Understanding How Companies Interact with Free Software Communities

Jesus M. Gonzalez-Barahona, Universidad Rey Juan Carlos
Daniel Izquierdo-Cortazar, Bitergia
Stefano Maffulli, OpenStack
Gregorio Robles, Universidad Rey Juan Carlos

When free, open source software development communities work with companies that use their output, it’s especially important for both parties to understand how this collaboration is performing. The use of data analytics techniques on software development repositories can improve factual knowledge about performance metrics.

FREE, LIBRE, OPEN source software (FLOSS) communities can be very complex and difficult to understand, so quantitative, neutral information about them becomes valuable, especially when they’re large and involve competing companies. Fortunately, many FLOSS projects operate in open, public development repositories, which has facilitated the progress of new analytics techniques to study them. In turn, these techniques have started to produce useful results for both practitioners and other stakeholders.

In particular, studies of company participation in large projects have started to raise industrial interest: FLOSS foundations want to learn about company participation in their projects, and the companies want to know more about corporate activity in the projects on which they rely. Here, we present two studies in this area—by LibreSoft, a research group specializing in the quantitative analysis of software development, and Bitergia, a LibreSoft spin-off company focused on software analytics services—into company activity in OpenStack and the fairness of WebKit’s review process.

Characterizing Company Participation

Although corporate contributions have always played a role in FLOSS projects, the emergence of “communities of companies” is driving an interest in analyzing company behavior. Although this new kind of community admits individual contributions, it clearly prioritizes corporate interests, and participating companies, which can be commercial competitors, employ most of the developers. Researchers can study several aspects of this participation:

- activity, or how companies contribute to the project in code changes, bug fixes, or participation in discussions;
- neutrality, or how neutral the project stays with respect to accepting contributions from or fixing bugs reported by companies; and
- collaboration, or how companies work in the same areas, collaborate to fix bugs, or take joint decisions.
From any of these analyses, we can learn a lot about company strategies—for example, code changes are linked to functionality; participation in bug fixing signals either long-term involvement or casual participation; development team size and turnover indicates investment in the project; and lack of neutrality shows a lack of interest in getting contributions from other parties. Similarly, collaboration in specific areas uncovers common interests.

Any participating company will want to know its competitors’ areas of interest or the community’s neutrality on certain subjects. Recent studies provide some examples, such as the Linux Foundation’s research into contributions to the Linux kernel and the dashboard that the Eclipse Foundation maintains of detailed information about company activity (http://dash.eclipse.org).

**General Methodology**

FLOSS projects have used software support for their development processes for many years, including source code management systems (SCMs) such as CVS, Subversion, and git; issue-tracking systems (ITSs), such as Bugzilla, Trac, and those provided by various forges; and mailing lists and forums, such as Mailman. These tools are usually integrated in software development forges such as SourceForge, Google Code, and GitHub, or are maintained by large communities such as Apache or Eclipse for their own use.

Most of these systems host detailed information about project activity. SCMs, for example, keep data about all the changes to the source code (commits), including meta-information such as who performed the change and when. ITTs keep details for all tickets, including when they were filed, by whom, and when, and then who did the updating (in addition to the updates themselves). In most cases, third parties can retrieve this wealth of information.

For our studies, we used the MetricsGrimoire toolset (see the sidebar) to retrieve information from the relevant repositories of the analyzed projects and store it in a database. Once this data was ready, we were able to query and analyze the database (see Figure 1).

To validate our studies, we shared our results with experts in companies deeply involved in the analyzed projects. For OpenStack, this included six experts from four organizations, including the OpenStack Foundation. For WebKit, we worked with five experts in three companies. All of them helped interpret the results, refine the analytics, and understand the value for their companies.

**OpenStack**

Some of the parties involved in the OpenStack project wanted to learn about how companies were contributing to the development effort, so we conducted a study to characterize company participation by tracking employee activity. This posed some problems:

- Company affiliation isn’t always easy to determine; sometimes this information isn’t publicly available or up to date. In addition, including all the identities that a developer might use is challenging.
- It isn’t easy to agree on measures that reflect a company’s contribution. A balance must be found between what would be desirable, such as contributed effort in maintenance or new functionality, and what’s feasible to calculate.

For OpenStack, we used heuristics and manual inspection to solve the first problem. We started with tentative affiliations based on domain names.
To address the second problem, we studied development practices, ultimately deciding that two estimators were good enough to characterize company participation and simple enough to be calculated: number of commits per company and authors per company. Different activities can produce very different commit patterns, and not all commits are equal, but the consulted experts agreed that this would be a suitable way to start.

The process involved the following steps:

- Retrieve, using CVSAnalY, information from the SCM (git) system. In this case, that meant 30,829 records since February 2010, when the repository was opened, to September 2012, a bot authored 4,954 commits (it was a software tool doing some maintenance tasks), leading to a total of 25,875 commits authored by developers.
- Identify authors using the “author” field in git commit records, which contains email addresses and names. Then, find unique identities using simple heuristics such as email addresses with exactly the
same name, similar names with the same email address, or same user identifiers in email addresses with different domains.

- Identify companies by using the domain part of the email addresses. Use manual inspection, such as looking for information on the Web, or ask people involved in the OpenStack community to identify the affiliations of people with companies, as well as the periods of those affiliations.

- Produc consolidated data per company using the activity of developers as found in the SCM database.

As a result of this process, we built a dashboard with different kinds of charts and tables: aggregated data for the project, for the release periods, for each of the subprojects, for each of the companies, for each of these cases over time, and so on. The complete dashboard provides a multifaceted view of how companies participate in the project—and that participation’s evolution.

Our analysis showed that although a single company had performed most of the activity in the project (see Figure 2), most components were quickly evolving with additional participants (see Figure 3). This single result was very valuable to the OpenStack community because it showed how the project no longer relied on a single company and that many others were now investing significant efforts. Although experts working on the project already knew that new companies were entering with energy, the study showed that for some key components, such as Nova, contribution leadership had already changed, and new entrants accounted for a large majority of commits.

Although these results can’t be considered a definitive picture of company participation, they led to more transparency, with a clearer representation of corporate involvement in the project.

Further steps to produce a more detailed view might include the study of activity in fixing bugs or in design and support discussions. Such studies are now under way, with the support of the OpenStack Foundation.

Other researchers have used a similar tool, gitdm, for other studies. The main difference with our study (other than the toolset) lies in the attention devoted to identifying affiliations and in the separate analysis of modules.

WebKit
Many large FLOSS projects have policies that enforce code review to verify
whether changes to source code meet certain criteria. Code reviewers accept new contributions, ask contributors to modify them before being accepted, or decline them. Project participants want this process to be based purely on technical grounds, but corporate relationships could affect it: both authors and reviewers work for companies that could be competing. In the WebKit case, we did an analysis to determine how some parties treated companies during the review process.

The code review process in WebKit starts when developers propose a change to the source code. Usually, they create a ticket in the ITS, attach the change, and ask for a review using a flag in the ticket. A reviewer inspects the change, decides if it complies with project policies, accepts or declines it, flags it accordingly, and annotates the reason if declined. After fixing any problems, the developer usually asks for a new review. The process continues until the change is accepted or definitely declined.

To track code reviews, we followed flags in tickets. We addressed the problems to identify author and reviewer affiliations in a similar way to the OpenStack case. To capture delays in the review process, we used the concept of iteration: we counted an iteration every time a change request was accepted or declined because it’s the “basic unit” of code review interactions. A directly accepted request goes through one iteration, whereas one that’s rejected three times and finally accepted follows four iterations.

**FIGURE 4.** Part of a company’s code review. For each developer, the figure shows the (a) number of accepted contributions (log scale) and total number of iterations and (b) a histogram of developer ratios (number of iterations by number of accepted contributions). The table (c) shows parameters characterizing the developers’ iteration ratio. Slope refers to the slope of the linear regression (R-squared: 0.96), which takes into account the different quantities of accepted contributions (and thus, experience) per developer.

<table>
<thead>
<tr>
<th>Contributors</th>
<th>Accepted</th>
<th>Iterations</th>
<th>Ratio</th>
<th>Ratio per developer (mean)</th>
<th>Ratio per developer (median)</th>
<th>Ratio per developer (slope)</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>664</td>
<td>1,865</td>
<td>2.80</td>
<td>3.35</td>
<td>3.30</td>
<td>2.40</td>
</tr>
</tbody>
</table>

**THE WEBKIT PROJECT**

WebKit (www.webkit.org) is a FLOSS project producing a Web browser engine that, up until April 2013, was the core of many popular Web browsers: Safari (Apple), Chrome (Google), and those of several mobile operating systems (iOS, Android, Tizen, and so on). It started when Apple forked the KHTML and KJS libraries from the KDE project in 2001. For several years, the descendants of KHTML and KJS were FLOSS, while the rest of WebKit was proprietary. In 2005, all WebKit code was released as FLOSS, and a community-based project was born, with many companies and individual developers joining. In early 2013, both Apple and Google were leading the development, until Google decided to fork it, launching Blink. WebKit had more than 3 million lines of code as of December 2012, mostly C++ and JavaScript.
The number of iterations per developer wasn’t enough to characterize delay because it’s correlated with experience, and we wanted to measure it as independently of experience as possible. We used the ratio between the number of iterations and accepted changes (which is a proxy for experience). Unfortunately, the distribution of ratios per developer in a company is usually skewed: most have relatively low ratios, but a few of them, usually those newer to the project, have larger ones. This makes it difficult to select a single parameter that’s significant enough for comparing companies.

We addressed this problem by using four parameters: the ratio for all the company’s contributions, the mean and median of the ratios per developer, and the slope of the linear interpolation of such ratios per developer. We considered two companies with similar parameters to be treated in a similar way by the code review process. Figure 4 shows an example of information characterizing a company that could be compared to any other in the project.

We found the review process to be fair in general; Figure 5 shows an example that represents one of the parameters for a set of companies over two consecutive years. The experts we consulted consider this kind of chart to be a good tool for quickly tracking how the review process treats companies. When produced for different periods, the charts can also be used to detect improvement or deterioration in specific cases. In general, the information provided led to productive discussions about fairness and how to improve it. However, experts also identified some complementary information that could be useful for companies, such as time to first review and time to commit.

Over the course of our studies, we found that complex situations can’t be characterized with a single number, or even with a small set of them. Participation in code development is more than counting commits, and fairness in the review process is much more than counting iterations. Nonetheless, more research and consensus among project participants is needed to define measures that are both significant and dependable—key factors for characterizing and monitoring FLOSS projects. Commits are far from being a perfect measure of activity, and iterations capture only a part of the review process, but both are simple enough to be easily understood and provide useful information even to experts.

In most FLOSS projects, anyone can make an analysis without the analyzed community being involved, thanks to the public availability of the data sources. A part of the transparency that FLOSS communities want for their development process lies in this opportunity for anyone to come and analyze it. Unfortunately, FLOSS projects still aren’t taking great advantage of analytics, which is reflected in the lack of certain information that
FOCUS: THE MANY FACES OF SOFTWARE ANALYTICS

would make analytics studies easier, such as detailed affiliation data. In our case, a large fraction of the effort was in identifying affiliations, information that projects could easily maintain. But on the bright side, as studies show their usefulness, this missing information is starting to be collected and updated. Some projects are deploying development dashboards and caring more about how to store traces of the development process for later analysis. Companies specialized in providing FLOSS analytics services are also emerging. In the coming years, we can expect a trend toward including analytics in the design and tracking of development policies, as well as in the evaluation of many aspects of FLOSS communities.

As a final note, it’s worth mentioning that company participation hasn’t been a key research area of software analytics to date, but it should be: it’s the answer to the “so what?” question posed in the call for articles for this special issue. Companies are interested in learning about how they interact in FLOSS communities, and software analytics provide techniques that help understand these interactions. The exact versions of the tools and data sources, as well as the details and scripts used to process and analyze the information, appear on this article’s companion website, along with the reproducibility elements and methodological details (http://gsyc.urjc.es/~jgb/repro/2013-software-experiences) that we built following the criteria presented elsewhere.

Acknowledgments

Jesus M. Gonzalez-Barahona, Daniel Izquierdo-Cortazar, and Gregorio Robles helped design and develop the studies described here as a collaboration between the LibreSoft team at Universidade Rey Juan Carlos and Bitergia. Stefano Maffulli participated in the design of the OpenStack study and provided specific feedback on the ideas and discussions presented in the article. He was an employee of RackSpace while he provided support for the study on OpenStack. Jesus M. Gonzalez-Barahona and Gregorio Robles were funded in part by the European Commission under project ALERT (FP7-IST-25809) and by the Spanish Government under project SobreSale (TIN2011-28110). We thank the developers of OpenStack and WebKit for enabling this study by making their repositories publicly available.

References


ABOUT THE AUTHORS

JESUS M. GONZALEZ-BARAHONA is a teacher and researcher at Universidad Rey Juan Carlos; he collaborates with Bitergia, a software development analytics company, and is interested in understanding free/open source software development, finding ways to improve its efficiency, and sharing this knowledge. Gonzalez-Barahona is a member of the IEEE Computer Society and the Asociación de Técnicos en Informática. Contact him at jgb@gsyc.es.

DANIEL IZQUIERDO-CORTAZAR is a researcher and one of the founders of Bitergia, a company that provides software analytics for open source ecosystems. Izquierdo-Cortazar has a PhD in computer science from the Universidad Rey Juan Carlos, Madrid. Contact him at dizquierdo@bitergia.com.

STEFANO MAFFULLI is OpenStack’s community manager. His interests include free software and open source systems, an area that he’s explored from his early days as a pre-sales engineer and product manager at Italian GNU/Linux distributor MadeInLinux to Italian Chancellor of the Free Software Foundation Europe, where he also created the FSFE Fellowship participation program. Contact him at Stefano@openstack.org.

GREGORIO ROBLES is associate professor at the Universidad Rey Juan Carlos. His research interests involve software engineering in free/libre open source software, mining software repositories, and technology-enhanced learning. He’s member of IEEE. Contact him at grex@gsyc.es.

Acknowledgments

Jesus M. Gonzalez-Barahona, Daniel Izquierdo-Cortazar, and Gregorio Robles helped design and develop the studies described here as a collaboration between the LibreSoft team at Universidade Rey Juan Carlos and Bitergia. Stefano Maffulli participated in the design of the OpenStack study and provided specific feedback on the ideas and discussions presented in the article. He was an employee of RackSpace while he provided support for the study on OpenStack. Jesus M. Gonzalez-Barahona and Gregorio Robles were funded in part by the European Commission under project ALERT (FP7-IST-25809) and by the Spanish Government under project SobreSale (TIN2011-28110). We thank the developers of OpenStack and WebKit for enabling this study by making their repositories publicly available.

References


