Why We Engage in FLOSS: Answers from Core Developers

Jailton Coelho, Marco Tulio Valente
Federal University of Minas Gerais, Brazil
{jailtoncoelho,mtov}@dcc.ufmg.br

Luciana L. Silva
Federal Institute of Minas Gerais, Brazil
luciana.lourdes.silva@ifmg.edu.br

André Hora
Federal University of Mato Grosso do Sul, Brazil
hora@facom.ufms.br

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, D3/D3—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:

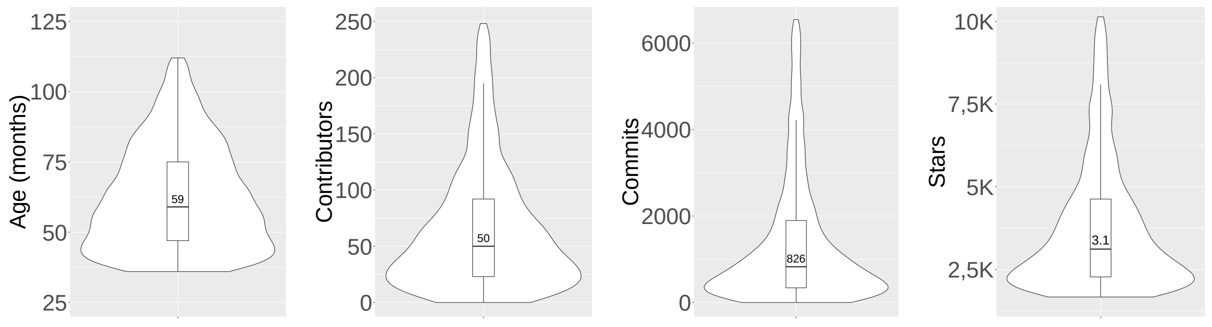
- We provide a list of motivations that led recent core developers to contribute to open source projects. We found that 60% of the survey participants contribute because they are also using the projects.
- We reveal a list of project characteristics and practices that helped recent core contributors 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

CHASE'18, May 27, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5725-8/18/05. . . $15.00
https://doi.org/10.1145/3195836.3195848



(a) Age **(b) Contributors** **(c) Commits** **(d) Stars**

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

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., vhf/free-programming-books and getify/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/cloc[2] 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%).

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

[1]This step represents just a first attempt to remove non-software repositories; step (2) is also used to this purpose.
[2]https://github.com/AlDanial/cloc
[3]https://www.tiobe.com/tiobe-index

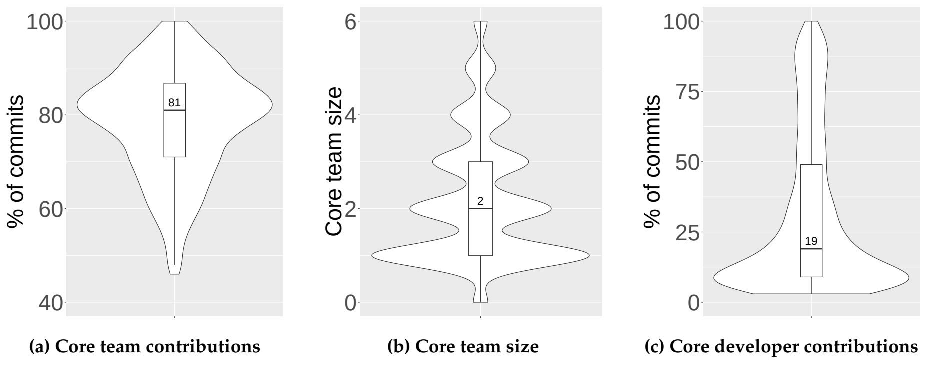


Figure 2: (a) Total percentage of commits by the selected core teams, (b) number of core developers per project, and (c) percentage of commits by the selected core developers. Outliers are omitted in these plots.

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.

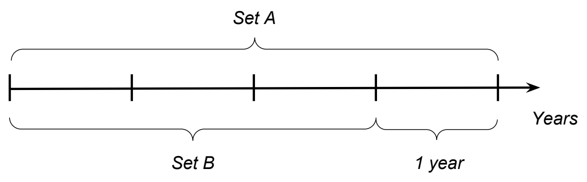


Figure 3: Set A= core developers computed considering the complete commit history; Set B= core developers computed considering the commits until the year before the study; *New Core Developers = Set A - Set B*

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.

Table 1: What motivated you to contribute?

Motivations	Dev.
To improve the project because I am using it	31
To have a volunteer work	10
I have interest or expertise on the project domain	7
I am a paid developer	5
To contribute to a widely used or relevant project	4
Other motivations (≤ 3 answers each)	9

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 example, 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*

Table 2: What project characteristics/practices helped you?

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

project (3 answers), *coding guidelines* (2 answers), *clear code* (2 answers), *contribution guidelines* (2 answers), *financial support by private company* (1 answer), *large scale tests* (1 answer), and *small number of core developers* (1 answer). In Table 2, we also provide a classification of the developers answers in two major groups: technical and non-technical characteristics.

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 3: What were the barriers you faced to contribute?

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

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 incompleted 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.

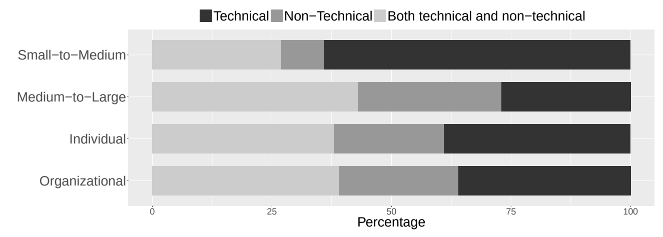
4 ANALYSIS BY PROJECT CATEGORIES

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.

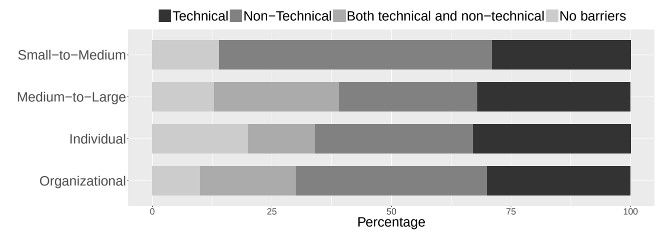
Project Categories: We classify the 52 projects according to their size, considering the distribution of LOC of the 2,262 projects. The projects in the first and second quartiles are classified as *Small-to-Medium* projects (LOC ≤ 4,894); the ones in the third and fourth quartiles are named *Medium-to-Large* 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/GUAVA).

Results: Figure 4a shows the results for project characteristics and practices. The figure shows the percentage of technical,



(a) Project characteristics and practices



(b) Barriers

Figure 4: Results grouped by project categories (Small-to-Medium, Medium-to-Large, Individual, and Organizational)

non-technical and both technical and non-technical characteristics. 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.[4] Pinto et al. [20] conduct surveys with (1) casual contributors to understand what motivates them to contribute and

[4]To clarify, OTCs have just one contribution, while casual contributions can have more than one contribution.

Table 4: Comparison of our findings with related studies.

Topic	Our study	Lee et al. [16]	Pinto et al. [20]	Steinmacher et al. [24]
Contributors	Core developers	One-Time code Contributors	Casual contributors	Newcomers
Motivations	To improve the project because I am using it	To fix bugs	To fix bugs	-
	To have a volunteer work	To give back to the community	To improve documentation	
	I have interest or expertise on the project domain	I am a paid developer	To add new features	
Project characteristics	Friendly community	Skilled project members	-	-
	Availability of the project leaders	Friendly project members		
	Unit tests	Helpful project members		
Barriers	Lack of time of project leaders	Lack of time of own contributor	Lack of time of own contributor	Technical barriers
	Large and complex project	Complex submission process	Limited skills or knowledge	Lack of contribution guidelines
	Unclear, complex or buggy code	Complex project	Complex project	Lack of documentation

(2) with project maintainers to understand how they perceive casual contributions.

- *Newcomers* are those contributors who attempted to conclude their first contribution to an open source project. Steinmacher et al. [24] elicit 58 barriers that may hinder newcomers onboarding to open source projects.

Table 4 contrasts our results with the aforementioned studies. The most common motivation for OTCs and casual contributors is *bug fixing* because it can affect their work. In contrast, the most common motivation for core developers engagement, as revealed in our survey, is *improving the project because I am using it*. Therefore, their motivation include not only bug fixing tasks, but also adding new features. Lee et al. [16] investigate impressions that increase the chances of a potential developer to contribute to a project. The most common positive impression reported by OTCs is the presence of skilled, friendly, and helpful project members. Similarly, we found that core developers are also attracted by a *friendly community* and by the *availability of the project leaders*. However, the third characteristic cited by core developers is the presence of *unit tests*, while on the case of OTCs are helpful project members. The most common barrier faced by OTCs and casual contributors is *lack of time of the own contributor*. By contrast, only three core developers reported this fact as a main impediment.

In a previous work [6], we conduct an investigation with maintainers of 104 open source projects that failed to understand the reasons of such failures. The most common reasons are projects that were usurped by competitors (27 projects), obsolete projects (20 projects), lack of time of the main contributors (18 projects), and lack of interest of the main contributors (17 projects). Robles et al. [22] describe a curated dataset with data from over 2,000 FLOSS contributors. Among the collected data, this dataset includes the contributors motivations for joining FLOSS. However, these answers can be seen as general motivations; in our survey, we decided to ask specific developers (core developers) about their motivations for recently joining well-defined open source projects.

In a recent survey promoted by GitHub with thousands of open source developers, documentation was indicated as a pervasive problem when contributing to open source, according to 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 projects. Jiang et al. [13] examine why and how developers fork 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.

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 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 CHAOSS[5] 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

[1] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A Novel Approach for Estimating Truck Factors. In *24th International Conference on Program Comprehension (ICPC)*. 1–10.

[2] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors that Impact the Popularity of GitHub Repositories. In *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 334–344.

[3] Amiangshu Bosu, Jeffrey Carver, Rosanna Guadagno, Blake Bassett, Debra McCallum, and Lorin Hochstein. 2014. Peer impressions in open source organizations: a survey. *Journal of Systems and Software* 94, 1 (2014), 4–15.

[4] Aline Brito, Laerte Xavier, Andre Hora, and Marco Tulio Valente. 2018. Why and How Java Developers Break APIs. In *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1–11.

[5] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2012. Who is going to mentor newcomers in open source projects?. In *20th International Symposium on the Foundations of Software Engineering (FSE)*. 44–54.

[6] Jailton Coelho and Marco Tulio Valente. 2017. Why Modern Open Source Projects Fail. In *11th Symposium on the Foundations of Software Engineering (FSE)*. 186–196.

[7] Daniela S. Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *5th International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 275–284.

[8] Nadia Eghbal. 2016. *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure*. Technical Report. Ford Foundation.

[9] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *36th International Conference on Software Engineering (ICSE)*. 345–355.

[10] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: The contributor's perspective. In *38th International Conference on Software Engineering (ICSE)*. 285–296.

[11] Georgios Gousios, Andy Zaidman, Margaret Storey, and Arie Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *37th International Conference on Software Engineering (ICSE)*. 358–368.

[12] Hideaki Hata, Taiki Todo, Saya Onoue, and Kenichi Matsumoto. 2015. Characteristics of Sustainable OSS Projects: A Theoretical and Empirical Study. In *8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 15–21.

[13] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. 2016. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22, 1 (2016), 547–578.

[14] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying Developers Into Core and Peripheral: An Empirical Study on Count and Network Metrics. In *39th International Conference on Software Engineering (ICSE)*. 164–174.

[15] Stefan Koch and Georg Schneider. 2002. Effort, co-operation and coordination in an open source software project: GNOME. *Information Systems Journal* 12, 1 (2002), 27–42.

[16] Amanda Lee, Jeffrey C. Carver, and Amiangshu Bosu. 2017. Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. In *39th International Conference on Software Engineering (ICSE)*. 1–11.

[17] Tom Mens, Bram Adams, and Josianne Marsan. 2017. Towards an Interdisciplinary, Socio-technical Analysis of Software Ecosystems Health. In *16th Belgian-Netherlands Software Evolution Symposium (BENEVOL)*. 7–9.

[18] Samim Mirhosseini and Chris Parnin. 2017. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In *32nd International Conference on Automated Software Engineering (ASE)*. 84–94.

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

[20] Gustavo Pinto, Igor Steinmacher, and Marco A. Gerosa. 2016. More common than you think: An in-depth study of casual contributors. In *23th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 112–123.

[21] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Israel Herraiz. 2009. Evolution of the core team of developers in libre software projects. In *6th International Working Conference on Mining Software Repositories (MSR)*. 167–170.

[22] Gregorio Robles, Laura Arjona Reina, Alexander Serebrenik, Bogdan Vasilescu, and Jesús M. González-Barahona. 2014. FLOSS 2013: A survey dataset about free software contributors: challenges for curating, sharing, and combining. In *11th Working Conference on Mining Software Repositories (MSR)*. 396–399.

[23] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why We Refactor? Confessions of GitHub Contributors. In *24th International Symposium on the Foundations of Software Engineering (FSE)*. 858–870.

[24] Igor Steinmacher, Tayana U. Conte, Christoph Treude, and Marco A. Gerosa. 2016. Overcoming open source project entry barriers with a portal for newcomers. In *38th International Conference on Software Engineering (ICSE)*. 273–284.

[25] Trung T. Trong and James M. Bieman. 2005. The FreeBSD project: A replication case study of open source development. *IEEE Transactions on Software Engineering* 31, 6 (2005), 481–494.

[26] Yunwen Ye and Kouichi Kishida. 2003. Toward an understanding of the motivation Open Source Software developers. In *25th International Conference on Software Engineering (ICSE)*. 419–429.

[27] Minghui Zhou and Audris Mockus. 2015. Who will stay in the FLOSS community? modeling participant's initial behavior. *Transactions on Software Engineering* 41, 1 (2015), 82–99.

[5] https://chaoss.community/