ABSTRACT
The maintenance and evolution of Free/Libre Open Source Software (FLOSS) projects demand the constant attraction of core developers. In this paper, we report the results of a survey with 52 developers, who recently became core contributors of popular GitHub projects. We reveal their motivations to assume a key role in FLOSS projects (e.g., improving the projects because they are also using it), the project characteristics that most helped in their engagement process (e.g., a friendly community), and the barriers faced by the surveyed core developers (e.g., lack of time of the project leaders). We also compare our results with related studies about others kinds of open source contributors (casual, one-time, and newcomers).

KEYWORDS
Core Developers, GitHub, Open Source Software.

1 INTRODUCTION
Free/Libre and Open Source Software (FLOSS) projects have an increasing impact on our daily lives. For example, many companies depend nowadays on open source operating systems, databases, and web servers to run their basic operations. Similarly, most commercial software produced today depend on a variety of open source libraries and frameworks. However, there is a growing concern on the long term sustainability of FLOSS projects [8, 12]. For example, in a recent study, Avelino et al. [1] looked at a sample of 133 popular GitHub projects and concluded that nearly two-thirds depend on just one or two developers to survive [1]. For this reason, FLOSS projects must continuously attract new core developers to mitigate the risks of failing.

Core developers are the ones responsible for the design, implementation, and maintenance of the most important features in a project. They are also responsible to manage the project and to plan and drive its evolution [14, 19, 21]. By contrast, peripheral contributors are those who occasionally contribute to the projects, mostly by fixing bugs and implementing minor features [20, 24, 26]. Usually, core developers represent just a small fraction of the project contributors. For example, n3/n3—a very popular JavaScript visualization library, with over 73K stars on GitHub—has 121 contributors. However, the system is maintained and evolved by just one core developer [1].

Since core developers are the heart and brain of FLOSS projects, we report in this paper a survey with 52 developers who, in the last year, contributed to popular GitHub systems to the point of becoming core developers in these projects. By surveying these developers, our goal is to reveal their motivations for joining an open source project. We also asked them about the project characteristics that most helped in this process and about the main barriers they faced. The survey results can help FLOSS developers to improve some of the management practices followed in their projects, aiming to possibly expand the base of core developers.

We make the following contributions in this paper:
• We provide a list of motivations that led recent core developers to contribute to open source projects. We found that 60% of the survey participants contribute because they are also using the projects.
• We reveal a list of project characteristics and practices that helped recent core contributors to join a FLOSS project. We found they are most attracted by non-technical characteristics, especially the ones related to a friendly and available FLOSS community.
• We provide a list of the main barriers faced by recent core contributors when joining a FLOSS project. We found that non-technical barriers are the most relevant impediment they face to contribute, as the lack of time of the project leaders.

We organize the remainder of the paper as follows. Section 2 presents the study design, how we selected the studied projects and the heuristic we used to identify core developers. Section 3 discusses the main findings of the survey. Section 4 presents a segmented analysis of the survey answers. Section 5 discuss threats to validity and Section 6 presents related work. Section 7 presents the main implications of our study, including implications to practitioners and researchers. Finally, Section 8 concludes the paper.

2 STUDY DESIGN
We start by considering the top-5,000 most popular GitHub projects, ranked by number of stars. Stars are similar to likes in popular social networks and therefore are a common measure of the popularity of GitHub projects [2]. Then, we apply four
strategies to discard projects from this initial selection, as follows:

1. **Non-Software Projects**: We discarded 61 repositories that are not software projects, including books (e.g., 📚/FREE-PROGRAMMING-BOOKS and 📚/YOU-DONT-KNOW-JS) and awesome-lists (e.g., sindresorhus/awesome). To remove these projects we relied on their GitHub topics. Specifically, we discarded projects with the topics book or awesome-list.¹

2. **Projects with no lines of code in a set of programming languages**: First, we used the tool ALDanial/cloc² to compute the size of the projects, in lines of code (LOC). We configured this tool to only consider code in the top-100 most popular programming languages in the TIOBE list.³ As a result, we discarded 397 projects, which are implemented in languages like HTML, CSS, and Markdown (i.e., in non-programming languages). For these projects, the size in LOC (counting only source code implemented in major programming languages) is equal to zero. As examples, we removed the following projects: GitHub/gitignore (which is a collection of textual .gitignore templates), jlevy/the-art-of-command-line (a selection of notes and tips on using Linux command-line tools), and necolas/normalize.css (a collection of HTML element and attribute style-normalization).

3. **Inactive projects**: We are interested in projects under active development. Therefore, we discarded 830 repositories without commits in the last six months.

4. **Non-mature projects**: Our central goal is to survey recent core developers of mature FLOSS projects. Particularly, it is important the projects have a minimal age in order to provide enough development time to compute new core developers. For this reason, we discarded 1,450 repositories with less than three years.

We ended up with 2,262 open source systems, including well-known projects, as facebook/react, angular/angular, and rails/rails. Figure 1 shows violin plots with the distribution of age (in months), number of contributors, number of commits, and number of stars of the selected projects, without considering outliers. The median measures are 59 months, 50 contributors, 826 commits, and 3.1K stars, respectively. 1,256 projects (55%) are owned by organizations and 1,006 repositories (45%) by individual users. These projects are mainly implemented in JavaScript (696 projects, 31%), followed by Ruby (232 projects, 10%), and Python (230 projects, 10%).

![Figure 1](image_url)  
**Figure 1**: Distribution of the (a) age, (b) contributors, (c) commits, and (d) stars of the selected projects, without outliers.

2.1 Core Developer Identification

To identify the core developers of each project, we use a Commit-Based Heuristic, which is commonly adopted in other studies [15, 19, 21, 25]. This heuristic is centered on the number of commits by the project contributors, which usually follows a heavy-tailed distribution [15, 19], i.e., a minority of developers accounts for most contributions. According to this heuristic, the core team are those who produce 80% of the overall amount of commits in a project. However, as usually defined, this heuristic accepts developers with few contributions, regarding the total number of commits. For this reason, we customized the heuristic after some initial experiments to require core developers to have at least 5% of the total number of commits; candidates who have fewer commits are excluded. For example, to achieve 80% of the commits in moment/moment, the core team initially identified by the heuristic consists of 41 contributors. However, 38 contributors have less than 5% of the overall amount of commits. Thus, only three developers are classified as core by our customized heuristic. These developers represent 35%, 25% and 7% of the project’s commits, respectively. In favor of using this second threshold, the literature reports that even in complex projects, the core team is no bigger than 10-15 developers [19].

Despite the adoption of this second threshold, we observe in Figure 2a that the median percentage of commits by the selected core teams is 81%. Figure 2b shows the core team size per project considering the minimal threshold of 5%. We can see that more than half of the selected projects have only one or two core developers. Finally, as presented in Figure 2c, the

¹This step represents just a first attempt to remove non-software repositories; step (2) is also used to this purpose.
²https://github.com/ALDanial/cloc
³https://www.tiobe.com/tiobe-index
median percentage of commits by the selected core developers is 19%, in contrast to 0.5% using the original strategy.

Finally, we follow three steps to select developers who became core contributors in the last year of each project (see an illustration in Figure 3): (a) we apply the proposed heuristic on all commits of the project (set A); (b) we remove the last year of commits and recalculate the core team (set B); and (c) the selected set of core developers is formed by developers in the set A, but who are not in set B. In other words, this group includes developers who entered in the core team in the last year. We ended up with a list of 380 core developers, distributed over 331 projects.

Figure 2: (a) Total percentage of commits by the selected core teams, (b) number of core developers per project, and (c) percentage of commits by the selected core developers. Outliers are omitted in these plots.

Figure 3: Set A = core developers computed considering the complete commit history; Set B = core developers computed considering the commits until the year before the study; New Core Developers = Set A - Set B

2.2 Survey Design

To some extent, our survey can be seen as a firehouse study, i.e., one that is conducted right after the event of interest has happened [4, 23]. Essentially, we surveyed recent core developers, to reveal their motivations to engage in FLOSS projects and the main barriers they faced during this process. After removing the core developers who do not have a public email address on GitHub, we obtained a list of 151 potential survey participants. We sent an email to these participants with two parts. First, we include the developer’s name and data on his/her percentage of commits in the project. Then, the second part includes three open-ended questions about his/her contributions to the project: (1) What motivated you to contribute to this project? (2) What project characteristics and practices helped you to contribute? (3) What were the main barriers you faced to contribute?

We received 52 answers (covering distinct projects), which corresponds to a response rate of 34% (and a confidence interval of 11.04 for a confidence level of 95%). Finally, we use Thematic Analysis [7] to interpret the survey answers. This technique is used for identifying and recording themes (i.e., patterns) in textual documents. Thematic Analysis consists of: (1) identifying themes from the answers, (2) reviewing the themes to find patterns for merging, and (3) defining and naming the final themes. The initial theme identification and merge steps were performed independently by the first two authors of this paper. Then, we had several meetings to resolve conflicts and define the final themes. In the first question, both authors suggested semantically equivalent themes for 32 answers (62%). These themes were then rephrased and standardized to compose the final theme set. As the remaining 20 answers had divergent themes, they were discussed by both authors to reach a consensus. For the last two questions, an initial agreement was reached in 36 (69%) and 38 (73%) answers, respectively.

3 SURVEY RESULTS

The presentation and discussion of the survey results are organized around the survey questions. To preserve the respondents’ anonymity, we use labels D1 to D52 to identify them. Furthermore, when quoting their answers we replace mentions to GitHub repositories, owners, and organizations by [Project-Name], [Project-Owner], and [Organization-Name], respectively. This is important because some answers include sensitive comments about developers or organizations. It is also important to note that a question could have received two or more themes during the thematic analysis process.

3.1 Motivations

In the next paragraphs, we present the reasons that emerged for the first survey question (What motivated you to contribute?) We discuss each reason and also give examples of answers.
To improve the project because I am using it: According to 31 new core developers, they increased their contributions primarily to fulfill their own needs. As examples, we have the following answers:

I started using it, I ran into minor issues or opportunities to improve, or things that were blocking me from making progress. Since it was an open source project, I was able to contribute improvements and make the project better for my needs, and everyone else’s. (D50)

First, I was a very active user of this project at the time. However, I felt that this software could be better. I believed I had enough experience to contribute, so I stepped in. (D15)

I was using [Project-Name] for my startup in our internal dashboards and I needed a couple of features. (D43)

To have a volunteer work: 10 developers answered they contributed to take part in an open source community. As examples, we have the following answers:

I’m also in love with the idea of people sharing tools for free in order to help build a better world and promote scientific development and improving people’s lives. (D48)

I think that the fact that I was helping a lot of people, immediate feedback, motivated me to contribute more at the difficult time. (D15)

I have interest or expertise on the project domain: According to seven respondents, they were motivated by their interest or expertise on the project domain or programming language. As examples, we have these answers:

I’ve always had an interest in optimizing things, which I definitely did a lot of in this case. (D06)

I’m well acquainted with the Ruby open source world... (D10)

I am a paid developer: Five new core developers mentioned they were paid to contribute, as in the following answer:

To be honest I’m paid for contributing to [Project-Name]... (D23)

To contribute to a widely used or relevant project: According to four developers, they were motivated by the fact the project is widely used or supported by well-known organizations. As examples, we have the following answer:

Working on a project as large as [Project-Name], and knowing that any work I contribute may be used by thousands of developers, was a pretty good motivator. (D06)

The remaining motivations are as follows: because I know the maintainer (3 answers), to improve my programming skills (2 answers), to improve my CV (2 answers), because the project has a nice design and implementation (1 answer), and to train developers to contribute to FLOSS (1 answer).

Table 1 summarizes the motivations reported by the participants for the first question. Among the 10 motivations mentioned by the developers, only two can be viewed as technical ones (e.g., because I have interest or expertise on the project domain and because the project has a nice design and implementation). The other motivations are non-technical and related to the interests of the developers or the project environment.

3.2 Project Characteristics and Practices

In the next paragraphs, we present the themes that emerged for the second survey question (What project characteristics and practices helped you to contribute?). We describe each reason and also provide examples of answers.

Friendly community: According to 13 developers, they decided to increase their contributions due to the friendly community of project maintainers, who helped with issues and provided detailed feedback when revising pull requests. As examples, we have the following answers:

The main thing that helped me contribute was the friendliness of maintainers and the instructions they’ve left in the issues they answered. (D48)

The [Project-Name] community gave very detailed feedback during pull requests (sometimes quite strict feedback!) which I found really helpful, and learned a lot about Git in the process. (D27)

Availability of the project leaders: According to 11 developers, the availability of the project leaders helped them to contribute, as in the following example:

In order for people to become contributors, in any kind of open source project, the most important thing is communication and availability of the owner/maintainer. (D07)

Unit tests: According to 9 respondents, the presence of unit tests helped them to increase the number of contributions. As examples, we have the following answer:

Unit tests also helped a lot, allowed me to make changes freely with the comfort that I most likely haven’t broken anything. (D25)

Documentation: This characteristic, as indicated by eight developers, refers to a clear and complete documentation. As examples, we have:

Extended documentation that helped to keep an idea of what it was all about: which things belongs to the project and which do not. (D26)

Documentation for the whole code, especially documentation for setting up development environments of the project, I would really have struggled without that. (D25)

Table 2 summarizes the answers for the second question. In addition to the previously mentioned characteristics, we received answers citing well-structured design (4 answers), code review (3 answers), continuous integration (3 answers), programming language (3 answers), open source license (3 answers), small
The code was plagued with race conditions, code smells, bad practices. The project as a whole is complex and requires specialized knowledge. The original developer basically stopped working on it years ago.

As examples, we have the following answers:

- Project complexity and size were the main barriers they faced
- Sometimes there were very slow replies to Issues/PRs as there were very few project leaders who could merge them.
- The original developer basically stopped working on it years ago. Many of us were still using the plugin, but bug reports and pull requests built up for years without attention.
- The project as a whole is complex and requires specialized knowledge or skill sets that I don’t always have.

3.3 Barriers

In this section, we present and discuss the themes that emerged for the third survey question (What were the main barriers you faced to contribute?).

Lack of time of the project leaders: According to eight developers, the main barrier was the absence of the project leaders. As examples, we have the following answers:

- Sometimes there were very slow replies to Issues/PRs as there were very few project leaders who could merge them. (D20)
- The original developer basically stopped working on it years ago. Many of us were still using the plugin, but bug reports and pull requests built up for years without attention. (D38)

Large and complex project: Seven developers answered that project complexity and size were the main barriers they faced to increase their contributions, as in the example:

- The project as a whole is complex and requires specialized knowledge or skill sets that I don’t always have. (D45)

Non-clear, complex or buggy codebase: According to five respondents, the main barrier concerns a non-clear, complex or buggy codebase. As example, we have the following answer:

- The code was plagued with race conditions, code smells, bad practices and ugly workarounds. This made it very hard for me to quickly make changes. (D41)

Table 2: What project characteristics/practices helped you?

<table>
<thead>
<tr>
<th>Type</th>
<th>Characteristics/Practices</th>
<th>Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>Unit tests</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Well-structured design</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Code review</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Continuous integration</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Programming language</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Open source license</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Small project</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Coding guidelines</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Clear code</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Contribution guidelines</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Other technical characteristics</td>
<td>8</td>
</tr>
<tr>
<td>Non-Technical</td>
<td>Friendly community</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Availability of the project leaders</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Financial support by a company</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Open and meritocratic culture</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Small number of core developers</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: What were the barriers you faced to contribute?

<table>
<thead>
<tr>
<th>Type</th>
<th>Barriers</th>
<th>Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>Large and complex project</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Non-clear, complex or buggy codebase</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Inappropriate design or architecture</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Lack or incomplete documentation</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Programming language</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Lack of tests</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Other technical barriers</td>
<td>8</td>
</tr>
<tr>
<td>Non-Technical</td>
<td>Lack of time of the project leaders</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Lack of time of the own contributor</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Conflicts among developers</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Inexperience of the own contributor</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Hostile attitude</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Unpaid work</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Other non-technical barriers</td>
<td>7</td>
</tr>
</tbody>
</table>

Inappropriate design or architecture: This barrier, mentioned by four respondents, refers to inappropriate design or architecture. As example, we have the following answer:

Less than optimal project structure or release structures. (D07)

Table 3 summarizes the responses for the third question. In addition to the previously mentioned barriers, we received answers citing inexperience of the own contributor (3 answers), lack of time of the own contributor (3 answers), lack or incomplete documentation (3 answers), programming language (3 answers), lack of tests (3 answers), and conflicts among developers (3 answers). Furthermore, English language, decisions must be approved by a committee, old coding styles, hostile attitudes, lack of build tools, project requires specialized knowledge are other mentioned barriers, all of them with a single answer. Finally, six (11%) participants answered they faced no barriers at all. As we can see, there in this case a balance between technical and non-technical barriers, which received 33 and 27 answers, respectively.

4 ANALYSIS BY PROJECT CATEGORIES

In this section, we provide results grouped by the following categories of projects: small-to-medium vs medium-to-large projects and individual vs organizational projects.

Project Categories: We classify the 52 projects according to their size, considering the distribution of LOC of the 2,262 projects. The projects in the first and second quartiles are classified as Small-to-Medium projects (LOC ≤ 4,894); the ones in the third and fourth quartiles are named Medium-to-Large projects (LOC > 4,894). We ended up with a list of 17 Small-to-Medium and 35 Medium-to-Large projects.

We also group the 52 projects considering the type of the account on GitHub: 18 projects are developed using individual accounts (e.g., JAVAN/WHenever) and 34 projects using an organizational account (e.g., Google/Java).

Results: Figure 4a shows the results for project characteristics and practices. The figure shows the percentage of technical,
Both technical and non-technical. For example, developers contributed to individual projects exclusively due to their technical (39%), non-technical (23%), and both technical and non-technical characteristics (38%). According to the results in Figure 4a, technical characteristics are the most important factor in small-to-medium projects (64%). By contrast, they are exclusively responsible to the engagement of core developers in only 27% of the medium-to-large projects. In these projects, most answers include a combination of technical and non-technical factors (43%). Finally, there is no major difference in the results for individual and organizational projects. For example, technical factors are the only factors responsible by the engagement of core developers in 39% of the individual projects and 36% of the organizational ones.

Figure 4b shows the breakdown results for the barriers faced by the surveyed developers. First, the percentage of projects presenting no barrier at all ranges from 10% (organizational projects) to 20% (individual projects). The most common barriers in small-to-medium projects are exclusively non-technical ones (57%). In other words, developers contribute to small-to-medium projects due their technical characteristics, but often face non-technical barriers. As example, we have this answer from a core developer about the technical characteristics and practices of a small-to-medium project:

*Proper coding guidelines and documentations do help a lot.* (D05)

But he also complained about non-technical barriers:

*Not all contributors have a consistent and equivalent share of time to invest in the project. This sometimes stalls the progress...* (D05)

Regarding medium-to-large projects, we found a balance among technical barriers (32%), non-technical barriers (29%) and both types of barriers (26%). As in the case of project characteristics, there is no major difference in the results for individual and organizational projects.

In summary, we found that core developers are engaged in small-to-medium projects mostly due to their technical characteristics (e.g., unit tests), but often face non-technical barriers (e.g., lack of time of the project leaders). In medium-to-large projects, the surveyed core developers increased their contributions due to a combination of both technical and non-technical characteristics; they also faced both technical and non-technical barriers. Finally, we found that there is no major difference between individual and organizational projects, regarding their characteristics and offered barriers.

5 THREATS TO VALIDITY

The threats to validity of this work are as follows:

External Validity: The dataset used in this study is restricted to popular open source projects on GitHub. We acknowledge that there are popular projects in other platforms (e.g., Bitbucket and GitLab) or projects that have their own version control infrastructure.

Internal Validity: This threat relates to the themes denoting the survey answers. We acknowledge that the selection of these themes is to some extent subjective. For example, it is possible that different researchers reach a different set of motivations, practices and barriers, than the ones proposed in Section 3. To mitigate this threat, the initial theme selection was performed independently by the first two authors of this paper. After this initial proposal, several meetings were performed to refine and improve the initial selection.

Construct Validity: A construct validity threat relates to the commit-based heuristic for core developer identification. However, we decided to use a traditional heuristic to this purpose, widely used in other studies [15, 19, 25]. Furthermore, we customized this heuristic to exclude developers with few commits (less than 5% of the total number of commits).

6 RELATED WORK

In this section, we first compare our results with related studies which focused on three profiles of open source contributors:

- **One-Time Code Contributors (OTC)** are developers who have exactly one accepted patch. Lee et al. [16] conduct a survey with OTCs to comprehend their impressions, motivations, and barriers, when contributing to FLOSS.

- **Casual Contributors** are those with few contributions (e.g., less than 2% of the total number of commits) and who do not want to become active project members. Pinto et al. [20] conduct surveys with (1) casual contributors to understand what motivates them to contribute and

---

*To clarify, OTCs have just one contribution, while casual contributions can have more than one contribution.*
Our study

Lee et al. [16]

Newcomers

Steinmacher et al. [24]

One-Time code Contributors

Casual contributors

- 

Motivations

To improve the project because I am using it

To have a volunteer work

I have interest or expertise on the project domain

To fix bugs

To give back to the community

I am a paid developer

To fix bugs

To improve documentation

To add new features

Project characteristics

Friendly community

Skilled project members

Helpful project members

Availability of the project leaders

Unit tests


Barriers

Lack of time of project leaders

Large and complex project

Unclear, complex or buggy code

Lack of time of own contributor

Complex submission process

Complex project

Technical barriers

Limited skills or knowledge

Complex project

Lack of documentation

Table 4: Comparison of our findings with related studies.

7 IMPLICATIONS

Our study has implications both to practitioners and researchers, as follows:

Implications to Practitioners: First, core contributors should strive to provide an interesting and high-quality software product, which can attract a large base of users. Then, some of these users will decide to improve the product to fulfill their own needs. Finally, they will share the improvements with the project community, which can trigger a new cycle of improvements. Second, two non-technical practices are important to engage core developers in open source projects: nurturing a friendly community and being always available. However, technical factors—specially, the availability of unit tests and documentation—are also important. Third, the main barrier faced by new core contributors is also non-technical, the lack of time of project leaders, followed by two technical ones: project complexity and low quality code.

Implications to Researchers: First, open source projects are increasingly important elements of the digital infrastructure that supports our modern societies [8]. We also know that 93% of the respondents (see http://opensourcesurvey.org/2017). However, in our survey, restricted to core developers, documentation is mentioned as barrier by only three participants.

Eghbal [8] reports on the risks and challenges to maintain open source projects. Ye and Kishida [26] describe a study to understand what motivates developers to engage in open source development. Other studies on open source have focused on how to attract and retain contributors [3, 5, 27]. Gousios et al. [9, 10, 11] provide insights on the pull-based development model as implemented in GitHub from the integrator and contributors’ perspective. Mirhosseini and Parnin [18] investigate the use of pull request notifications in GitHub repositories on GitHub. They found that developers fork repositories to submit pull requests, fix bugs, add new features, and keep copies. joblin et al. [14] categorized core and peripheral developers based on social and technical perspectives.
these projects depend on a small number of core developers [1, 19]. Thus, researchers should continue to investigate strategies to improve open source practices and communities. Particularly, our findings might contribute to current efforts to develop health and analytics models and tools to open source projects, as proposed for example by the CHAOS3 and SECOHealth [17] projects. Second, we also showed the importance of requiring a minimal percentage of commits when identifying core developers. When this threshold is not applied, the traditional heuristic can select core developers with very few commits, which are included just to reach the total amount of 80% of the commits of a system.

8 CONCLUSION

In this paper, we reported the main reasons that led recent core developers to contribute to open source projects. We also reveal the most common project characteristics and practices that motivated them to engage in FLOSS and the barriers they faced to contribute. As future work, we plan to conduct interviews with selected project contributors, to validate and extend our findings. We are also working on a tool to assess the “health” of FLOSS projects.

ACKNOWLEDGMENTS

Our research is supported by CAPES, FAPEMIG, and CNPq.

REFERENCES