of choice, may have the potential to fundamentally change this process by facilitating new patterns and opportunities for developers.

By analyzing a subset of GitHub repositories, this report will show how GitHub has influenced some intrinsic aspects of traditional OSS development, such as developer hierarchies and issue close velocity. We find that many of the traditional aspects of OSS development remain, such as most project development being done by a small group of core developers. Other traditional assumptions about OSS developer hierarchy, such as a large number of Issue Reporters compared to Committers, seems unsupported by the GitHub data. We conclude that GitHub represents an evolution of the OSS development process, and not necessarily a large shift.

Keywords. Git, GitHub, Open Source

1 Introduction

Open Source Software (OSS) has fundamentally changed how we view the Software Development Process [25]. OSS projects are not only viable, but successful and thriving.

A case study of the Apache Server project [20] showed that a dedicated community can produce software that rivals or exceeds commercial offerings. Furthermore, it showed that even in the unstructured context of OSS, certain structures, hierarchies, and codes of conduct emerge. Contrast the Apache Server project to a successful project on GitHub. Although the spirit and intent may be the same, the tool set is drastically different. The intent of this work is to explore how GitHub facilitates this process, and also how it may be causing it to evolve.

GitHub[1] is a popular code hosting website that uses Git Source Code Management (SCM).[2] Along with hosting software repositories, GitHub incorporates the social aspects of development. GitHub users have community-visible profiles, and user actions can be tracked and followed by other community members. This type of social integration of a user's identity and actions is unique to GitHub [11].

Git itself, the SCM of GitHub, lends itself to the collaborative nature of GitHub by allowing for development to take place in a more distributed manner than previously available in other SCM systems [27]. Git also allows for a variety of work flows [8]. These work flows may be tailored to the individual needs of a project. The Linux Kernel, for example, uses a Dictator and Lieutenants Workflow [24], which is hierarchical in nature. Other work flows may be more distributed, or resemble more traditional centralized SCM systems. Although work flow techniques aren't explicitly addressed in this work, it is important to note that work flow change may contribute to *process change*.

To better understand quantitatively the GitHub process, three hypotheses are introduced. The goal of these hypotheses is to bridge the gap between the more qualitative research of GitHub [11, 4] and the statistical analysis of data from a random sampling

[1] https://github.com/
[2] http://git-scm.com/

of GitHub projects.

In addition, three supporting research questions are proposed. These questions will draw upon data from this work, but also reference several case studies to better represent how GitHub may be changing the way in which OSS development is conducted. These qualitative observations, paired with the more quantitative data analysis, allow for a broad description of the way that GitHub is changing the Software Engineering landscape for OSS.

1.1 Hypotheses

The following hypotheses are presented in an effort to better understand the GitHub development process by analyzing data from actual GitHub projects. Each hypothesis will be backed by previous research in an attempt to show concrete ways in which GitHub may be changing the OSS development process. Where possible, results from previous research on the Apache Server [20] are compared, as this work closely aligns with Hypothesis 2 and 3.

Hypothesis 1: As the number of Watchers increase, the number of Repository Forks will increase.
In Open Source Software, a feeling of community belonging can be intrinsically motivating to developers [19]. This feeling of belonging can be expressed passively, as a *watch*, or actively, as a *fork* of a repository.

In GitHub, a *watch* is a user act of subscribing to changes in a repository. It is also referred to as a *star*. A *watch* allows users to receive status updates, such as when code has been committed. This is considered a *passive* act, as it signifies involvement in the community, but no direct action to contribute.

A *fork* is the creation of a personal copy of a repository [11], from which changes may be made without impacting the original code source. A *fork* is an *active* action, as it signifies both involvement and intent to contribute. It should be noted, however, that it only signifies *intent* – a *fork* of a repository could be made and no further actions taken.

As *watches* don't necessarily signal intent to contribute, a positive direction for an OSS project is to not only increase the number of *watchers*, but also the number of *forks*. As both signify a general interest in the project, it is expected that there will be some correlation between the two.

Hypothesis 2: As the number Repository Forks increase, the Issue Resolution Time will decrease.
In OSS, a project *fork* has at times carried negative connotations, and has even been referred to as a hazzard [17]. In the context of Git and GitHub, however, a *fork* is a positive occurrence for a project, as it signifies greater project involvement. As Git allows for easy merging of *forks*, code contributions in the form of defect fixes can be incorporated quickly. Because of this, it is expected that *Issue Resolution Time* will decrease as the number of *forks* increases.

The Apache Project noted that a rapid response to problems can be obtained because OSS is not bound to release schedules in the same way as commercial software. "Patches" may be released at any time, by any member of the community [20]. "Patches" in the context of GitHub could be equated to *Pull Requests*. A *Pull Request* is an announcement by a *fork* that there have been changes made that may be appropriate to add back to the main repository. The owners of the main repository may then audit these requests, and *pull* some or all of them back into the main repository.

This hypothesis is, ultimately, an examination of "Linus Law" [25] in the context of GitHub. In other words, the more exposure the code gets (in terms of *forks*), the easier bugs will be to find and fix.

Hypothesis 3: There will be more Issue Reporters than Committers by an order of magnitude.
Research into the Apache Server project observed that there were far more *Issue Reporters* than there were code *Committers* [20]. The Apache Server project is an excellent case study in this type of Developer Hierarchy. Apache is built around a small set of *Core Developers*, followed by *Defect Repairers* and *Defect Reporters*. Each level of this hierarchy brings with it an order of magnitude increase in number of participants. The 10-15 *Core Developers* contribute around 80% of the new functionality, while the rest of the 400 code contributors focus primarily on bug fixes. The *Defect Reporters* were by far the largest group, with over 3000 individuals submitting bug reports.

Because issue reporting is of low risk to the code base, but has potentially high value, it is a perfect way for large numbers of people to contribute. It is expected that the findings of the Apache Server project research will hold true for GitHub projects as well.

1.2 Research Questions

To expand the analysis scope slightly, three qualitative research questions are posed. These, although

backed by analyzed data where possible, are intended to show broader patterns, ideas, and motivations around the use of GitHub. Where possible, interviews [4] were conducted to explore high level concepts and motivations.

Research Question 1: Are GitHub projects primarily focused around a small set of core Committers?
A small core of developers contributing to a project seems to be a well entrenched pattern of OSS development [20, 21, 18]. At the extreme of this, a study of projects on the software hosting site SourceForge[3] found that a large percentage of OSS development is done by lone developers [18]. Is a small core of developers intrinsic to OSS development, or have the social aspects of GitHub and the distributed nature of Git changed this?

Research Question 2: How is GitHub changing the OSS process?
The social aspects of GitHub are an important part of the development experience [11]. User interactions and social pressures can drive OSS development in interesting ways. Social aspects of OSS development have existed, however, before GitHub, with mailing lists [20], gift giving mechanisms [5], and other collaboration tools. What is GitHub doing to facilitate and capitalize on the social aspects of OSS development, and how might this influence the OSS development process?

Research Question 3: Can GitHub be used for more than code artifacts?
The Object Management Group®[4] is a computer industry consortium focused on technology standards. Potential standard submission teams must go through a vetting process [16] which allows industry representatives to provide feedback on the standard. Gathering and recording this feedback is a challenge, but GitHub's issue tracking system may be able to streamline the process. This is just one example of how GitHub is being used for non-code artifacts. Are organizations exploring this type of work flow?

[3]http://sourceforge.net
[4]http://www.omg.org

2 Methods

2.1 Collection and Storage

Data collection was done using the GitHub REST API.[5] A random selection of 1000 repositories was selected, and metrics were gathered. Only repositories owned by a GitHub *Organization* were considered. GitHub *Organizations* are group-owned accounts,[6] as opposed to normal individual user-owned accounts. *Organization* owned repositories were chosen to increase the likelihood of analyzing high-participation projects. The selection process used for choosing the repository sample is described in Algorithm 1.

Algorithm 1 Data Collection Algorithm

```
1: while i < 1000 do
2:     i ← i + 1
3:     word ← RANDOM(word_list)
4:     repos ← GITHUB_SEARCH(word)
5:     j ← RANDOM(repos)
6:     repo ← repos_j
7:     if ISORGREPO(repo) then
8:         STORE(repo)
9:     end if
10: end while
```

To select a repository, a random word is selected from an arbitrary list via the RANDOM(*word_list*) function. Using the GitHub search API, a list of repositories containing that word in the description is retrieved using the GITHUB_SEARCH(*word*) function, from which one repository is randomly selected using RANDOM(*repos*). When a random repository has been selected, the ISORGREPO function will filter out all non *Organization* owned repositories. Next, the STORE function accepts as an input the given repository. The purpose of this function is to persist the given repository to a database for further analysis. Results were stored in a MySQL relational database management system. The database schema to store the data is described in Figure 1.

Data was collected and analyzed using a Python[7] script. Because the GitHub REST API constrains the number of API calls per hour made by a client, logic was included to pause execution when the allotment was exhausted.

Matplotlib [14], a 2D graphics package for Python, was used to render graphs and charts. NumPy and SciPy [15] were used for data analysis calculations, such as correlation coefficients.

[5]http://developer.github.com/v3/
[6]https://github.com/blog/674-introducing-organizations
[7]http://python.org/

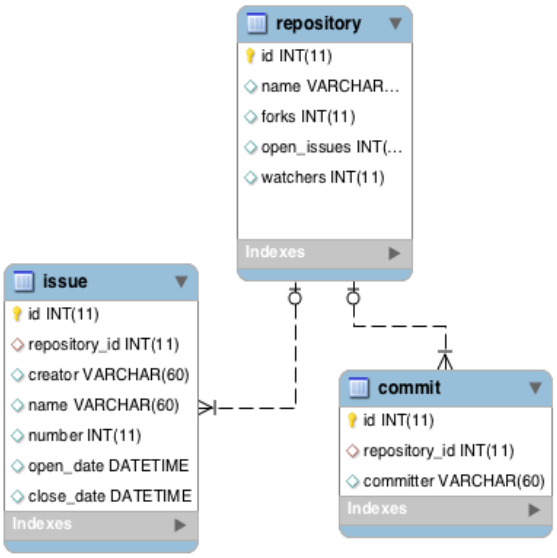

Figure 1: GitHub data entity relationship diagram

All source code and materials used in data collection and analysis are hosted at GitHub (naturally!), and may be accessed by the following URL: `https://github.com/kevinpeterson/github-process-research`.

2.2 Analysis

As stated, the main focus of data collection was to gather metrics pertaining to GitHub *Repositories*. In this context, a *Repository* is defined as a set of *Attributes*. An *Attribute* is a metric of interest. See Section 2.2.1 for further information on individual *Attributes*. For analysis purposes, these metrics of interest are used as an Indexing Set A', and have labels as follows:

$$A' = \{i, c, i', i'', \Delta, \omega, \epsilon\}$$

Where i = *Issues*, c = *Commits*, i' = *Closed Issues*, i'' = *Open Issues*, Δ = *Issue Close Time*, ω = *Issue Reporters*, and ϵ = *Committers*.

All analyzed *Repositories* can be thought of as the Indexed Family of *Attribute* Sets, or a set of sets of *Attributes*. Let the set of these sets of *Attributes* be called S, or the *Repository Sample*. We can label the *Attribute* sets by arbitrary integers 0 through $|S|$, or:

$$\{A_i\}_{i \in \{0,1,2...|S|\}}$$

In this way, we can reference a set of *Attributes* (or, an individual *Repository*), by A_i.

Going further, we can also denote an individual *Attribute* of a given *Repository* by:

$$\{A_{(i,\alpha)}\}_{(i,\alpha) \in A' \times \{0,1,2...|S|\}}$$

or, $A_{i,\alpha}$. For example, $A_{10,\Delta}$ would denote the value of the *Issue Close Time* (Δ) of Repository *10*.

To analyze the metrics, the sets of all individual *Attribute* across all *Repositories* are processed by some aggregate function f. The input of this function is the union of all sets of a particular *Attribute* over each *Repository* in the sample. For this analysis, the aggregate functions are: MEAN, MIN, MAX, STDDEV. This analysis is applied to all repositories in the sample S for which data was collected. Each attribute α is analyzed independently, so given:

$$g(\alpha) = f\left(\bigcup_{r=0}^{|S|} A_{r,\alpha}\right)$$

each individual attribute is processed via $g(\alpha)$ and will result in the output of the given aggregate function. The result of each aggregate function when applied to the entire sample set of *Repositories* is shown in Table 1.

2.2.1 Attributes

The individual *Attribute* names, or the elements of A', are described below. For detailed summary statistics of these *Attributes*, see Table 1.

Issues: The number of issues posted to a repository's issue tracker.[8]
Commits: The number of individual source code commits to a repository. It is important to note that a *commit* is a single transaction that changes the code contents of a repository. Actual commit "size" (lines of code modified, number of files changed, for example) is not taken into account.
Closed Issues: The number of issues in a repository in a state of CLOSED at the time of processing.
Open Issues: The number of issues in a repository in a state of OPEN at the time of processing.
Issue Close Time: The average time (in days) taken to close a given issue in a repository. Non-CLOSED issues were not counted.

The average *Issue Close Time* is calculated using the following method: Let $d(x)$ be a function that converts seconds to a Date, and let $s(x)$ be a function that converts a Date to seconds. Let I equal the set of *Issues* in a given repository, with elements comprised

[8] https://github.com/blog/411-github-issue-tracker

of tuples (τ, τ'), where τ is *Issue Open Date* and τ' is *Issue Close Date*. We calculate *Issue Close Time* by:

$$\Delta = d\left(\frac{\sum_{i \in I} s\left(p_2(i)\right) - s\left(p_1(i)\right)}{|I|}\right)$$

Where for each issue i in I, the *Issue Open Date* $s(p_1(i))$ is subtracted from the *Issue Close Date* $s(p_2(i))$, where $p_1(i)$ and $p_2(i)$ denote projections of the tuple, or *Issue Open Date* and *Issue Close Date*, respectively. The result of this is averaged and converted into a Date via $d(x)$, resulting in the average issue close time interval Δ.

Issue Reporters: The number of distinct users that have opened at least one *Issue*.

Committers: The number of distinct users that have summited at least one *Commit*.

3 Results

3.1 Summary Statistics

Variable	Mean	Min	Max	Std. Dev.
Issues	67.91	1	2708	283.65
Commits	548.68	1	61129	3659.2
Closed Issues	64.82	1	2392	269.04
Open Issues	15.68	1	316	41.49
Issue Close Time	28.46	0	515	64.08
Issue Reporters	20.72	1	349	58.33
Committers	26.54	1	4245	245.37

N = 1000 Repositories

Table 1: GitHub summary statistics

Table 1 describes summary statistics of the repository variables described in Section 2.2.1. The most evident feature of the result is the large variance. In fact, for the set of tested variables A, each variable A_j is observed to have a larger standard deviation than the variable mean:

$$\forall j \in A : \overline{j} < \sigma_j \tag{1}$$

All measured variables demonstrated this characteristic. We will refer to this characteristic as Summary Data Assertion 1.

The sample ($N = 1000$) GitHub repositories were selected, at random, out of a total of 5.7 million [1] public GitHub repositories.

3.2 Hypotheses

Hypothesis 1: As the number of Watchers increase, the number of Repository Forks will increase.

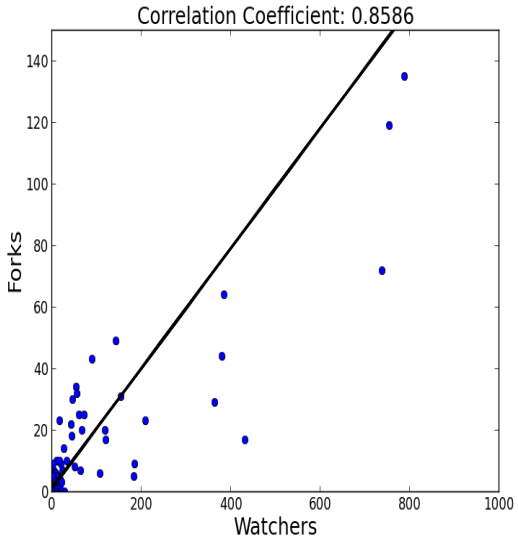

Figure 2: Repository watchers and forks

A correlation coefficient of 0.8586 was observed between the number of Watchers and Forks of a repository. This suggests that a correlation between Watchers and Forks exists.

The data reinforces the hypothesis, as does other research in the area. Even though there are arguments that the number of Forks is the true measure of project success [3], the data suggests that the two measures are related. Also, *Dabbish et al.* have noted that the number of Watches can be a key for others to gauge interest in a project, and thus determine if they should participate [10]. If the number of Watchers of a repository serves as a social cue for participation, even "passive" involvement with a project (Watching) could influence the "active" participation (Forking). This could possibly contribute to the correlation of the two variables. It is important to note, however, that we cannot conclude any causal relationship between the two variables. Concluding whether the social cue for participation is the Forks, the Watchers, or some combination, is not possible given the data and methods used.

Hypothesis 2: As the number Repository Forks increase, the Issue Resolution Time will decrease.

No significant correlation was found between the number of repository forks and issue resolution time, as shown by the data represented in Figure 3. It is important to note, however, that issue resolution time is not necessarily related to software product

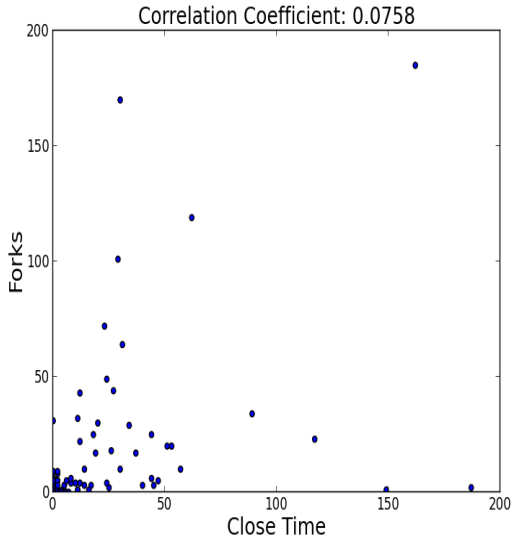

Figure 3: Repository issue close time (in days) and number of forks

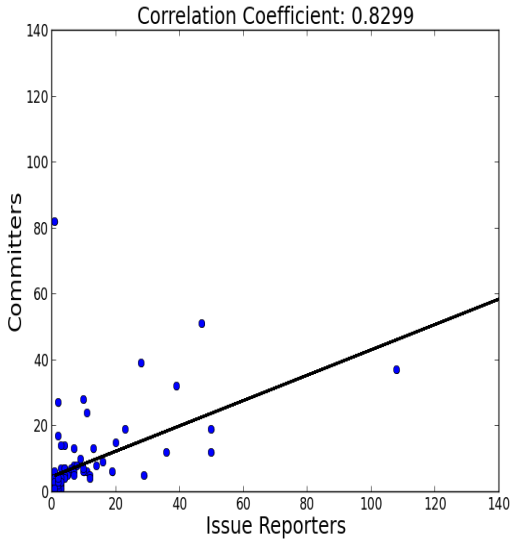

Figure 4: Repository issue reporters and committers

quality in the context of GitHub. We can assert that for several reasons. *First*, GitHub issues are not necessarily product defects. Issues may be feature requests, comments, or general support questions. *Second*, GitHub projects may have primary issue tracking services elsewhere, while using the GitHub issues as secondary services or not at all. *Third*, cultural shifts in how testing is approached in GitHub may have significant impacts on software quality. Pham *et al.* explore an emerging "testing culture" in GitHub, which suggests that GitHub may be facilitating testing by making it more *triable* by lowering entry barriers, and making the process as a whole more *observable*. This aspect is something which could have sizable impacts on software quality but wouldn't necessarily be reflected in this metric [23].

It is possible that other factors, such as issue prioritization, may have a greater impact on issue resolution time. Research does support a relationship [21] between the stated priority and fix interval. As the GitHub issue tracker can only support prioritization by free-text *tags* placed on issues, it is not feasible to programmatically gather issue priority metrics.

Hypothesis 3: There will be more Issue Reporters than Committers by an order of magnitude.
Data does not support an exponentially increasing number of distinct Issue Reporters as compared to distinct Committers. As shown in Figure 4, the number of Issue Reporters is larger than the number of Committers, but not by an order of magnitude. Table 1 reinforces this, as the averages of the two variables are closer than the hypothesis states. This does not seem to completely align with findings in the Apache Server project [20].

Figure 4 does suggest, however, that that the two variables are related. The hypothesis implies that there will be a correlation between the two variables – which is demonstrated by a reasonably strong correlation coefficient of 0.8299. More investigation would be necessary to further examine this finding.

3.3 Research Questions

Research Question 1: Are GitHub projects primarily focused around a small set of core Committers?
The data in Figure 5 represents commit percentage breakdown, or an analysis of developer *Contribution* percentages to a repository. The intent of this measurement is to show how the number of repository commits are distributed among it's committers. The slices represent the distribution of committers that have committed a given percentage of commits (or a *Contribution* percentage, as labeled in the figure) to a project. For example, a small percentage of developers have contributed $95-100\%$ of the commits to a repository, so this is represented as a small slice in the figure. Contrast that to developers that have committed $0-5\%$ of the commits to a repository. This

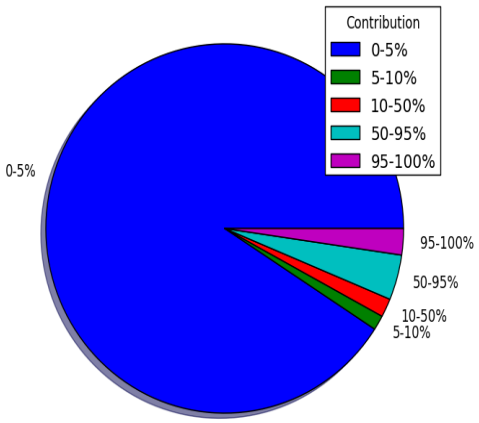

Figure 5: Distribution of individual committer contribution percentage to a repository

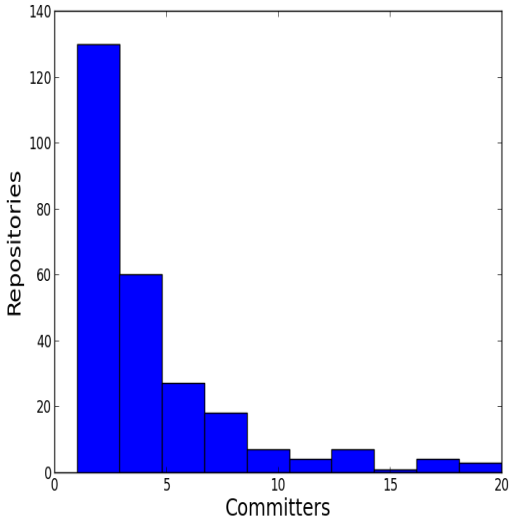

Figure 6: Count of individual committers per repository

type of contribution is common, and represented as a larger slice. The *Contribution* percentages are calculated by:

$$P = \left\{ \frac{C_u}{|\mathscr{C}|} \times 100 \middle| u \in U \right\} \tag{2}$$

Where U is the set of *Committers* to a given *Repository*, $\mathscr{C}$ is the set of all *Commits* to a *Repository*, and C_u denotes sets of *Commits* by an individual *Committers* to a *Repository*, or:

$$\{C_u\}_{u \in U}$$

Each *Committer*, then, will produce a subset of all *Commits* to a a *Repository*, or $C_u \subset \mathscr{C}$.

To find the percentages, or set P of Equation 2, the commits by each user C_u are divided by the total number of commits $|\mathscr{C}|$, yielding the set of contribution percentages P for the repository. Elements of P are then assigned to ranges $0-5\%$, $5-10\%$, $10-50\%$, $50-95\%$, and $95-100\%$ (with shared values inclusive to the larger range). The distribution of this range assignment is plotted on Figure 5.

Most notably, the data shows that a large percentage of developers committed small amounts $(0-5\%)$ of the total number of repository commits. This means that a small core are not entirely monopolizing the commits, but rather opportunity exists for users to contribute, even if in small ways. Instances of developer monopolization of commits to a repository (for example, where a developer contributes $95-100\%$ of the total commits), are shown to be small.

When looking strictly at the number of committers to a repository, however, the notion of a small set of core committers is strongly supported, as the vast majority of repositories have fewer than 10 committers (Figure 6).

Research Question 2: How is GitHub changing the OSS process?

Thung *et al.* showed that the social coding aspects of GitHub – specifically the developer-to-developer connectivity – enabled high collaboration rates [28]. In fact, the developer connectivity was shown to be higher than SourceForge and even Facebook. In a recent interview with GitHub staff, the social aspects of GitHub were referred to as "unique and powerful [4]," which seems to suggest that this is a strong underlying product goal.

Ye also observed that the contribution environment, or "society," is important in providing a framework for developer trust building:

> Only in a society where technical supremacy is highly appreciated can developers acquire good reputations among their peers by displaying their skills through free distribution, and often wider acceptance, of their systems.The good reputation attracts attention, trust, and cooperation from others and lays the foundation for advancing the original developers agenda and the establishment and development of OSS communities. [29]

With this process change can also come cause for concern. The move to Git from a centralized version control system (such as SVN or CVS) is not without its challenges. Differences in philosophy and nomenclature may place added strain on developers [6]. Also, it is not to be assumed that process change is always welcome [12], even if it is an improvement. Developers and organizations have existing processes, SCM tools, and workflow patterns – which all change at a cost.

Despite the potential migration cost, the National Center for Biomedical Ontology,[9] the National Cancer Informatics Program,[10] NASA,[11] and the WhiteHouse[12] are just some organizations that have moved all or portions of development to GitHub. As these and other organizations consider the move to GitHub, the distributed and "social" aspects of the OSS process will continue to evolve.

According to GitHub Education Liaison John Britton, lowering the barrier to participation is an important cultural shift:

> GitHub moved the open source community away from a permission culture. Prior to GitHub, contributing to an open source project was very difficult and often involved asking permission. By allowing people to fork a repository and make changes without asking for permission, GitHub has made contributing to someone else's project much easier than it has ever been. By submitting a Pull Request with suggested changes, project maintainers and contributors have a place to discuss changes to code that is already written and to refine it. [7]

Spending large amounts of time and resources to participate is a detriment to knowledge-sharing in general, not just OSS development [2]. Because of this, we conclude that this permissive culture of knowledge-sharing that GitHub fosters represents an important change in the OSS process.

Research Question 3: Can GitHub be used for more than code artifacts?
The OMG® standardization process is a process by which products strive to produce an interoperability standard by way of an industry standards consortium [13]. The standardization process itself aims to, at its conclusion, produce a set of model artifacts containing sufficient detail such that when interpreted and implemented, produce interoperable software. At a conceptual level, this can be thought of as a software requirements engineering process as much as a standardization process. If so, of the five main tasks of requirements engineering, Elicitation, Analysis and Negotiation, Documentation, Validation, and Management [26], we can show that GitHub is an appropriate forum for many of them.

As a case study, the CTS2 OMG® Specification [9] used a GitHub issue tracker[13] to track specification changes. Elicitation, Analysis and Negotiation were natural fits for the issue tracker. This was chosen in part because the barrier to participation was very small (only a GitHub account was needed), and the issue tracker itself allowed for dialogs in the form of comments. Because customer involvement is critical to these activities [22], ease of contribution was a main concern. OMG® has a predescribed format for requirements Documentation, but using the GitHub issue API,[14] much of this documentation can be automated. Validation and Management are areas of requirements engineering that GitHub has interesting solutions for. First, in order to track (or validate) changes to the specification, when the actual specification is modified, the commit that modified it can be traced to a named issue by simply putting the issue number in the commit log. This level of traceability is powerful, as it allows every change to the specification artifacts to be tracked consistently to an open issue. As the normal OMG® process proceeds, requirements can be managed through "tagging" the issues. By allowing issues to be grouped by arbitrary tags, users are more free to adapt their own work flow. For instance, when community voting on issues needed to be done for CTS2, a simple "ready for vote" tag signified to the voting board exactly what was under consideration

As mentioned, the CTS2 specification made heavy use of the GitHub issue tracker, as well as another OMG® specification, ServD.[15] For open source specifications, or requirements engineering activities, GitHub has shown to be a viable platform.

4 Conclusion

The analysis suggests an *evolution* in process, but not necessarily a large shift. We classify this as an *evolution* because although many of the traditional aspects of the OSS development process remain, there are notable differences. For example, a small sub-

[9] https://github.com/ncbo
[10] https://github.com/ncip
[11] https://github.com/nasa
[12] https://github.com/WhiteHouse
[13] https://github.com/cts2/cts2-specification/issues
[14] http://developer.github.com/v3/issues/
[15] https://github.com/servd/servd-specification

set of core developers, a traditional aspect of OSS development [20, 21, 18], seems supported by both the summary data in Figure 1, and the distribution of committers per repository in Figure 6. Figure 5 shows a significant number of participators, albeit with small contribution amounts, in many repositories. This may indicate a lowered entry barrier for participation, and is consistent with the conclusions reached in Research Questions 1 and 2.

The data does not, however, reflect some other traditional aspects of OSS development. Notably, it was observed that data gathered for Hypothesis 2 (*As the number Repository Forks increase, the Issue Resolution Time will decrease*) and Hypothesis 3 (*There will be more Issue Reporters than Committers by an order of magnitude*) seemed contrary to what the Apache Server report concluded [20]. A possible explanation is the amount of variability in the data collected. Summary Data Assertion 1 (Section 3.1), which is derived from the summary data shown in Table 1, suggests large amounts of variety in the observed variables. This may suggest a wide range of repository purposes, or specifically, that not all repositories on GitHub are intended to be consumable software artifacts for users, or "customers." A large number of repositories may not be intended for customer consumption, but merely as example code, experimentation, or exercises. Given this, and specific reasons outlined in the hypotheses themselves, we cannot state any definitive conclusions regarding these two hypotheses. More research into categorizing GitHub repositories is needed.

The most notable *difference* in OSS process driven by GitHub, considering both the quantitative and qualitative analysis, is the lowered barrier to participation. Qualitative analysis conducted in Research Question 2 suggests a cultural shift [4, 7] in the participation process. The data also supports high levels of participation, especially Figure 5, which shows that many repositories have large numbers of developers committing small amounts of code. This, we conclude, represents an important *evolution* in the OSS development process.

5 Acknowledgments

We thank Gilberto Fragoso, Ph.D. (NIH/NCI), Paul Alexander (NCBO), John Britton (GitHub), and all of the GitHub Staff for interviews, insights and ideas, Harold Solbrig, Craig Stancl, and Dick Hedger for guidance, and Rick Kiefer for his review.

References

[1] GitHub press page. https://github.com/about/press. Accessed: 18/03/2013.

[2] Alexander Ardichvili, Vaughn Page, and Tim Wentling. Motivation and barriers to participation in virtual knowledge-sharing communities of practice. *Journal of knowledge management*, 7(1):64–77, 2003.

[3] Benoit Baudry and Martin Monperrus. Towards ecology-inspired software engineering. *arXiv preprint arXiv:1205.1102*, 2012.

[4] Andrew Begel, Jan Bosch, and Margaret-Anne Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *Software, IEEE*, 30(1):52–66, 2013.

[5] Magnus Bergquist and Jan Ljungberg. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, 11(4):305–320, 2008.

[6] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. The promises and perils of mining git. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pages 1–10. IEEE, 2009.

[7] John Britton. email interview, Apr. 17 2013.

[8] Scott Chacon, Junio C Hamano, and Shawn Pearce. *Pro Git*, volume 288. Apress, 2009.

[9] Common terminology services 2 (CTS2). http://www.omg.org/spec/CTS2/1.0, 2012.

[10] Laura Dabbish, Colleen Stuart, Jason Tsay, and James Herbsleb. Leveraging transparency. 2013.

[11] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[12] Brian De Alwis and Jonathan Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Cooperative and Human Aspects on Software Engineering, 2009. CHASE'09. ICSE Workshop on*, pages 36–39. IEEE, 2009.

[13] Object Management Group. *Policy and Procedures of the OMG Technical Process, pp/12-12-01*, 2012.

[14] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[15] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[16] Cris Kobryn. Uml 2001: a standardization odyssey. *Communications of the ACM*, 42(10):29–37, 1999.

[17] Bruce Kogut and Anca Metiu. Open-source software development and distributed innovation. *Oxford Review of Economic Policy*, 17(2):248–264, 2001.

[18] Sandeep Krishnamurthy. Cave or community?: An empirical examination of 100 mature open source projects. *First Monday*, 2002.

[19] Karim Lakhani and Robert Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. 2003.

[20] A. Mockus, R.T. Fielding, and J. Herbsleb. A case study of open source software development: the apache server. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 263–272. IEEE, 2000.

[21] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[22] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 308–313. IEEE, 2003.

[23] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. Creating a shared understanding of testing culture on a social coding site.

[24] Andreas Platschek and Nicolas McGuire. Floss for safety: Mastering mission critical development with git.

[25] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

[26] Ian Sommerville and Gerald Kotonya. *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[27] Diomidis Spinellis. Git. *Software, IEEE*, 29(3):100–101, 2012.

[28] Ferdian Thung, Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, et al. Network structure of social coding in github. In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, pages 1–4, 2013.

[29] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation of open source software developers. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 419–429. IEEE, 2003.