Noname manuscript No.
(will be inserted by the editor)

On the Challenges of Open-Sourcing Proprietary Software Projects

Gustavo Pinto · Igor Steinmacher · Luiz Felipe Dias · Marco Gerosa

Received: date / Accepted: date

Abstract The open source software (OSS) movement has become widely recognized as an effective way to deliver software. Even big software companies, well-known for being restrictive when it comes to publishing their source code artifacts, have recently adopted open source initiatives and released for general use the source code of some of their most notable products. We conducted an exploratory study on merits of the widespread belief that open-sourcing a proprietary software project will attract external developers, like casual contributors, and therefore improve software quality (*e.g.,* "*given enough eyeballs, all bugs are shallow*"). By examining the pre- and post-migration software history of eight active, popular, non-trivial proprietary projects that became open source, we characterize the phenomenon and identify some challenges. Contrary to what many believe, we found that only a few projects experienced a growth in newcomers, contributions, and popularity; furthermore, this growth does not last long. The results from the study can be useful for helping software companies to better understand the hidden challenges of open-sourcing their software projects to attract external developers.

Keywords Open source software · Proprietary software · Community Engagement · Open Collaboration · Popularity

Gustavo Pinto
E-mail: gpinto@ufpa.br

Igor Steinmacher
E-mail: igorfs@utfpr.edu.br

Luiz Felipe Dias
E-mail: luizdias@alunos.utfpr.edu.br

Marco Gerosa
E-mail: marco.gerosa@nau.edu

1 Introduction

The open source movement has become a popular distribution model for gold-standard software [12,40]. More interestingly, however, is the fact that several software companies, otherwise well-known for being restrictive when it comes to publishing their software artifacts, have begun not only using open source software, but also promoting open source initiatives [2]. In the last few years, Google released more than 900 open source projects, amounting to more than 20 million lines of source code.[1] Similarly, Microsoft and Apple released some of their products as open source projects. Notable examples include the `swift` programming language, which was forked more than 1,900 times in the first week after open-sourcing (as further discussed in Section 4). The increasing use and promotion of open source might indicate a strong interest from external developers, empowering them to "*take their code-based innovations to a whole new level and explore new depths,*" as stated by a `roslyn` — the .NET Compiler — member [27].

At a first glance, one might note that open-sourcing a proprietary project yields several benefits. For instance, open source software may: (1) foster external contributions [12]; (2) create new and innovative ideas [26]; and, possibly, (3) increase the pace of change [37]. In fact, there is a recurring belief that popular open source projects are more prone to attracting new source code contributors [9], which in turn increases a project's diversity and enhances problem-solving skills [52]. Moreover, open-sourcing might also improve software quality, as already envisioned by Linus's Law, which says "*Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone*" [40], which was empirically investigated by Meneely and colleagues [31]. However, due to the lack of literature focusing on the transition from proprietary software to open source software, it is unclear whether these well-known assertions from the traditional open source literature also hold for proprietary software that transitions to open source.

The potential benefits of open-sourcing, if achieved, do not come without cost. Since open-sourcing involves engagement and collaboration [40], going open source means creating, maintaining, and fostering a community around technology; *e.g.,* attracting new developers who are eager to implement new features [8,15,53]. Moreover, when a software company open-sources a software project, the software development team might face additional overhead due to the non-trivial costs of: (1) refactoring the code base to be comprehensible; (2) setting up a development website and communication channels; and (3) writing documentation for newcomers [12]. It is not uncommon for newcomers to find a project too difficult to understand [47] or contribute to [37]. As we shall see, some projects might have perceived the hidden cost of open-sourcing: as one maintainer said, "*we've invested a huge amount of effort into making Zulip's*

[1] `https://developers.google.com/opensource/projects`

development environment really easy to setup and use compared to other large webapps.".

To better understand this complex landscape, in this paper we explore the process of open-sourcing proprietary software projects. For this study, we selected multiple cases of software projects that published their source code on GitHub. As of 2016, GitHub was the most popular collaborative coding environment in the world, with over 38 million software repositories[2] and more than 15 million contributors. Indeed, GitHub emerged as the *de facto* place for software developers to get acquainted with, and eventually contribute to, open source projects [51]. To analyze the transition of proprietary software to open source through GitHub, we curated a list of 8 popular, active, non-trivial, and diverse projects (see Section 3 for details). The list includes `swift` from Apple, `hhvm` from Facebook, `roslyn` from Microsoft, and `atom` from GitHub. Using data and meta-data acquired from software repositories, we analyzed collaboration (such as the number of newcomers, active contributors, and contributions) and popularity metrics (such as the number of stars and forks) [6, 51], before and after the migration to open source. To cross-validate our findings, we conducted a survey with the newly open source communities.

Considering the projects analyzed, the main findings of the paper are the following:

1. In spite of widespread belief, the rise of contributions is not straightforward; only 3 out of 8 studied projects presented considerable growth of contributions.
2. A "newcomers wave:" a high number of newcomers make a few contributions to the project once it becomes open source, but do not return.
3. The number of pull requests (PR) increases soon after the transition to open source, and 7 out of the 8 studied projects merged a steady flow of them into the main repository. By inspecting pull requests, we found that 82% of the PRs submitted to `hubot` are from external members (37% for `atom`), which is not always the case in OSS projects [9,14]. A manual investigation of ∼480 PRs confirmed this fact.
4. A burst on the number of change requests (issues) occurred after the analyzed projects became open source (*e.g.,* `roslyn` received about ∼500 new issues in the first two months after open-sourcing). Analyzing these requests, we found that, for our cases, this is due to: (1) the migration process from another issue tracking system; and (2) several change requests made by external users.
5. A growth in popularity (in terms of the number of stars) of the studied projects (4/8 projects faced a fast or viral growth) when compared to the top-2,500 most starred public projects on GitHub. We also identified a fast growth of forks for 4 out of the 8 studied projects.
6. Although the studied projects present a fast growth of forks, only 6% of the forks remained active (that is, were used to submit contributions).

[2] `https://github.com/about/press`

2 Related Work

In this section, we describe studies related to proprietary software projects on GitHub, GitHub's social features, social factors in the retention of new developers, and the phenomenon of casual contributors.

Proprietary Software Projects on GitHub. Kalliamvakou *et al.* [22] examined how organizations with projects producing proprietary software use GitHub, finding that these projects apply typical software development practices used in OSS projects, including reduced communication, independent work, and self-organization. In our study, we also analyzed proprietary projects that use GitHub; however, we are interested in those projects that became open source, while Kalliamvakou and colleagues' work focused on the use of GitHub infrastructure to produce closed-source, proprietary software in private repositories.

Community Building on GitHub. Many studies focus specifically on GitHub's social features, which allow developers to track each other's activities and thus form detailed impressions of their social and technical abilities and behavior. Marlow *et al.* [28] found that developers form impressions of users and projects based on signals (skills, relationships, etc.) in GitHub profiles. Dabbish *et al.* [8] investigated the influence of transparency on the behavior of GitHub participants and noticed that the number of watchers of a project can serve as a social cue to attract developers. Similarly, Tsay *et al.* [51] found that developers use both technical and social information to evaluate contributions to open source software projects. Other than that, community size is often evidenced as an indicator of an OSS project's success. In their interviews with maintainers of GitHub projects, McDonald and colleagues [30] found that the features provided by GitHub are cited as one of the main reasons for the increase in contributions and contributors to an OSS project. These studies focus on social coding environment features as drivers for attracting developers and for generating signals that form impressions about projects and developers. However, they do not investigate how the migration to social coding environments influences the onboarding of new members and the number of contributions the projects receive.

Influence of Social Factors on newcomer retention. Some studies in the literature focused on analyzing the influence of social factors on the retention of new developers on OSS projects [19,20,24]. These studies analyzed social networks (*e.g.,* mailing lists) in order to understand (1) with whom new developers collaborate; and (2) how the network evolves over the years. Moreover, Jensen *et al.* [18] analyzed four projects to understand if newcomers are quickly responded to when they reach out the community, if their gender and nationality impact the kind of answer that they receive, and if the treatment they receive is similar to the ones that other members of the project received. Although these studies focus on the relationship between social aspects and the onboarding and retention of contributors, they do not analyze whether social coding environments foster contributions to OSS projects.

Casual Contributors Phenomenon. Some papers explore the phenomenon of casual contributors (or drive-by commits) in the context of social coding environments. Several authors have acknowledged the existence and the growth of this behavior [14,35,36,37,52]. However, they do not analyze the migration of projects to such environments.

3 Research Method

In this section, we describe our research question (Section 3.1), the studied projects (Section 3.2), and the employed research approach (Section 3.3).

3.1 Research Question

To guide our research, we investigated the following important but overlooked research question:

- **RQ.** What can companies expect when open-sourcing a proprietary software project?

This broad research question is aimed at understanding factors such as: (1) the joining and retention of new contributors in new open source communities; (2) the level of activity before and after migration; and (3) the growth of interest in these communities. Therefore, we studied several metrics related to *collaboration* and *popularity*. The collaboration metrics studied are:

1. The number of newcomers that placed their first contribution to the project.
2. The number of active contributors in a given time window.
3. The number of source code contributions.
4. The number of pull requests received, merged, and closed.
5. The number of issues opened and closed.

These metrics are commonly found in the open source literature (*e.g.,* [9, 37]). By "newcomer," we mean those contributors who placed their first *source code contribution* to a project, and who previously had not contributed to the project. In this scope, a contributor is no longer considered a newcomer after placing their first contribution. We made no distinction between whether newcomer's contributions were intended to improve the source code, for documentation, or translate code. By "active contributor," we mean a contributor who placed at least one contribution during the time frame. We also studied popularity metrics, including:

1. The number of stars.
2. The number of forks.

We chose these metrics because they describe how developers appreciate or are interested in a given open source project [6,51]. These measures are known as a proxy for *project popularity*, as they reflect the project's activity levels and developer interest. This is a common approach for selecting open source projects to investigate [3,37,39].

3.2 Selected Projects

In this study, we characterize our sample as multiple cases of popular proprietary software systems that became open source at a given point in their life-cycle. To define our sample, we searched for blog posts, newsletters, and README files for evidence clearly indicating that a proprietary project open-sourced its code. We imposed the sole restriction that the project must have retained its software history, since only then could we compare our metrics before and after the transition. We found only 8 projects that met these criteria:

1. `atom`[3], a cross-platform text editor. GitHub started its development in 2011, and open-sourced it in May 2014 [43]. It is mostly written in CoffeeScript.
2. `hhvm`[4], a virtual machine designed for executing programs written in Hack and PHP. Facebook started its development in late 2009, and open-sourced it in February 2010 [11]. It is mostly written in C++.
3. `swift`[5], a new programming language. Developed by Apple, it was started in July 2010, and was open-sourced in December 2015 [23]. It is mostly written in C++ and Swift.
4. `roslyn`[6] is the .NET Compiler, which provides code analysis APIs. Microsoft started its development in March 2014, and open-sourced it in November 2014 on Codeplex. In 2015, `roslyn` moved to GitHub [7]. It is mostly written in C# and Visual Basic.
5. `storm`[7], a distributed realtime computation system. Twitter started its development in 2010, and open-sourced it in September 2011 [29]. Later on, `storm` moved again to an Apache repository. It is mostly written in Java.
6. `zulip`[8] is a group chat. Dropbox started its development in August 2012, and open-sourced it in September 2015 [1]. It is mostly written in Python and JavaScript.
7. `hubot`[9] is a customizable life-improvement robot. GitHub started its development in August 2011, and open-sourced it in October 2011 [10]. It is mostly written in CoffeeScript.
8. `plotly.js`[10] (`plotly` for short), is a high-level, declarative charting library. Plotly (the company) started its development in June 2012, and open-sourced it in November 2015 [38]. It is entirely written in JavaScript.

These projects are relevant along different *dimensions* [32] (data collected on July-2016):

[3] `https://github.com/atom/atom/`
[4] `https://github.com/facebook/hhvm/`
[5] `https://github.com/apple/swift/`
[6] `https://github.com/dotnet/roslyn/`
[7] `https://github.com/apache/storm`
[8] `https://github.com/zulip/zulip`
[9] `https://github.com/github/hubot`
[10] `https://github.com/plotly/plotly.js`

- ***Popular***: These projects have, on average, 14.5K stars on GitHub (max: 32K, min: 3K), and more than 2.4K forks (max: 5K, min: 410).
- ***Non-Trivial***: Most of the projects have hundreds of thousands of lines of code (Max: 498K, Min: 2K, Mean: 241.5K). Five out of the eight studied projects use more than one programming language. On average, they have about 5 years of historical records.
- ***Active***: These projects received an of average 429 contributions over the last 12 months (Max: 940, Min: 23), with an average of 64 Pull requests (PRs) per month. These contributions are performed by more than 1,289 different source-code contributors.
- ***Diverse***: They span different domains, varying from text editors, graphic frameworks, programming languages, and virtual machines. Yet, they are written in different programming languages, including Java, C++/C#, JavaScript, and CoffeeScript.

Table 1 summarizes the characteristics of the studied software projects. We used the `cloc` utility to calculate the Lines of Code (LOC). It includes code from all the languages in which a project was developed, as well as blank lines and commented lines. `plotly` has the greatest number of lines of code (498K), whereas `hhvm` has the greatest number of unique contributors (467), `swift` has the greatest number of commits (35K), and `roslyn` has the greatest number of PRs (4K). With regard to their popularity, `swift` has the highest number of stars (32K), and `atom` has the highest number of forks (5K). `hhvm` and `swift` are the oldest (6 years of software development each). Finally, as we can see in this table, some projects lack issues data. This is because the projects `swift` and `storm`, although hosted on GitHub, do not use GitHub's issue tracking system. Figure 1 depicts each distribution.

Table 1 The diversity of our selected projects. LoC means Lines of Code. PR means Pull requests. Age is presented in years. (Data collected in July-2016)

Projects	*LoC*	*Committers*	*Commits*	*Issues*	*PRs*	*Stars*	*Forks*	*Age*
`atom`	51K	272	28K	8K	2K	29K	5K	5
`hhvm`	131K	467	17K	5K	2K	13K	2K	6
`swift`	406K	251	35K	—	3K	32K	5K	6
`roslyn`	97K	140	11K	5K	5K	5K	1K	2
`storm`	352K	257	6K	—	1K	3K	2K	5
`hubot`	119K	122	12K	562	632	11K	2K	4
`zulip`	2K	232	2K	583	640	4K	758K	5
`plotly`	498K	74	10K	384	346	5K	506	4

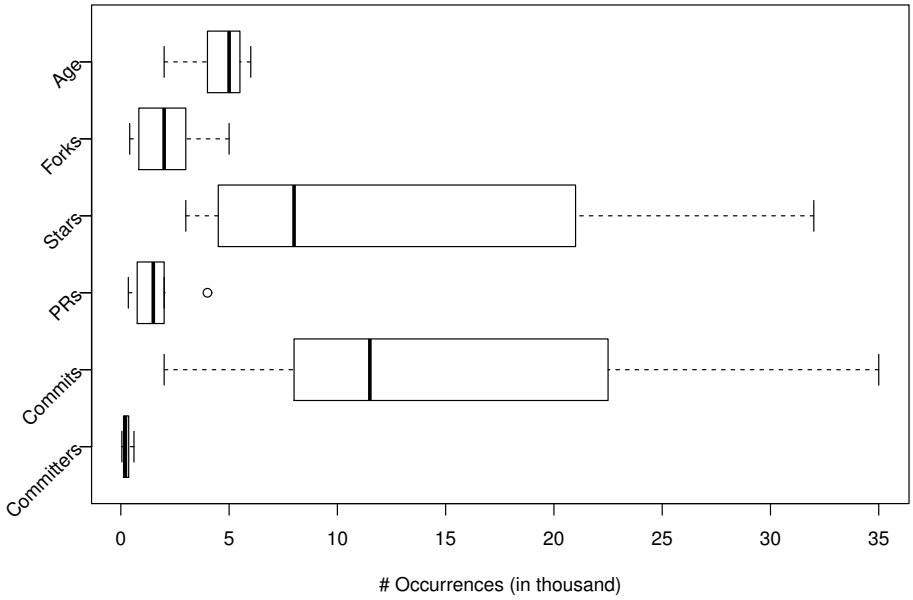

Fig. 1 Selected projects. The number of occurrences are presented in thousands, except 'Age'.

3.3 Research Approach

We conducted a two-step approach: investigating data and meta-data of the studied projects; and employing questionnaires seeking additional evidence.

3.3.1 Collecting Data From the Repositories

For each studied software project, we used the GitHub API to collect the following projects' characteristics:

- The number of newcomers that have joined the project over time, number of contributions (*i.e.*, commits) to the project, and the number of active contributors in a given time window.
- The number of pull requests that are either *opened*, *closed*, or *merged*.
- The number of issues that are either *opened* or *closed*.
- The number of *stars* and *forks*.

Next, we compared the distributions of each collected metric with respect to the migration phases of the studied projects. In order to verify whether the migration is associated with an impact on the collaboration metrics, we compared the number of newcomers of a project before and after its migration to open source. We were unable to apply this approach to the popularity metrics (stars and forks), since the selected projects do not have representative

data before the transition to GitHub. We used a disambiguation technique proposed [4] to match users with different email addresses.

3.3.2 Survey Design and Application

After collecting and analyzing the data from the repositories, we designed a survey aimed at gathering further insights. Based on the recommendations of Kitchenham *et al.* [25], we followed the prescribed phases: planning, creating the questionnaire, defining the target audience, evaluating, conducting the survey, and analyzing the results. Before sending the survey, we discussed the three general questions with two specialists. The feedback obtained from this discussion helped us to better clarify and rephrase some questions.

We sent our questionnaire by creating issues in the issue tracker of the projects on GitHub. This approach works as an effective feedback loop between the survey authors and the survey respondents, making it easy to further clarify questions. For the projects that do not use the issue-tracking system (or do not use it for the purpose of posing questions), we sent the questionnaire to their official mailing lists. The URLs for the questionnaires are presented in Table 2. For each project, we opened one issue. Since we used GitHub issues as a means to deliver the questionnaire, we employed a criterion to check the credibility of the respondents. We inspected the list of contributors of each project to verify whether the respondent was indeed active in the project, and found that all participants in our survey were indeed active project contributors.

Table 2 URL for the questionnaires submitted to each project

Project	Sent via	URL
atom	Mailing list	http://bit.ly/2poYuYT
hhvm	Issue tracker	https://github.com/facebook/hhvm/issues/7122
swift	Mailing list	http://bit.ly/2poSFKZ
roslyn	Issue tracker	https://github.com/dotnet/roslyn/issues/11714
storm	Mailing list	http://bit.ly/2ptfkXq
hubot	Issue tracker	https://github.com/hubotio/hubot/issues/1244
zulip	Issue tracker	https://github.com/zulip/zulip/issues/1968
plotly	Issue tracker	https://github.com/plotly/plotly.js/issues/712

Survey Design. Our survey had three main questions.

> *Question 1:* What motivated the project to open its source code on GitHub? How do you evaluate the benefits of this migration?

In our first question, we were motivated to understand the main reasons why our studied projects became open source. Although one might reasonably believe that opening the source code at GitHub would have clear benefits, (*e.g.,* using a decentralized version control system), we believe that asking developers might also uncover reasons that are not straightforward to hypothesize. We

asked whether they perceived any benefits from this migration process and, if so, how they evaluate our data.

> *Question 2:* Does this snapshot make sense? Did you find any inconsistency in the data?

For each project contacted, we also attached an overview figure (the same one presented in Figure 2) describing the evolution of the project in terms of the number of newcomers, active contributors, and contributors. This question aims to validate our data collection procedure. As we shall see in Section 6, it is not always possible to differentiate contributors, particularly when moving from a centralized to a decentralized version control system.

> *Question 3:* Do you have any internal policies to promote/attract/retain new contributors? If so, do they succeed?

In our third question, we intended to discover whether the selected projects maintain any internal policy to deal with newcomers. This question is particularly relevant to our group of studied projects, since their popularity, along with GitHub's social features, make them suitable for attracting new contributors. This question also aimed to revalidate one of the findings of a recent study, which suggests that project maintainers fail to encourage new contributors [37].

In addition to these three general questions posed to all studied projects, we also asked questions specifically designed for each project. These context-specific questions aimed at understanding the reasons behind trends observed in the figures (URLs for the questionnaires are available in Table 2). For instance, in issue `#1968`[11], which we opened for `zulip`, we asked two additional questions:

> *Question 4:* Why did the number of active contributors and contributions decrease shortly after the project became open source?
> *Question 5:* Why is there a new pick of contributors some months after the migration to GitHub? Is it related to any release?

During a period of over 30 days, we received answers for 7 out of 8 projects for which issues were opened. However, only 5 answered issues were selected for analysis; these issues received a total of 14 comments from 11 project members. The remaining two issues were discarded, because the answers did not help us at better understanding, confirming, or refuting the challenges related to the transition to open source. For instance, one respondent said that he was "*dealing with aftermath of Hurricane Matthew, and only working part-time as part of my paternity leave.*"

We found that not only were our respondents active project members, but those for projects `plotly.js`, `zulip`, and `hubot` were the most active. For

[11] `https://github.com/zulip/zulip/issues/1968`

the remaining projects, the top respondents varied from the 15^{th} most active (`hhvm`) to the 27^{th} most active (`roslyn`). The average number of contributions per respondent was 1,012 (median: 181, min: 8, max: 5,158, standard deviation: 1,818.24). By checking their affiliation information, we found that the top respondents from `hhvm`, `atom`, `hubot`, `plotly`, and `zulip` were staff members. For the `roslyn` project, only one respondent was affiliated with Microsoft (the other two did not show any affiliation). These facts might increase the trustworthiness of the responses.

To compile the survey results, we qualitatively analyzed the answers following coding procedures [48]. The qualitative analysis was conducted independently by the first two authors, followed by a consensus meeting. However, due to the small number of comments and codes that emerged, we could not bring interesting insights based on the analysis; they were (1) expressed in a single the project or (2) very specific. Therefore, instead of introducing the categories and discussing them, we opted to make use of some quotes throughout the paper as a way to enrich some of the findings from our quantitative analysis.

4 Results

In this section, we discuss the results of our main research question. We organize our findings in terms of Contributors and Contributions (Section 4.1), Pull Requests (Section 4.2), Issues (Section 4.3), and Stars & Forks (Section 4.4).

4.1 Contributors and Contributions

In order to provide a comprehensive overview of our dataset, Figure 2 presents a temporal perspective of the different contribution characteristics: the red solid line represents the number of active contributors in the particular time window; the green dotted line represents the number of new contributors that onboarded the open source project; and the blue dotted line represents the number of contributions performed. Contributions can be performed in terms of both commits and pull requests. The vertical black dotted line indicates when the project became open source. Projects `storm` and `rosyln` have two vertical lines, because they were initially open-sourced in one repository, but were later on moved to another. The time frame for each plot is one month.

Newcomers Wave. We observed that some of the analyzed proprietary projects had the potential to attract a non-trivial number of new contributors right after becoming open source, an observation that we call the "newcomers wave." That is, a high number of new contributors effectively placed a contribution soon after the project open-sourced, but disappeared a few commits later. This wave can be seen in several projects, such as `atom`, `swift`, `hubot`, and `roslyn`. On the other hand, projects `storm` and `hhvm` took several months to recruit a steady flow of new source code contributors. On average,

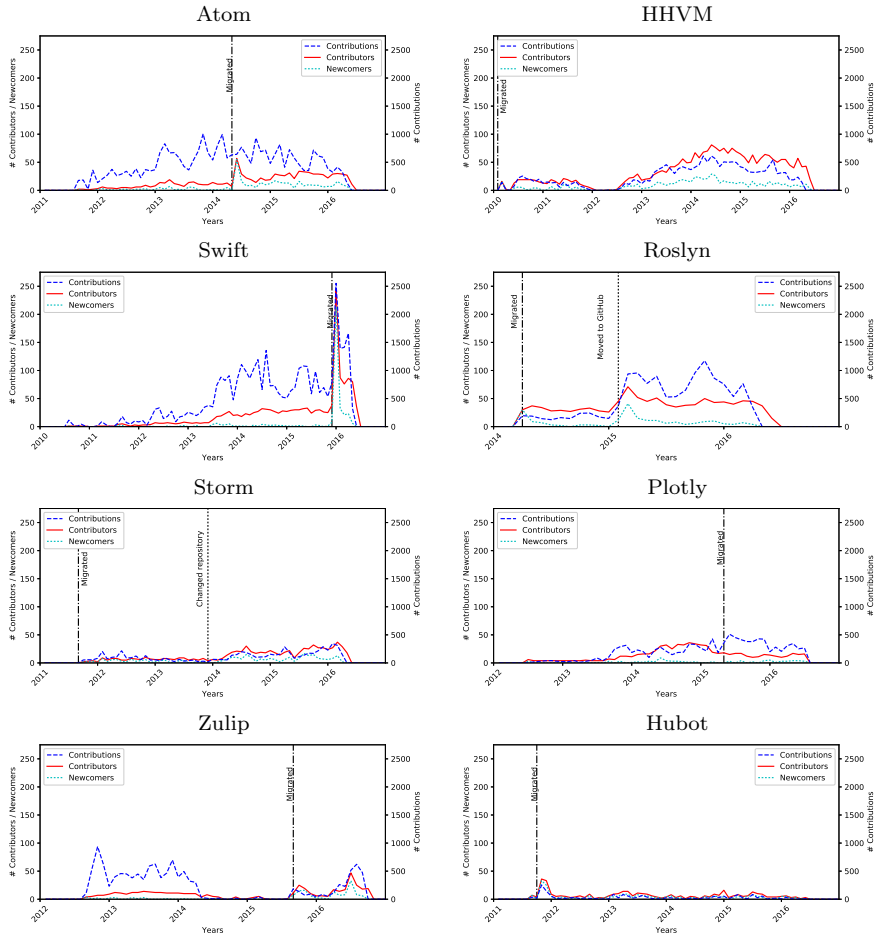

Fig. 2 A temporal perspective of the number of contributors that onboarded in the project, active contributors, and contributions. The vertical dotted line in each chart indicates when the studied project open-sourced its source code at GitHub.

our studied projects faced a rate of 77.56 new contributors per year. In particular, project `hhvm` leads the group with a rate of 155 newcomers per year. Similarly, the `swift` programming language attracted more than 140 newcomers in only its first month after becoming open source. On the other hand, project `plotly` presents a rate of 11.75 new source code contributors per year. Figure 3 shows the average number of newcomers per month, considering four different time windows: the beginning of the project up to 6 months prior to open sourcing ("Initial" bar), the 6 months before open sourcing ("Before" bar), the 6 months after open sourcing ("After" bar), and the period after the 6 initial months after open sourcing ("Final" bar). As we can see, for all projects except `hhvm` and `zulip`, the majority of newcomers onboard 6 months

after open sourcing. The `swift` project has no "final" bar because we collected data up to the fourth month after open sourcing.

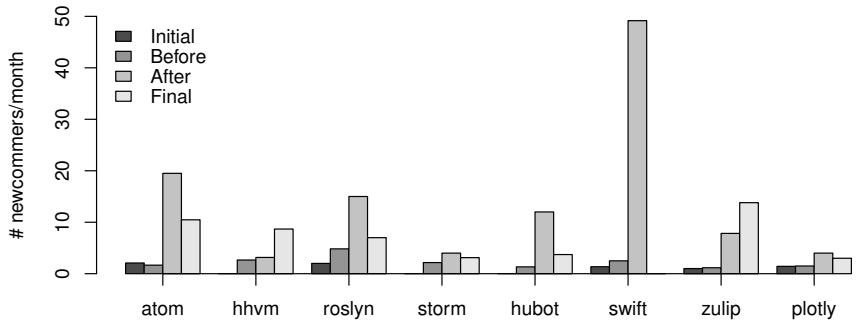

Fig. 3 The average number of newcomers per month, considering four different time windows: the beginning of the project up to 6 months prior to open sourcing ("initial"), the 6 months before open sourcing ("before"), the 6 months after open sourcing ("after"), and the period after the 6 initial months after open sourcing ("final").

Discussion. We believe there are two explanations for this behavior. First, since the number of newcomers shortly decreased a few weeks after the migration process, most of these contributors can be considered casual contributors [37]: that is, contributors that do not want to become active members of the project, but nevertheless want to contribute. For instance, taking into consideration project `roslyn`, we found 62 (44%) contributors that made only one contribution. Projects `atom`, `swift`, and `hubot` evidenced a total of, respectively, 173 (63%), 193 (76%), and 89 (72%) casual contributors. By comparison, in a previous study, we identified that casual contributors account for up to 61% of the contributors of open source projects written in JavaScript [37]. One respondent highlighted the importance of such contributions: "*My contributions may not be big but I still feel that they are appreciated. Especially if they are for parts of the repo that the team members may have forgotten about, needed dusting off and shown some love.*"

These contributors seem to be a particular kind of casual contributor: they want to contribute to well-known projects, probably as a means to gain visibility among their peers. In fact, due to the public visibility of actions on GitHub, developers who made casual contributions to different (non-trivial) open source projects might be seen as experienced developers. Recent literature suggests that gaining reputation and prestige are among the main reasons for contributing to open source projects [8,16,33]. However, retention is still an issue: for project `hubot`, only 18.5% of newcomers contributed more than once (36.3% for `atom`). Second, the contrasting numbers of contributors among the studied projects might be explained in terms of *popularity*. The `swift` programming language not only counts with Apple to promote its us-

age, but is also specifically designed for mobile apps, which are facing a growth themselves [41]. These key reasons may make `swift` particularly attractive to newcomers. In fact, 58 (34%) `swift` contributors contributed more than once.

Welcoming newcomers. According to the responses to our survey, one of the motivations for open-sourcing a software project is to allow external developers to (re)use the software. For instance, one respondent mentioned that "*The software was not being actively developed by its owners, but it had many users who wanted to continue using and improving the software.*" In order to ease the contribution process, GitHub has an official set of guidelines for those interested in launching an open source project[12]. According to the guidelines, every open source project should include the following documentation: (1) an open source license; (2) a README file; (3) a contributing guidelines file; and (4) a code of conduct file. As stated on the website, "*as a maintainer, these components will help you communicate expectations, manage contributions, and protect everyone's legal rights (including your own). They significantly increase your chances of having a positive experience.*" Analyzing the studied projects, we found that although all of them include a README file, projects `zulip` and `storm` lacked a contributing file, and projects `atom`, `storm`, `plotly`, and `hhvm` lacked a code of conduct file.

Discussion. These contributing and code of conduct files are important, because they tell potential new contributors: (1) how to participate in the project; and (2) how to behave in open source communities. Although lacking such files might make it harder to contribute or even participate in open source projects [50], some studied projects did not provide them. Still, while these files are important, other approaches can achieve the same or even better results. To welcome newcomers, one respondent said that she "*try[s] to be super responsive to new contributors on GitHub. [...] We also have the zulip.tabbott.net community server, which makes it easy for new contributors to get realtime help.*". Any type of onboarding support requires time and effort, amounting to a hidden cost that may be non-trivial for welcoming new contributors; as the same respondent complemented "*we've invested a huge amount of effort into in making Zulip's development environment really easy to setup and use compared to other large webapps."* Such efforts might pay off in the long-term, as the respondent reflected: *"I feel that open-sourcing Zulip has been a huge success, since the open source project is now moving faster than Zulip was moving when it had 11 fulltime engineers as a startup prior to the Dropbox acquisition."*

The rise of contributions. We found that for three out of the eight studied projects, the number of contributions significantly increased. In particular, projects `swift`, `plotly`, and `roslyn` faced the highest rise in contributions. In particular, the `roslyn` project presents a steep uptick in activity, as the project moved from proprietary to open source. Moreover, we believe it is too early to consider this an effective example of the rise of contributions (it became open source at the end of 2015). On the other hand, project `plotly`

[12] `https://opensource.guide/starting-a-project/#launching-your-own-opensource-project`

seems to maintain the same contribution curve, with a significant increase in the number of newcomers. Interestingly, none of the studied projects faced a decrease in the number of contributions.

Discussion. Although some projects faced an increase in the number of contributors, one respondent drew attention to the fact that "*while the number of overall committers has grown, the size of the team of maintainers who review and approve PRs has remained relatively static. This puts a cap on the amount of code that can be accepted into the project per unit of time no matter how many committers there are.*" Therefore, the lack of manpower is a serious challenge that these companies ought to address.

4.2 Pull Requests

Regarding PR usage, Figure 4 shows the number of PRs opened, closed, and merged over time. Since most of the projects did not use GitHub during their proprietary life-cycle, we cannot compare the rise of pull requests before and after the migration process. However, project `atom` was developed as a proprietary project on a GitHub private repository. For this project, we found that the number of pull requests increased 2.4 times after the transition. Since pull requests are the main way that external contributors submit changes to the main repository, we also observe the "newcomers wave" for PRs in the `atom` figure. In addition, we can observe that the number of merged pull requests of the studied projects is, on average, 80%. This suggests that the analyzed projects are willing to receive and incorporate contributions from external members, which is not always the case in open source projects (*e.g.,* [9,14]). On the other hand, project `hhvm` merged only 4.0% of the submitted pull requests. When manually analyzing a random sample of 50 pull requests, we found that `hhvm` does not use GitHub's pull request merge system. In fact, the pull requests are merged locally, through git command-line facilities (*e.g.,* PR #322[13]).

One might argue that pull requests are not restricted to external members: internal members can also provide pull requests to their own software projects. To better understand this behavior, we conducted another round of manual analysis aimed at investigating: (1) whether these contributions were indeed from external members; and (2) what were the reasons for pull request acceptance, in particular, proposed by external members. To provide answers to these questions, we focused on projects `atom` and `hubot`. We chose these projects because they were created and maintained by GitHub and, therefore, employ GitHub features that can differentiate internal and external members.

To answer ***whether these contributions were indeed originated from external members***, we verified the pull requests to check whether the submitter had the flag `site_admin` enabled. If enabled, this flag promotes an ordinary user to a site administrator. According to GitHub official documentation, a

[13] `https://github.com/facebook/hhvm/pull/322`

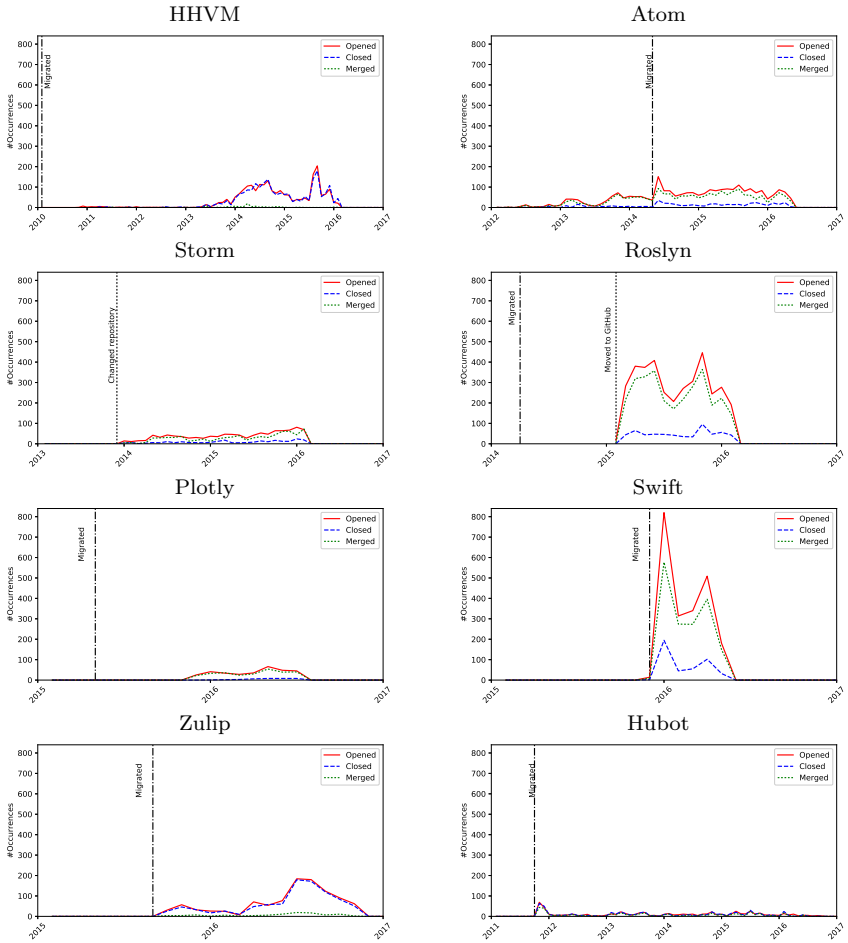

Fig. 4 Number of pull requests opened (red lines), closed (blue lines) and merged (green lines).

site administrator can *"manage high-level application and VM settings, all user and organization account settings, and repository data."*[14] We consider such site administrators as internal members, and the other pull request authors as external members. From the total of 3,297 pull requests submitted to `atom`, 1,548 of them (47%) were submitted by external members. Regarding pull request acceptance, we found that 947 (37.5%) of the pull requests submitted by external members were, in fact, accepted. More interestingly, we found that 82% (557) of the pull requests submitted to `hubot` were submitted by external members; and that 75.8% of them were accepted. Table 3 summarizes the total number of accepted (merged) and unaccepted (unmerged) pull

[14] `https://enterprise.github.com/security`

requests, divided by internal and external members. Finally, we also investigated whether the core members of these two projects comprised internal or external members. For `atom`, we found only 2 out of the 20 top contributors to be external members. However, for `hubot` 13 out of the 20 top contributors were external members. These findings corroborate with the main finding of this section: when transitioning to open source, such software projects might be willing to receive and incorporate changes provided by external members.

Table 3 Summary of pull requests acceptance for projects `atom` and `hubot`. "Internal" means "Internal member" while "External" means "External member". (Data collected in June-2017)

		Submitted		**Accepted**		**Not Accepted**	
	Total	Internal	External	Internal	External	Internal	External
`atom`	3,397	1,749	1,548	1,582	947	167	601
`hubot`	679	122	557	114	356	8	201

To answer ***what are the reasons for pull request acceptance***, we studied a random sample of 334 pull requests accepted at `atom`. This sample size provides a confidence level of 95% with a ±5% confidence interval. We also validated this experiment with another manual analysis in a random sample of 150 pull requests accepted at `hubot`. For the `atom` project, before creating a pull request, internal members create an issue that describes the project needs. Therefore, most of the pull requests proposed are accepted, because internal members *expect* them. However, external members can also propose pull requests that scratch their own itches [37], which are contributions intended to solve developers own problems. Pull requests that fix documentation problems are the most common. Examples include: broken URLs,[15] not enough information,[16] and code comments.[17] Contributions from external members are shorter than internal ones; as noted elsewhere, smaller changes can reduce the chance of breaking the continuous integration build [42]. Notwithstanding, non-trivial code changes often come with a detailed description (images are common). However, all pull requests are subject to a rigorous code review process. We found a similar pattern for `hubot`. Most of the pull requests from external members were related to documentation issues,[18] although complex code changes existed.[19] Finally, these two projects seem to welcome external users: they not only answer most of the requests from external members, but also guide their contributions to an acceptable state.

[15] `https://github.com/atom/atom/pull/1929`
[16] `https://github.com/atom/atom/pull/2602`
[17] `https://github.com/atom/atom/pull/8452`
[18] `https://github.com/hubotio/hubot/pull/788`
[19] `https://github.com/hubotio/hubot/pull/489`

4.3 Issues

Figure 5 shows the number of issues created and closed during the analysis period. This figure does not show projects `swift` and `storm`, because these projects do not use GitHub's issue tracking system.

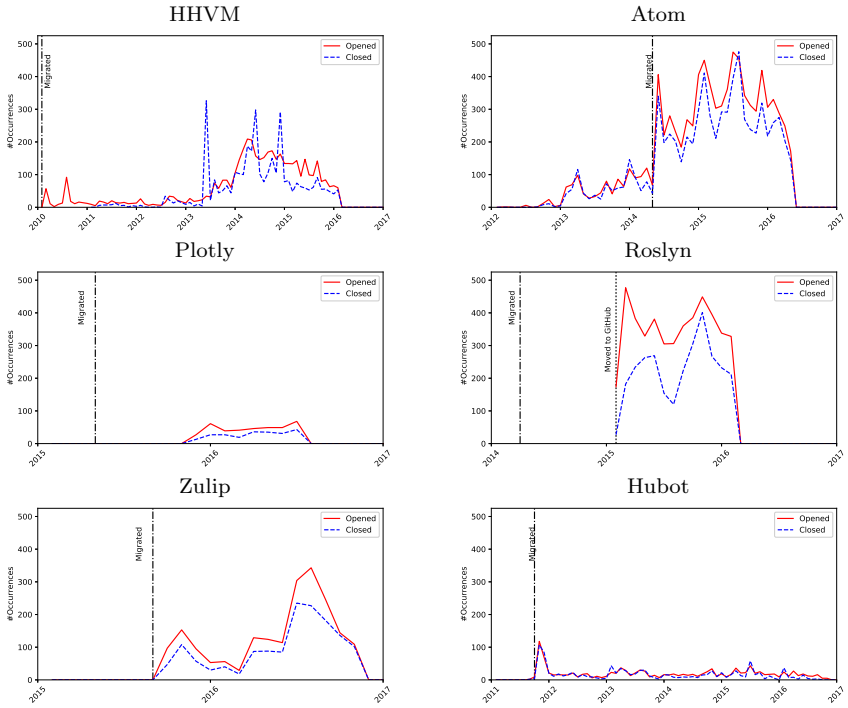

Fig. 5 Number of issues opened (red lines), and closed (blue dotted lines).

As we can see in this figure, for the studied projects there are two different issue tracker usage patterns: in the first, which includes projects `hhvm` and `atom`, the number of issues starts small and suddenly increases; and, in the second, which includes projects `plotly` and `roslyn`, they start by creating a high number of issues, and then plateau shortly after. In the first case, since projects `hhvm` and `atom` started their development as a private repository on GitHub, they used issues from their very beginning. After their transition to open source, it is likely that the increased number of issues was related to external users reporting bugs or requesting features. Similarly, projects `plotly` and `roslyn` faced a high number of issues being created soon after they migrated to a public repository in GitHub.

To further understand this behavior, we conducted two rounds of manual analysis on the `roslyn` project. For the first round, we investigated the first

50 issues created shortly after the migration to GitHub. For the second round, we randomly selected a sample of 50 issues from a group of ∼730 issues that were opened in the first three months after the migration to GitHub. Among the first 50 issues, we found that 46 were labeled either as "Feature Request" (*e.g.,* Issue #14[20]) or "Bug" (*e.g.,* Issue #23[21]). For the random sample of issues, we found 17 feature requests and 22 bug reports. These two findings corroborate our initial belief that software companies planning to open-source their software projects might expect an increased activity on the issue tracking system.

4.4 Stars & Forks

In this section, we provide results for two popularity metrics: the number of stars and forks. GitHub offers a feature that allows one to express interest in a project by "starring" [13]. Figure 6 shows the growth of stars.

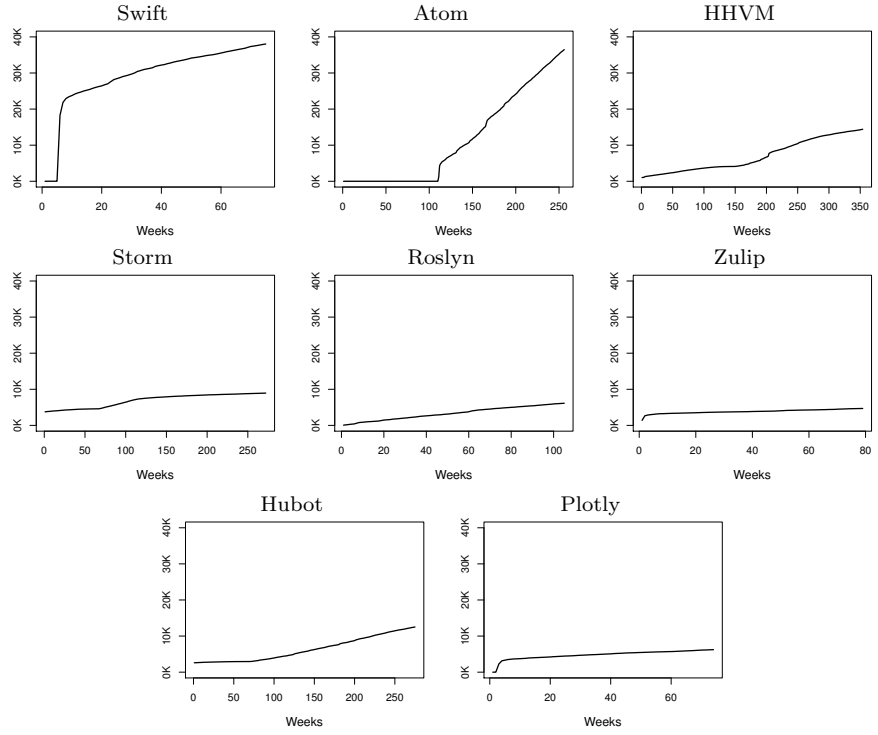

Fig. 6 The growth of stars. The line represents accumulative data. We sum up data from the two Storm repositories.

[20] `https://github.com/dotnet/roslyn/issues/14`
[21] `https://github.com/dotnet/roslyn/issues/23`

As this figure shows, the studied projects exhibit different growth patterns. We employed the same growth patterns proposed by Borges *et al.* [5], which encompasses *slow*, *moderate*, *fast*, and *viral* growth. We found that `swift` was the only project that presented a *viral* growth, receiving about 19,000 stars in a single week. Projects `atom`, `plotly`, and `zulip` presented a *fast* growth, receiving, respectively, 5,500, 3,500, and 3,000 stars in the first month. On the other hand, projects `roslyn`, `hubot`, `hhvm`, and `storm` presented a *moderate* growth. When compared to the top 2,500 most starred public projects on GitHub, we noticed that our selected projects presented a fast growth pattern — according to Borges *et al.* [5], 65.7% of the most starred projects presented a *slow* growth. This might suggest that these new open source projects succeed in attracting attention from external developers.

Similarly, the fork feature unifies the contribution process [14] and can indicate popularity. GitHub users interested in contributing to the project must first fork the project (that is, clone it), before implementing and submitting the changes back to the main repository. If a developer is only interested in using the project – not contributing to it – there is no need to fork it. Indeed, as a recent study suggests, 46% of the surveyed developers use the fork system to submit pull requests; only 9% keep copies for themselves [21]. In this sense, the number of forks can be a proxy to the number of GitHub users willing to contribute to the project. Figure 7 presents the growth of forks.

This figure shows a similar behavior when compared to Figure 6. As we can see, although on a smaller scale, the `swift` project presents a *viral* growth, whereas `atom`, `plotly`, and `zulip` present a *fast* growth, and the remaining ones present a *moderate* growth. Indeed, in most cases, there is a strong positive correlation between stars and forks. To interpret correlation, we used the thresholds from Hopkins [17]: for Spearman rho, $|rho| < 0.3$ indicates small correlation, $0.3 > |rho| < 0.5$ indicates medium correlation, and $|rho| > 0.5$ indicates strong correlation. Projects `swift`, `zulip`, and `plotly` present a Spearman correlation of, respectively, 0.996, 0.974, 0.940, with a p-value $<$ 0.001 for all of them. This result is in line with recent studies [3, 49], which reinforces the importance of a large base of contributors to the success of these projects.

Interestingly, none of the analyzed projects have more forks than stars. We believe that is because the fork is a contribution feature; that is, when one intends to make changes to a given project, they first must fork it. As a result, developers that fork a project tend to take a more active role in it. Furthermore, of the 21,352 forks in this data-set, only 1,281 (6%) had at least one additional commit. This result is in line with a recent study [34] that indicates that a majority of forks are stubs. Figure 8 shows the percentage of forks per project that performed additional contributions.

This result suggests that, although our studied projects presented a fast growth of stars, more effort is still required to motivate external developers to contribute to the original project.

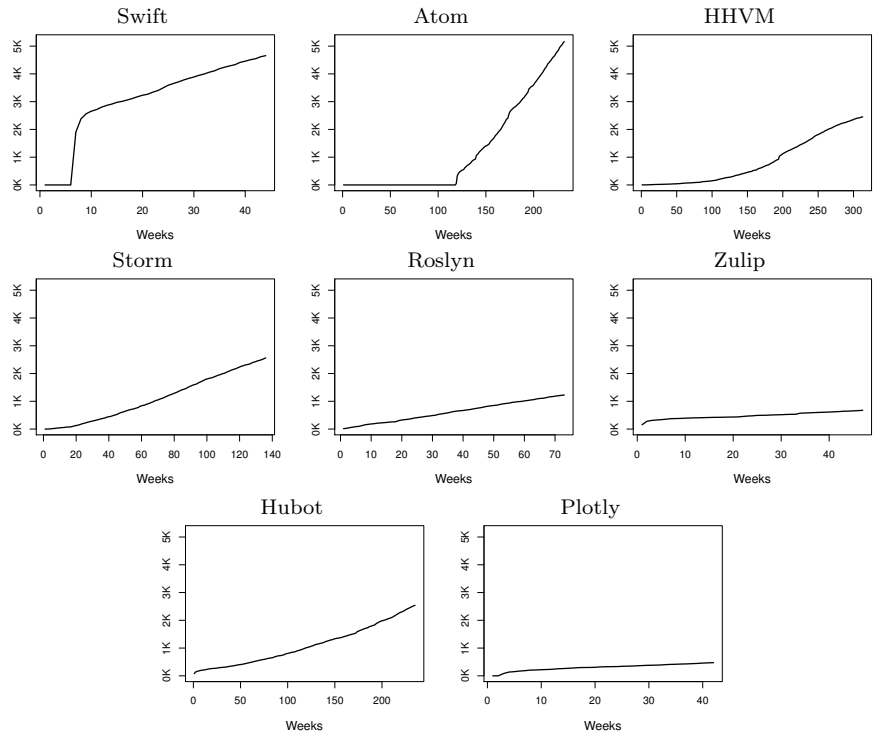

Fig. 7 The growth of forks. The line represents accumulative data. We sum up data from the two Storm repositories.

5 Additional Discussion

In this section, we summarize the challenges found and discuss some implications of this work.

5.1 Challenges

Our study reveals distinct challenges of open-sourcing a proprietary software project. Some of them are discussed below.

– **It is hard to retain new contributors.** Although numerous works study developer onboarding and retention (*e.g.,* [44,45,46]), this paper adds another perspective. We found that even though some projects faced a high number of newcomers onboarding right after their transition to open source (77.56 new contributors per year, on average), most of these newcomers abandon the project a few commits later. For instance, for `hubot` in particular, only 18% of the newcomers contributed more than once. Still, although guidelines exist for welcoming newcomers, we found that not all

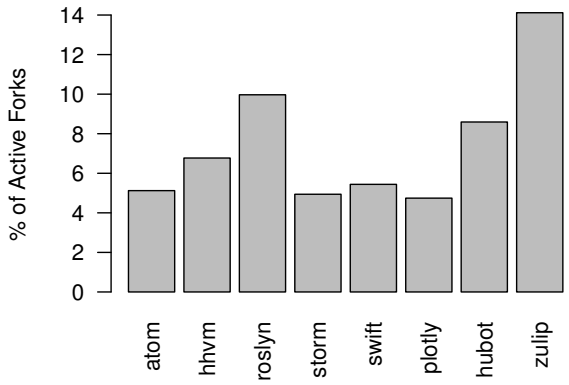

Fig. 8 The percentage of active forks per analyzed project.

studied projects follow them. Ad hoc approaches, such as using chats for synchronous help or increasing responsiveness, were also perceived as useful (Section 4.1).

– **Maintainers might expect a high volume of work after going open source.** Because of the newcomers wave phenomenon, in addition to the work needed to make the environment easy to set up and use prior to open-sourcing (Section 4.1), project maintainers should also find the time to properly review each one of the proposed contributions and give feedback to contributors (as a study participant said: *"While the number of overall committers has grown, the size of the team of maintainers who review and approve PRs has remained relatively static."*). We also observed that in the first weeks after open-sourcing external developers are fairly active using the issue tracking system to propose new features and spot bugs (Section 4.3). The challenge here relates to the amount of work required to deal with the community's needs, which might not always align with the project's needs. Therefore, projects interested in going open source might expect overhead for managing issues and code reviews.
– **Many forks do not translate into actual contributions**. Although many forks were created (`atom` itself has ∼5K forks), a minority of them are active (from the 21,352 forks in this data-set, only 1,281 (6%) had at least one additional commit). To make matters worse, when analyzing the active forks, we revealed that patch non-acceptance is a challenge for external members (only 37.5% of the pull requests submitted by external members were accepted). Furthermore, few of those external members that successfully have a patch accepted contribute with source code changes (the majority contribute with documentation files).

5.2 Implications

This research could offer significant insights and implications for different stakeholders, including two groups discussed below.

Researchers. This study introduces the possibility of a new set of research topics related to open-sourcing proprietary code. For instance, it would be interesting to understand what drives developers to contribute to these projects. Another area that can be explored is how companies take advantage of the flow of contributors; *e.g.*, do they use it as a "recruitment tool," or are they simply trying to build a sustainable open source community? Since there is discrepancy between the number of contributors and maintainers, researchers can investigate mechanisms to ease source code review (for instance, fast-rejecting patches that are beyond the scope of this project's roadmap). Still, more research is needed to reproduce our method in a larger set of projects, and in order to better understand how/whether social coding environments influenced the decision to go open source.

Practitioners. Companies can benefit from our results, which evidence diverse aspects of eight open-sourcing proprietary projects. Our results yield arguments that can support managers' decision-making regarding a possible migration of their products to open source, and help to create expectations and action plans. By opening source code, a company can improve its visibility and attract interested developers, as was evidenced by the recurrent "newcomers wave." Companies may leverage this wave, for example, to recruit new employees and establish a direct communication channel with the community. In addition, since the results showed that, for some projects, the amount of contributions and contributors increased after migration, company managers can start thinking about policies for creating a sustainable community by maintaining the flow of new contributors and retaining existing contributors.

6 Limitations and Threats to Validity

In a study such as this, there are always limitations and threats to validity. One might argue that we analyzed a small number of projects, which limits the generalization of our results. Nevertheless, our selected projects are *diverse* [32] in terms of *domain*, *age*, number of *contributions*, and *contributors* (refer to Section 3.2 for details). Our strategy was to focus on well-known proprietary projects that recently open-sourced their code at GitHub. We acknowledge that it is likely that these projects received more attention from external contributors than less popular projects. Therefore, our findings cannot be directly generalized to other projects (popular or not), either hosted on GitHub or other forges. Furthermore, we selected our projects by searching blog posts, newsletters, and README files; the first author manually conducted this process. Due to the qualitative nature of this approach (and the timeliness of a proprietary project becoming open source), one could find other projects. Still,

although we consider our projects diverse, we certainly did not discover all possible characteristics related to open-sourcing proprietary software projects. More replication is necessary to fully understand the phenomenon. To facilitate replication, we made available the scripts used to analyze data at the companion website: `https://github.com/gustavopinto/migration-to-oss`.

We also acknowledge that we could only conduct the manual analysis regarding patch acceptance (Section 4.2) in two projects (`atom` and `hubot`), since the other ones did not employ the `site_admin` flag, which we used to differentiate project participants. Still, one might argue that this approach might hidden false negatives, that is, a contributor is internal yet the person does not have `site_admin` enabled. To better understand the presence of false negatives, we manually analyzed the top 20 external contributor profiles from each of these two projects. We extended the analysis to manually investigating the personal home pages (when available) and LinkedIn profiles (when matching their GitHub profile). From the 40 developers, we could find only one that had a previous affiliation with GitHub, which could be someone who was internal contributor but was classified as an external contributor.

In addition, one threat relates to how we compare contributors across different version control systems. For instance, SVN and CVS systems track authorship attributes differently than Git. So, the number of contributors before the move might represent only those who had privileged commit access, whereas with Git all commits are attributed to the original author. We mitigate this threat by comparing the total number of contributors to the ones present on the GitHub webpage. Although this web page shows the total number of contributors, it only lists the top 100 most active ones. Therefore, we cannot compare all source code contributors of each project. Using GitHub as our ground truth, we manually compared the total number of contributors that we found with the ones that GitHub reports. We found that our study reports between 7% to 10% additional contributors — the ones that we were unable to disambiguate; however, we believe that this margin of error is not sufficient to skew the main results of our study — most of whom appear after the migration to GitHub. Yet, none of our respondents mentioned that our snapshot differs from what they expected.

We downloaded the data of the selected project in the beginning of 2016. However, since we chose active projects, it is likely that the number of contributors kept increasing, thereby augmenting the difference from the numbers presented in this study.

Moreover, in this study, we used GitHub's issues to send out our questionnaires. Such public participation can also be a threat, since anyone could answer our questions. To mitigate this threat, we verified whether the respondents were in fact active project members. We observed that respondents for projects plotly.js, zulip, and hubot were indeed the most active. For the remaining projects, the respondents varied from the 15^{th} most active (hhvm) to the 27^{th} most active (roslyn). Also, we observed that some open source projects do not use issues for discussions. For these cases, our issues were closed almost right after their creation. We therefore got in touch through the

official mailing list. Still, although our survey received comments from different project members, some of the responses were (1) short or (2) incomplete. Therefore, we were unable to answer questions such as "is the main motivation of the companies to attract contributors" or "is the main motivation perhaps to promote a profile of open source-friendliness and achieve goodwill from the community?" We plan to conduct interviews in a future work to answer such in-depth questions.

Lastly, one could argue that this study does not provide a novel contribution; *e.g.,* "of course a project will get more contributors / pull requests / bugs / contributions after going public". However, such common-sense assumptions are often in fact not backed up by scientific evidence. This paper provides such evidence and, more significantly, quantifies the phenomenon: even though some perceptions are confirmed (*e.g.*, "not all users that fork a project actually contribute to it"), others are challenged (*e.g.*, "the rise of contributions is not straightforward").

7 Conclusion

In this paper, we studied the shift from proprietary, closed source software to open source software in terms of *collaboration* and *popularity* metrics. Investigating a curated set 8 of well-known, formerly proprietary projects, we observed that some projects indeed experienced a growth of contributions and contributors after becoming open source, even though this growth does not last long. One of the reasons is what we call the "newcomers wave:" a high number of contributors who effectively place a contribution right after the project becomes open source, but abandon the project a few commits later. Some of the challenges that projects may face include: it is hard to retain new contributors; maintainers might expect a high volume of work after going open source; and many forks do not translate into actual contributions. For future work, we plan to conduct interviews with project members to gather additional insights about the motivation behind the transitioning to open source.

References

1. Abbott, T.: Open sourcing zulip a dropbox hack week project. Online (2017). URL `https://blogs.dropbox.com/tech/2015/09/open-sourcing-zulip-a-dropbox-hack-week-project/`. Accessed: Nov-20-2017
2. Anthes, G.: Open source software no longer optional. Commun. ACM **59**(8), 15–17 (2016)
3. Avelino, G., Passos, L.T., Hora, A.C., Valente, M.T.: A novel approach for estimating truck factors. In: 24th IEEE International Conference on Program Comprehension, ICPC 2016, Austin, TX, USA, May 16-17, 2016, pp. 1–10 (2016)
4. Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining email social networks. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR '06, pp. 137–143 (2006)
5. Borges, H., Hora, A., Valente, M.T.: Understanding the factors that impact the popularity of GitHub repositories. In: 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME) (2016)

6. Borges, H., Hora, A.C., Valente, M.T.: Predicting the popularity of GitHub repositories. In: Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2016, Ciudad Real, Spain, September 9, 2016, pp. 9:1–9:10 (2016)
7. CSharpFAQ: We're moving to GitHub! Online (2017). URL `https://blogs.msdn.microsoft.com/csharpfaq/2015/01/10/were-moving-to-github/`. Accessed: Nov-20-2017
8. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J.: Social coding in GitHub: Transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, CSCW '12, pp. 1277–1286 (2012)
9. Dias, L.F., Steinmacher, I., Pinto, G., da Costa, D.A., Gerosa, M.: How does the shift to GitHub impact project collaboration? In: IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, EUA (2016)
10. Donohoe, C.: Say hello to hubot. Online (2017). URL `https://github.com/blog/968-say-hello-to-hubot/`. Accessed: Nov-20-2017
11. Evans, J.: The hiphop virtual machine. Online (2017). URL `https://www.facebook.com/note.php?note_id=10150415177928920&hn=2`. Accessed: Nov-20-2017
12. Fogel, K.: Producing Open Source Software: How to Run a Successful Free Software Project, first edn. O'Reilly Media (2013)
13. GitHub: GitHub help — about stars. Online (2017). URL `https://help.github.com/articles/about-stars/`. Accessed: May-4-2017
14. Gousios, G., Pinzger, M., Deursen, A.v.: An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 345–355 (2014)
15. Hars, A., Ou, S.: Working for free? motivations for participating in open-source projects. Int. J. Electron. Commerce **6**(3), 25–39 (2002)
16. Hertel, G., Niedner, S., Herrmann, S.: Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. Research Policy **32**(7), 1159–1177 (2003)
17. Hopkins, W.G.: A New View of Statistics. Sport Science (2004)
18. Jensen, C., King, S., Kuechler, V.: Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists. In: Proceedings of the 44th Hawaii International Conference on System Sciences, HICSS '10, pp. 1–10. IEEE (2011)
19. Jergensen, C., Sarma, A., Wagstrom, P.: The onion patch: Migration in open source ecosystems. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, pp. 70–80 (2011)
20. Jergensen, N.: Developer autonomy in the freebsd open source project. Journal of Management and Governance **11**(2), 119–128 (2007)
21. Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P.S., Zhang, L.: Why and how developers fork what from whom in GitHub. Empirical Softw. Engg. **22**(1), 547–578 (2017)
22. Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., German, D.M.: Open source-style collaborative development practices in commercial projects using GitHub. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15, pp. 574–585 (2015)
23. Kastrenakes, J.: Apple's new programming language swift is now open source. Online (2017). URL `https://www.theverge.com/2015/12/3/9842854/apple-swift-open-source-released`. Accessed: Nov-20-2017
24. Ke, W., Zhang, P.: The effects of extrinsic motivations and satisfaction in open source software development. Journal of the Association for Information Systems **11**(12), 784–808 (2010)
25. Kitchenham, B., Pfleeger, S.: Personal opinion surveys. In: F. Shull, J. Singer, D. Sjberg (eds.) Guide to Advanced Empirical Software Engineering, pp. 63–92. Springer London (2008)
26. Kraut, R.E., Burke, M., Riedl, J., Resnick, P.: Building Successful Online Communities: Evidence-Based Social Design, chap. The Challenges of Dealing with Newcomers, pp. 179–230. MIT Press (2012). URL `http://www.worldcat.org/isbn/0262016575`

27. Landwerth, I.: A journey through open source: The trials & triumphs in roslyn's first year of open source. Online (2017). URL https://blogs.msdn.microsoft.com/dotnet/2015/04/06/a-journey-through-open-source-the-trials-triumphs-in-roslyns-first-year-of-open-source/. Accessed: Jun-4-2017
28. Marlow, J., Dabbish, L., Herbsleb, J.: Impression formation in online peer production: Activity traces and personal profiles in GitHub. In: Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW '13 (2013)
29. Marz, N.: History of apache storm and lessons learned. Online (2017). URL http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html. Accessed: Nov-20-2017
30. McDonald, N., Goggins, S.: Performance and participation in open source software on GitHub. In: CHI '13 Extended Abstracts on Human Factors in Computing Systems, CHI EA '13, pp. 139–144 (2013)
31. Meneely, A., Williams, L.: Secure open source collaboration: An empirical study of linus' law. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, pp. 453–462 (2009)
32. Nagappan, M., Zimmermann, T., Bird, C.: Diversity in software engineering research. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pp. 466–476 (2013)
33. Oreg, S., Nov, O.: Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. Computers in Human Behavior **24**(5), 2055–2073 (2008)
34. Padhye, R., Mani, S., Sinha, V.S.: A study of external community contribution to open-source projects on GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pp. 332–335 (2014)
35. Pham, R., Singer, L., Liskin, O., Figueira Filho, F., Schneider, K.: Creating a shared understanding of testing culture on a social coding site. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pp. 112–121 (2013)
36. Pham, R., Singer, L., Schneider, K.: Building test suites in social coding sites by leveraging drive-by commits. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pp. 1209–1212 (2013)
37. Pinto, G., Steinmacher, I., Gerosa, M.A.: More common than you think: An in-depth study of casual contributors. In: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1, pp. 112–123 (2016)
38. plotly.js: Plotly.js open-source announcement in plotly.js. Online (2017). URL https://plot.ly/javascript/open-source-announcement/. Accessed: Nov-20-2017
39. Ray, B., Posnett, D., Filkov, V., Devanbu, P.: A large scale study of programming languages and code quality in GitHub. In: Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, pp. 155–165 (2014)
40. Raymond, E.S.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly & Associates, Inc., Sebastopol, CA, USA (2001)
41. Rebouças, M., Pinto, G., Ebert, F., Torres, W., Serebrenik, A., Castor, F.: An empirical study on the usage of the swift programming language. In: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1, pp. 634–638 (2016)
42. Rebouças, M., Santos, R.O., Pinto, G., Castor, F.: How does contributors' involvement influence the build status of an open-source software project? In: Proceedings of the 14th International Conference on Mining Software Repositories, MSR '17, pp. 475–478 (2017)
43. Sobo, N.: Atom is now open source. Online (2017). URL http://blog.atom.io/2014/05/06/atom-is-now-open-source.html/. Accessed: Nov-20-2017
44. Steinmacher, I., Conte, T., Gerosa, M.A., Redmiles, D.F.: Social barriers faced by newcomers placing their first contribution in open source software projects. In: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW '15, pp. 1–13. ACM, New York, NY, USA (2015)

45. Steinmacher, I., Silva, M.A.G., Gerosa, M.A., Redmiles, D.F.: A systematic literature review on the barriers faced by newcomers to open source software projects. Information and Software Technology **59**, 67–85 (2015). DOI http://dx.doi.org/10.1016/j.infsof.2014.11.001. URL `http://www.sciencedirect.com/science/article/pii/S0950584914002390`
46. Steinmacher, I., Wiese, I.S., Chaves, A.P., Gerosa, M.A.: Why do newcomers abandon open source software projects? In: Proceedings of the 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '13, pp. 25–32. IEEE (2013)
47. Steinmacher, I., Wiese, I.S., Conte, T., Gerosa, M.A., Redmiles, D.: The hard life of open source software project newcomers. In: Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '14, pp. 72–78. ACM (2014)
48. Strauss, A., Corbin, J.M.: Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory, 3rd edn. SAGE Publications (2007)
49. Torres, M.R.M., Toral, S.L., Perales, M., Barrero, F.: Analysis of the core team role in open source communities. In: 2011 International Conference on Complex, Intelligent, and Software Intensive Systems, pp. 109–114 (2011)
50. Tourani, P., Adams, B., Serebrenik, A.: Code of conduct in open source projects. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 24–33 (2017)
51. Tsay, J., Dabbish, L., Herbsleb, J.: Influence of social and technical factors for evaluating contribution in GitHub. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 356–366 (2014)
52. Vasilescu, B., Filkov, V., Serebrenik, A.: Perceptions of diversity on GitHub: A user survey. In: Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '15, pp. 50–56 (2015)
53. Wang, J., Sarma, A.: Which bug should i fix: helping new developers onboard a new project. In: Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '11 (2011)