Understanding Collaborative Software Development: An Interview Study

Kattiana Constantino, Shurui Zhou, Mauricio Souza, Eduardo Figueiredo and Christian Kästner
Federal University of Minas Gerais (UFMG) - Brazil and Carnegie Mellon University (CMU) - U.S.

ABSTRACT

In globally distributed software development, many software developers have to collaborate and deal with issues of collaboration. Although collaboration is challenging, collaborative development produces better software than any developer could produce alone. Unlike previous work which focuses on the proposal and evaluation of models and tools to support collaborative work, this paper presents an interview study aiming to understand (i) the motivations, (ii) how collaboration happens, and (iii) the challenges and barriers of collaborative software development. After interviewing twelve experienced software developers from GitHub, we found different types of collaborative contributions, such as in the management of requests for changes. Our analysis also indicates that the main barriers for collaboration are related to non-technical, rather than technical issues.

CCS CONCEPTS

• **Software and its engineering → Open source model**; **Programming teams**.

KEYWORDS

Open Source Software projects, Fork-based Development, Collaboration in Software Development, Distributed Collaboration, Sustained Developer Community Participation

ACM Reference Format:
Kattiana Constantino, Shurui Zhou, Mauricio Souza, Eduardo Figueiredo and Christian Kästner. 2020. Understanding Collaborative Software Development: An Interview Study. In *15th IEEE/ACM International Conference on Global Software Engineering (ICGSE '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3372787.3390442

1 INTRODUCTION

Global software engineering involves globally distributed developers collaborating to develop large software systems [15]. In this environment, many software engineers have to collaborate and deal with issues from geographical, temporal, cultural, and language diversity [15, 46]. In fact, collaboration is challenging – there are many barriers to collaborative development in globally distributed software projects. It is therefore important for researchers and practitioners to understand these barriers in collaborative software development.

Several previous studies [6, 10, 13, 14, 18, 40, 51] have tried to understand developer collaboration practices as well as communication and social tools used to support developers working together in software development tasks. In fact, understanding collaboration and how it can be improved with more effective tools and processes has long been a challenge in global software engineering research and practice. For instance, Steinmacher et al. [40] reviewed the literature focusing on the *3C* collaboration model and found 42 relevant papers on this subject. However, most previous work proposes and evaluates models [1, 42], theories [2, 45], and tools [4, 18, 28] to support collaboration and to help developers in collaborative development tasks. For instance, Lanubile et al. [18] enumerated several existing tools to support collaborative work along the software product lifecycle.

Collaboration is vital to the sustainability of the project. One of the ways to promote sustainability is to enhance the motivation, engagement, and retention of developers in the project [34, 35, 39]. Therefore, before we purpose strategies to potentialize collaboration among developers, we want to understand the motivations, the process, and barriers involved in it. We still lack qualitative studies to understand the reasons and barriers involved in collaborative software development.

This paper presents results from an interview study aiming to understand the complexities and barriers for collaboration in globally distributed software projects. After a pilot study, we designed a semi-structured interview protocol and interviewed twelve experienced software developers mined from GitHub. Each interview lasted between 30 to 60 minutes and was guided by three main research questions about (i) what motivates developers to collaborate, (ii) the collaboration process adopted, and (iii) challenges and barriers involved in collaboration.

Our results indicate three main types of collaborative contributions: (i) repository management tasks, (ii) issue management tasks, and (iii) software development tasks. That is, developers collaborate not only on writing code and implementing features, but they also organize themselves and coordinate management tasks, such as coordinating and planning change requests. We also uncovered a number of communication channels used by developers to collaborate, ranging from GitHub forum to Slack and email. Furthermore, the main barriers for collaboration mentioned in our study are related to non-technical, rather than technical issues.

In summary, we contribute with (i) providing detailed empirical evidence about open collaboration in fork-based development faced by open source software developers; and (ii) bringing a discussion of the opportunities of collaboration could be more explored. We hope the open source software communities and researchers will take advantage of this paper to comprehend better the opportunities

ICGSE '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7093-6/20/05…$15.00
https://doi.org/10.1145/3372787.3390442

of collaboration among developers and design strategies to make greater use of them.

2 BACKGROUND AND RELATED WORK

In developing a theoretical basis for this paper, we have drawn from collaboration literature [9, 26, 38] to discuss how collaboration occurs in the context of fork-based development. Section 2.1 introduces some basic concepts used in this work and Section 2.2 focuses on previous works related to collaboration.

2.1 Background

Open source software (OSS) is a notable model of open collaboration, which happens when people are trying to make something together to achieve the same goal [19, 36, 50]. It is the basis for sharing knowledge, experience, and skills among multiple team members to contribute to the development of a new product. In our context, software development is a collaborative problem-solving activity where success is dependent upon knowledge acquisition, information sharing, and integration, and the minimization of communication breakdowns [2, 45]. Indeed, software developers must collaborate in all software life-cycle phases to successfully build software systems. In a typical software development process, developers perform many different tasks, such as developing software artifacts (source code, models, documentation, and test scenarios), managing and coordinating their development work or team, and communicating with each other [50].

In this research, we focus on globally distributed development of open source software on GitHub. GitHub is a Web-based code-hosting service that uses the Git distributed version control system. Furthermore, GitHub is collaborative and available for free. As of January 2020, GitHub reports having over 40 million users and more than 100 million repositories [1]. GitHub has become an essential tool in technology areas that demand collaboration, such as globally distributed software development [43].

Fork-based development and the corresponding model of submitting *pull requests* lower the "barrier for entry" for users interested in collaborating on an open source software project [14]. In this model, collaborators "fork" the repository to create an own copy of the source code and make changes without asking permission from the repository owner. Next, collaborators can submit a pull request to inform the project maintainers to integrate the changes into the main branch of the project. Pull requests are often used to initiate a code review or discussion around commits.

2.2 Related Work

Several studies have discussed diverse aspects of collaboration in open source software development projects [11, 13, 14, 17, 22, 52]. However, each one focuses on distinct aspects and perspectives.

For instance, two studies investigated the "fork & pull" development model from the integrators' [14] and contributors' [13] perspectives. Both studies investigated practitioners' working habits and challenges alike. Gousios et al. [14] investigated which motivation keep contributors working on the project and how to not discourage them. In a complementary study, Gousios et al. [13] pointed out that it is essential to reduce response time, maintain awareness, improve communication, and improve quality (e.g., source code and documentation). Contributors also need to follow guidelines and best practices for their contributions to be accepted. Although these papers investigated different perspectives of "fork & pull" development, they do not target collaboration as we do in this paper. Our work includes an interview study with twelve experienced collaborators of open source software projects hosted on GitHub. We aim to know how the community supports growing supplies of collaborators through spontaneous collaboration.

Zhou et al. [52] show that there are significant inefficiencies in the fork-based development process of many communities, such as lost contributions, rejected pull requests, redundant development, and fragmented communities. They pursue two complementary strategies: (i) identifying and suggesting the best practices for inefficient projects; and (ii) designing new interventions to improve the community awareness for correct using fork-based development and helping developers to detect redundant development. In contrast, we focus on a qualitative analysis based on interviews with experienced software developers.

Social theories propose that the sustainability of the community depends on engaged and aware community members in order to support demands on the community [5, 11]. Some studies focus on how developers use GitHub 's social features to understand developer behavior and project activity to evaluate success, performance, and potential collaboration opportunities. For instance, Marlow et al. [23] observed that developers use signals (e.g., skills, relationships) from the GitHub profile to form impressions of users and projects, focusing specifically on how social activity streams improve member receptivity to contributions through pull requests. Tsay et al. [44] found that both technical and social information influence the evaluation of contributions. McDonald and Goggins [24] noted that one of the main reasons for the increasing number of contributions and contributors to a project are features provided by GitHub. In the present study, we focus on how the project contributors perceive the collaboration process. This work may include but does not focus on the motivations and strategies to find new contributors to a project. Additionally, we focus on the social aspect of collaboration, as opposed to the technical aspects in Tsay's work [44], for instance.

3 STUDY DESIGN

This section describes the goals, research questions, and methods used in this study.

3.1 Goal

Collaboration among open source software developers could take many forms. In a sense, there is a range from the enduring partnership between members at an open source software project team, such as to join insights to solve an issue, to program in pairs, to share time, resources, and to acquire knowledge. Collaborators may have expectations concerning the kinds of contributions they want and concerning the roles and responsibilities of each party. Sometimes, the term "collaboration" may have different meanings to the collaborators and others who may be directly or indirectly involved. Therefore, *the main goal of this paper is to understand how*

[1] https://en.wikipedia.org/wiki/GitHub

collaboration happens in fork-based development. We want to identify which possible collaboration tasks are common in fork-based development projects. Furthermore, we aim to comprehend how collaboration among developers happens in fork-based development projects. We expect that it could help project maintainers to optimize collaboration opportunities and attract more community members.

3.2 Research Questions

To obtain a deep understanding on how collaboration in fork-based development happens, we interviewed twelve experienced collaborators in open source software projects in the context of the social coding site GitHub to address the research questions described below.

RQ1 - What are the motivations to work collaboratively in fork-based development projects? With RQ1, we want to investigate the individual motivation of fork-based development contributors. We also discuss the reasons for working independently mentioned by participants.

RQ2 - How does the collaboration process occur in fork-based development projects? To make the analysis more comfortable, we further refine RQ2 in the following sub-questions.

RQ2.1 - What are the collaborative contributions in fork-based development projects? With RQ2.1, we want to identify all collaborative contributions reported by participants and highlight the main collaborative contributions. Moreover, we want to know which collaborative contributions could be further explored into a software project.

RQ2.2 - What are the roles involved in fork-based development projects? With RQ2.2, we want to comprehend how the contributors organize themselves, in particular, considering their roles and interest in different types of collaborative contributions.

RQ2.3 - How this collaboration happens (communication channels) in fork-based development projects? With RQ2.3, we want to know which communication tools are often used by participants and understand how communication occurs during the collaborative contributions in fork-based development projects.

RQ3 - What are the challenges and barriers in working collaboratively in fork-based development projects? With RQ3, we are interested to know which challenges and barriers are faced by collaborators when working collaboratively in fork-based development. Besides, we want to know which directions we could pursue in order to optimize the opportunity for collaboration in fork-based development.

Overall, answering these questions helps research and industry alike to analyze, understand, and improve the opportunities of collaboration in fork-based development.

3.3 Research Method

Pilot. To better understand perceptions about collaborative contributions in fork-based development projects, we met with all researchers involved in this study to discuss an interview protocol. We first decided to run a pilot study by selecting three developers from our contact lists. With these pilot interviews, we refined the protocol, questions, focus, and time for interviews. All participants in the pilot interviews are Ph.D. candidates in Computer Science or Information Systems, while also software developers and technical contributors to GitHub projects. Among others, we learned that time for answering was too short, and we decided to give more time to explore the topic better. Therefore, we reformulated the interview script and excluded the pilot interview data from our analysis.

Qualitative Data Analysis. We qualitatively analyzed the interview transcripts using standard coding techniques for qualitative research [7, 8]. Two researchers analyzed the responses individually and marked relevant segments with codes (tagging with keywords). Later, the researchers compared their codes to reach consensus and tried to group these codes into relevant categories. With the support of the other two researchers, the codes and categories were discussed to extract key findings and theories.

Interview Protocol. We conducted individual interviews that were conducted face to face or with Skype. Each interview lasted between 30 to 60 minutes. We recorded and transcribed interviews in order to code them. The interviews were semi-structured based on five guiding questions, as follows.

(1) In the fork-based development project hosted on GitHub which you are mostly involved in, do you prefer to work alone or with other developers? Why?
(2) In the *<project name>* project, what kind of collaboration is the most common among developers?
(3) Are there other collaborations that are important, but little explored in the *<project name>* project that you participate in?
(4) Do you usually open some communication channels with any of the project developers in *<project name>* for more collaborations? How? Which tools do you use?
(5) In the *<project name>* project, in which you are involved in, do you usually visit the forks of other developers to find something that might be useful for you?

Since our interviews were semi-structured, the interviewers asked follow-up questions that were not in the interview script in order to further explore potentially interesting points that participants said. Moreover, we talked for a short time informally with the participants before the interviews to facilitate a friendly and relaxed atmosphere. During the interviews, we encouraged them to talk freely. In some cases, the participants referred to GitHub for clarifications or explanations. All these strategies supported the exploratory nature of our study and allowed us to examine unexpected insights.

Participant Selection. In order to select participants, we first mined frequent GitHub contributors with more than 500 commits

Table 1: Participants Demographics. Each participant (P#) answered the interview questions focusing on the project to which they collaborated. Participants' demographic information concern gender, last education status, years of OSS development experience, number of contributions on GitHub for the past three years, and previous or current roles in the project.

Participant	Gender	Education	OSS Experience (years)	Nb. of GH Contributions* (last 3 years)	Contribution Type**
P01	M	B.S.	4	>870	TC/U
P02	M	B.S.	2	>1,800	TC
P03	F	B.S.	4	>2,740	TC/SEO
P04	M	M.S.	5	>600	TC/ME
P05	M	B.S.	9	>4,580	TC
P06	M	B.S.	11	>3,900	TC
P07	F	M.S.	7	>2,550	FPM/TC/SEO
P08	M	Ph.D.	10	>3,500	CM, SEO/M/TC
P09	M	M.S.	11	>3,950	ME/TC/SEO
P10	F	M.S.	4	>2,500	CM/ME/TC/SEO
P11	M	B.S.	8	>4,300	FM/TC/SEO
P12	M	M.S.	5	>530	PM/TC/M/U

**We updated the values of the "Nb. of GH Contributions" column in March 2020.*
***The acronyms used in the "Contribution Type" column stand for: Community Manager (CM), Technical Contributor (TC), Former Project Manager (FPM), Social Event Organizer (SEO), Maintainer (M), Mentor (ME), and User (U).*

in the last three years and Portuguese speakers (the language of the Brazilian authors) using GitHub's REST API v3[2], in line with prior studies on GitHub [29, 30, 47, 48]. Next, we sent e-mails to invite the top eighty contributors for an interview. A total of eighteen contributors replied to the invitation. However, six of them later cancelled the interviews. Therefore, twelve contributors participated in the final interview process. Before starting the interview, participants provided their demographic information, including whether they are more than 18 years old (condition to be interviewed). In Table 1, we summarize demographic information of our interviewees. We identified each participant with an anonymized identifier (P#). Nearly all participants volunteered their time to contribute to their respective project and only two work full time as a professional in the project (and receive financial incentives). Notably, all participants have knowledge in software engineering or software development and more than three years of software development experience.

Project Selection. For each interviewee, we focused the discussion on a single open source software project, that we would pick upfront. We used the project during the interviews to contextualize the questions for participants. Moreover, the project could help them remembering or focusing answers on their collaborative experience into a project in the context of fork-based development. When the interviewee is active in multiple projects, we would pick the more popular one, in terms of stars and forks.

4 RESULTS

This section presents the results of the interviews according to each research question proposed in Section 3.2.

[2]https://developer.github.com/v3/

4.1 RQ1 - What are the motivations to work collaboratively in fork-based development projects?

The first research question aims to investigate the individual motivation of open source software contributors. Regarding the collaborative aspects, i.e., working with others in the execution of specific tasks, five participants stated that they prefer to work with others, five participants prefer to work alone, and two stated that it depends on each specific situation.

Working Collaboratively. In Table 2a, we present the reasons why collaborators prefer to work as a team. The results show that some motivations for working collaboratively are related to positive impacts on the project (e.g., benefits to knowledge sharing, strengthened synergy in teamwork, increased productivity, and increased code quality). For example, P04 explained *"I prefer to work with other developers, because of their opinion on the developed code is essential to make developers more confidence,"* and similarly, P09 emphasized *"I prefer to work together with other developers. It increases code quality. Also, you can learn more, because you interact with your colleagues all the time. Moreover, you can have new ways to solve problems."*

Working Independently. In Table 2b, we show the reasons why collaborators prefer to work independently. The motivations for working independently are for particular benefits, own satisfaction, no pressure, and own pace. Besides, the participants mentioned some drawbacks for working in groups, such as the dependence of other collaborators and the time-consuming nature of collaborative work. For example, when collaborators need to make a joint decision. These drawbacks could postpone the project results. P10 reported: *"You depend on the result of other people and of more time*

Table 2: Reasons for working collaboratively and independently.

(a) working collaboratively

Codes	Participants
Knowledge sharing/learning	4
Teamwork	3
Productivity	2
Increased code quality	2
Professional activities	1
Positive if it is asynchronous	1
Improved review	1

(b) working independently

Codes	Participants
Work on personal interests	2
No pressure\own pace	1
Working collaboratively is time consuming	1
Dependency on others	1
Different timezone	1
Work on isolated contributions	1
Not a core contributor	1
Coding is an independent task	1
Only collaborate if help is needed	1

to solve an issue or make a decision. It can be a problem." P08 was one of the participants who remarked that sometimes likes to work with other developers, like *"pair programming,"* and other moments it prefers to work in own pace: *"I prefer to work alone, in my own time, for fun, without any pressure."*

Our results show the developers' motivations for collaborating into the open source software project both in a group or independently. These motivations coincide with the motivations found by previous works [31, 32, 41, 49]. We realized that the participants reported some of their experiences and possible outcomes of the collaborations in the project. As well as their expectation for some recognition concerning their works, even if it is not monetary.

RQ1 - *Working Collaboratively.* Developers' motivations are focused on providing positive results on the project. They highlight benefits of knowledge sharing, strengthened synergy in teamwork, increased productivity, and increased code quality.

Working Independently. In this context, developers perceive personal benefits by working without pressure and at their own pace. For instance, they do not have to deal with dependencies among developers and time-consuming joint problem-solving activities.

4.2 RQ2 - How does the collaboration process occur in fork-based development projects?

The second research question aims at characterizing the collaboration process in the fork-based development community. Therefore, we aim to understand: what are the collaborative contributions perceived by the participants, who are the involved people, and how this collaboration happens (communication channels).

RQ2.1 - Collaborative Contributions. In Table 3, we indicate that feature developing, issue solving, code integrated into the upstream, and code review are the most recurring collaborative contributions mentioned by the participants. In fact, previous study [21] confirms that issue fixing ("fix bug") and feature development ("add new feature") are the top motivations for developer. The responses of participants also indicate a transparent collaboration process that revolves around solving issues. It includes opening and categorizing issues, developing code to implement changes (e.g., new features, improvements, or fixes) required in these issues, submitting pull requests, reviewing the code, and integrating the code to the upstream as presented in Table 3. The fork-based development model helps the control of the contributed code quality by selecting new contributions, and incremental code reviews [12, 44]. Thereby, collaborators perceive each task as a contribution to the projects, not only the code itself. We observe that some tasks are suitable for a collaborator to do alone. However, some collaborators prefer looking for collaboration with other developers to perform these tasks. As seen in Table 3, `Software Development Tasks` and `Issues Management Tasks` are the most prominent categories regarding what participants understand as collaborative contributions in fork-based development projects.

RQ2.1 - Software Development Tasks and Issues Management Tasks are the most notable categories concerning what participants understand as collaborative contributions. In particular, they emphasize the tasks of feature developing, issue solving, code integrated into the upstream, and code review.

RQ2.2 - Roles involved. In Table 4, we show the roles of developers involved in collaborations. The first column of Table 4 presents the two categories we identity: `Software Developer Project Roles` and `Committer Type`. The analysis of Table 4 revealed four perspectives on the roles involved in fork-based development projects: their role in the development process, their contributor type, their expected characteristics, and their responsibilities. Regarding software development roles, the roles of "developer" and "maintainer" were the most mentioned by the participants. The roles of project promoter, coordinator, core team, and reviewer were also mentioned.

Two participants also used categories to classify committer type as "peripheral contributor", "core contributor", and "newcomers" as presented in Table 4. Interestingly, these types of committers match the terminology used in recent research paper [20]. This finding indicates an alignment between software development research and practice. There are a set of responsibilities attributed to these collaborators, as stated by P05 and supported by P03 and P08:

Table 3: Categories and codes for the types of collaborative contributions in fork-based development projects. We observed four main types of collaborative contributions: (i) `Software Development Tasks`, such as development of new features, code review, and handling pull requests; (ii) `Issues Management Tasks`, for the management of issues including their opening, categorization and solving; (iii) `Repository Management Tasks`, for tasks related to handling the repository (e.g., updating the repository with new code); and (iv) `Documentation Tasks`, such as writing, improving and translating documentation.

Category	Codes	Part.(#)	Freq.
Software Development Tasks	feature developing	9	19
	code review	6	10
	writing code	4	9
	opening a pull request	4	7
	submitting pull request	2	2
Issues Management Tasks	issue solving	7	15
	issue reporting	4	8
	triaging issue	4	6
	issue opening	4	4
	issue detecting	2	3
Repository Management Tasks	code in upstream	7	9
	code in forks	2	4
	manage repository	2	2
Documentation Tasks	writing documentation	4	6
	translating documentation	3	5
	internationalization project	3	4
	improving documentation	3	3

Table 4: Categories and codes for the people roles involved in collaboration in fork-based development projects.

Category	Codes	Part.(#)	Freq.
Software Developer Project Roles	developer	8	12
	maintainer	5	5
	team leader	3	3
	project promoter	2	3
	reviewer	2	2
	coordinator	1	2
Committer Types	peripheral contributor	2	4
	core developer	2	3
	newcomer	2	2

"I am a core team member and also, one of the project maintainers. So, I can create a branch, or ask for someone to review a pull request." Besides, P02 said (supported by P01 and P04): *"Some issues are more appropriate for newcomers."*

Some codes identified in this analysis are related to desired characteristics of contributors. The main desired characteristics are "availability to collaborate" and "engagement", also cited in literature [21, 25, 33]. For instance, availability includes the contributor's ability to conciliate the volunteer aspect of collaborating in fork-based development projects with their formal employment schedule. P07 affirmed (supported by P02): *"Collaborators need to be very engaging to work on an open source project."*

Table 5: Categories and codes for the collaboration channels.

Category	Codes	Part.(#)	Freq.
Communication and Coordination	GH Issues (forum)	7	12
	Slack	3	4
	GH PR (forum)	3	3
Remote Interactions	e-mail	3	5
	Gitter tool	3	3
	Telegram	3	4
	IRC	1	2
	Meeting.gs	1	1
	Twitter	1	1
	WhatsApp	1	1

**The acronyms used in the "Codes" column stand for: GitHub (GH), Pull Request (PR), and Internet Relay Chat (IRC).*

RQ2.2 - Software Developer Project Roles and Committer Types are the categories of people involved in collaboration into the project. Committer Types are concerned with how often the collaborators commit to the project.

RQ2.3 - Collaboration Channels. In Table 5, we present the codes related to the communication channels and tools used for collaboration in fork-based development projects. GitHub issues (forum) was the most cited. Considering that all participants are GitHub users, it is not surprising that they mention this platform as their main collaboration channel. However, it is interesting to notice that the issue system is a core element in the open source software collaboration process. It complies with the participants' perception of issue solving being an essential collaborative contribution in the fork-based development. Bissyandé et al. [3] found a considerable correlation between the number of forks and the number of issues that brings a positive impact on the project. Indeed, through issue reporting, collaborators help identify and fix bugs, document software code, and enhance the software. More than one participant also mentioned E-mail, GitHub Pull Requests, Gitter, Telegram, and Slack as communication channels for collaboration. Prior works reported similar findings for open source software projects [10, 33] and Commercial projects [16] on GitHub. They pointed out that developers usually communicate through text messages.

Participants pointed out the communication tools are used in several contribution tasks in the life cycle of the open source project. For example, these tools are used for promoting the project, for recruiting new collaborators, for asking help to develop a new feature or to fix an issue, for discussing to solve an issue, for directing the project, or other demands. P8 explained: *"... I posted on Twitter*

asking whether someone could help me to develop a new feature for this repository... Several developers answered me." and also, P9 shared: *"...I sent an e-mail asking whether they could accept to be members of the maintainer group of the repository."*

RQ2.3 - Participants pointed out the communication tools used in several contribution tasks in the life cycle of the open-source project. GitHub issues (forum) was the most cited one by collaborators. Moreover, they usually tend to communicate with text messages.

4.3 RQ3 - What are the challenges and barriers in working collaboratively in fork-based development projects?

In order to answer RQ3, we identify the challenges and barriers that would impact on decisions into fork-based development as presented in Table 6.

Knowledge and time. Knowledge is one constant challenge and barrier in a project. Therefore, the core team needs to know how to manage and make the knowledge available in the project. P07 (supported by P11) was emphatic in saying that *"Project needs to have collaborators answering questions..."* for the project to survive because without the developers answering questions, the project fails in its first goal, which is to attract collaborators or maintainers [27]. P01 agrees with P07, but looks for knowledge between the project's collaborators senior (supported by P02 and P06).

"I have a greater tendency to ask for help from a collaborator who has contributed for a longer time and who has more excellent knowledge."

Therefore, when the core team is aware of these matters, it has the challenge of being available to meet the demands of peripheral and newcomer collaborators who want to participate in the project. For instance, it can happen as mentoring in a forum or the code review phase.

Another point related to this issue is that collaborators do not always have the time and technical knowledge necessary to help, as P02 warns *"... we use many packages made by other developers... I encounter several problems ... I try to solve them myself, but we don't always have the time or knowledge to do it."* That is, the participant highlights that acquiring knowledge takes a lot of time and effort. P02 and P07 also reported the lack of time as one of the collaboration barriers prevalent. For instance, P2 states: *"The maintainer has a job and does not have time for the project ..."* and P07 concludes *"(the collaborator needs) to have time to contribute, after all, everyone has limited time."*

When the collaborators have technical knowledge enough for the project, and all team is comfortable to share knowledge with each other, the next goal for the core team is to retain this knowledge into the project. One possibility is encouraging the collaborators to share their experience. For instance, whether the developer knows a specific functionality, they could develop a similar functionality or mentor someone that wants to develop it. P02 explains:

"...I worked in a functionality previously about phone patterns. After someone, from another country, asked me the same functionality, I adapted it to his country pattern..."

It could be a drawback for the project losing trained collaborators, and with relevant knowledge. Thus, some strategies to retain these collaborators need to be done, as P11 explains (supported by P07): *"... some regular contributors sometimes do not talk to the collaborator much. However, as the collaboration is excellent, we try to keep talking, to get closer, because the collaborator may know that in the future we can help with another contribution."*

Steinmacher et al. [41] found that the lack of experience of the quasi-contributors deemed the work unacceptable. Qiu et al. [33] related that lack of time is one of the reasons for developers to stop contributing to GitHub projects. Lack of time was also identified by Pinto et al. [32] as the most common barrier to participation faced by peripheral contributors and Miller et al. [25] found that "no time" was one of the reasons for disengagement in open source software projects.

Documentation. A barrier related to documentation is mainly the lack or out-of-date documentation. A newcomer enters the project to collaborate, but they do not find enough documentation to understand the project, as P11 reported: *"... (When) people wanted to know about more advanced things before collaboration. Then they opened issues that the documentation did not supply."*

The core team recognizes the importance of documentation. However, documentation tasks are not as prioritize as coding tasks. The following are the reports of P05 and P10, respectively: *"...The documentation is left towards the end (of the project)"* and *"Collaboration on documentation in both the code itself and official documentation are little explored, but it is as relevant as the primary collaboration (coding)."*

On the other hand, documentation is also the gateway to start collaboration on a project. It can be an excellent opportunity to become part of the contributor team. Newcomers who are unfamiliar with the project could start to collaborate with the project making or improving documentation, as reported by P06 (supported P03 and P09): *"You have to emphasize that documentation is not so explored. For instance, the project's internationalization is an exciting contribution. You could collaborate with translating documents..."* and *"I believe that a little more documentation would be very important. Many people could collaborate with documentation, since they do not collaborate with code."*

Several works stated the lack of documentation as a barrier for collaboration in the open source software project. The usual recommendation is to keep useful project documentation up to date, since documentation is one of the sources of knowledge about the project. In this way, it is the opportunity for encouraging collaboration from newcomers and becoming them familiar with the project. Hence, documentation should be easily accessible for developers. Some careful with documentation of the project could avoid demotivating developers, losing contributions for misunderstanding, and

Table 6: Challenges and barriers faced by participants. We present the categories as follow: `Knowledge and Time`, `Documentation`, `Collaborators`, and `Community Issues`. For each category, we outline the main codes based on the frequency of code and the number of participants citing them.

Category	Codes	Participants	Frequency
Knowledge and Time	Clarifying any doubts	6	11
	Collaboration depends on specific knowledge	5	7
	Collaboration depends on free time	4	8
	Seeking information/clarification	4	4
	Conciliating employment and volunteer job	2	3
Documentation	Lack of documentation	5	6
Collaborators	Lost contribution	3	4
	Dependency of collaborators	2	2
	Lack of mentor	2	2
	Lack of maintainer	2	2
Community Issues	Code in forks	3	3
	Community expectancy	2	3
	No compliance with contributing guidelines	2	2
	Problem with maintainability	2	2

overloading the discussion forums with questions quickly clarified in documentation.

Collaborators. Collaborators are the key to the success of an open source project. Therefore, we show the barriers that the participants based on their experiences point out as situations that discouraged these collaborators from remaining in the project. As a consequence, the project loses possible contributions. P08 exemplifies a situation *"... Something that happens a lot is someone who starts a contribution, and for some reason, he does not end it."*... [the participant showed a fork with 26 ahead commits in his GitHub project]... *"For example, this collaborator worked a lot, and we did not even know about it."* Such a problem likely happens because of carelessness with the collaborators. Another possible explanation concerns the high number of forks in the project, which makes it difficult to know who is working and in what [35]. Hence, to avoid losing collaborators or collaborations, P07 alerts:

"If volunteer collaborators are making several contributions, adding several things to the project, and the core team stops giving feedback to them. Certainly, they will abandon that work."

Other participants faced difficulties when the project decisions were too centralize only on one member of the core team. Many decisions to take, issues to solve, pull requests to review were cumulated or frozen waiting for maintainer decision. P02 reported:

"First, Maintaining a project is too a hard task, mainly when it is not the main activity of the maintainers. It may overwork them and make them abandon the project. But, whether they receive financial incentives, then they could dedicate themselves exclusively to the main project and attends the community demands."

Considering the statements of these participants, the collaborators have a great challenge to bring these developers back (if possible). For P07, this depends more on the motivation of the collaborators than on the efforts of the core team:

"If after a while, someone (from the team) gives feedback to the collaborator again, he will have to be very motivated to return working on the issue. After a while, without working on the code, the collaborator does not remember the entire context anymore. So, it's too difficult. The collaborator has to be really motivated to keep contributing with the project"

Since it may not be possible to have the "lost" collaborator back, the next challenge for the maintainers presented by P08 is to find another developer who can continue the activity that was previously stopped: *"So, I wanted to highlight it for someone who wants to continue this work."* Another challenge is knowing how to appreciate each contribution, P10 emphasizes the importance of valuing all collaborations, even those that seem not to have much value:

"(After, listed some contributions) These three contributions are less used, sometimes they are seen as not as important, but in fact, they are essential. ", "... I believe these are collaborations that are not the main ones, but they are just as relevant."

Community Issues - Fork Fragmentation. Some developers take advantage of the fork to specialize the project for their interest. Furthermore, in some cases, these new functionalities can not be updated for the main project; thus, this practice makes it difficult to manage the project's functionalities. P12 confesses this practice:

"Some forks are more advanced than others. One fork can have a subset of functionality, while others may have other different subsets.

Thus, if you want to use the project as a whole, there are several fragmented versions."

Therefore, fragmentation is one of the drawbacks of fork-based development and requires collaboration. Many development efforts over multiple project versions are wasted, and many bug-fixes are not propagated [37, 52]. Consequently, it makes a great set of specialized forks. For projects with a large number of forks, it becomes impracticable [37].

Community Issues - Failing to Comply with the Project's Contribution Guidelines. In order to attract collaborators and keep them active, project maintainers usually have their code of conduct and guidelines of best practices for contributions. This documentation drives new collaborators on the community's expectations regarding posture, commitment, and quality of work. However, we see reports on some procedures that collaborators still fail to follow or that require improvement, before accepting a relevant pull request as P10 explains:

"... So, to prioritize code organization and review more appropriately, I have asked the author to separate this in different pull requests and different commit to keep history, to make it easier to fix, to facilitate the review."

Another situation was reported by P05 that impacted the project and future maintenance *"Another collaboration ... the tests! The collaborators send the features, but the tests are missing, so that feature can cause a break (in the project)."*

Community Issues - Work Overload. Besides, some contributions demand great efforts from both the newcomer or peripheral collaborators and the core team, as P05 reported and supported by P04.

"Some projects are so challenging to test. For instance, they do not have any test suite for what you have done. So, first, you have to open a new issue to create the test suite for that. Next, you have to involve the core team and then do the regression tests, in this case. Also, some projects are so complex to test. For example, android test library, or libraries for platforms (some are in C ++, not even in JavaScript). So, you have to make an excellent suite to test everything."

Finally, P2 exposes the difficulty of a project having only one maintainer: *"I think that because it does not have a company behind it, it has only one individual, so, I think that sometimes there is a little lack of organization."*

RQ3 – We identified four categories of barriers and challenges faced by developers. That is, collaborators face issues regarding lack of knowledge and time, documentation, the dependency of collaborators, and community issues. Fork fragmentation and work overload were recurrently mentioned.

5 DISCUSSION OF THE FINDINGS

In this section, we discuss and summarize some of our main findings for researchers and practitioners.

Task Segregation. Many tasks have a level of difficulty that requires the equivalent level of expertise to be solved. Several projects have rated some issues as easy for newcomers, so that those can feel more comfortable starting to collaborate on a project. Newcomer issues are also a way of communicating that the project is friendly. In fact, GitHub has some default flags for easier issues, such as *"good first issue"*. Our work shows that many projects are aware of newcomers and advertise a friendly project. On the other hand, some comments raised concerns about how overwhelmed the core team can be, especially, since for many first option is always to seek help from the core team. Indeed, some issues require the experience and knowledge of the core team. Since, the core team needs to direct the project, it holds great responsibility. However, we wonder to what extent some issues could be solved by peripheral or even by newcomers? (It is important to note that some of them have expertise coming from other projects). To what extent collaborations that are reserved for the core team could be delegated to other collaborators able to help them? Furthermore, to what extent it motivates them to collaborate and engage in the project?

Task Sizing. Our results show that dependencies can be challenging, especially if they involve the core team. Hence, such contributions may be complicated. We wonder to what extent a larger task could be better divided in smaller tasks in order to optimize the efforts of collaborators and minimize the dependence? How many developers would be enough to perform the complete issue without impacting project schedule and resource.

Challenges in collaboration with non-domain members. Indeed, collaborations in the software development process depend on specific knowledge. Depending on the nature of the project, it can be both technical and non-technical knowledge. Capturing the knowledge of non-technical professionals, who are not software developers or engineers, is not a trivial task. Mediators who can bridge the gap between the two different universes of knowledge are urgently needed.

More tools does not mean better communication. Geographical and chronological distances between software developers can restrict both formal and informal communication. Hence a more significant number of tools does not equate to better communication. Therefore, to improve communication between developers, several types of communication tools, synchronous or asynchronous, can be impartially engaged during software development processes.

Nonetheless, malpractice and dependency of such tools can, at times, result in adverse effects. In other words, it can bring upon misunderstandings and misinterpretation during any of the different phases of project development processes:

(1) provide knowledge to different channels, failing to reach every one of them;
(2) provide incomplete information or partial experience to all involved or interested parties;

(3) provide unsolicited information to non-interested parties;
(4) produce excessive details; and
(5) hold back on hard to reach or inaccessible data.

Therefore, it is crucial to define and update the communication policies specifying guidelines for critical information as well as possible communication noise that should be kept available to all members, including all communication channels.

6 LIMITATIONS AND CREDIBILITY

In this section, we clarify potential threats to the study's credibility and discuss some bias that may have affected the study results. We also explain our actions to mitigate them.

Results. The results presented in the study are first and foremost observations and interpretations. These results reflect individual perceptions of practitioners, and our interpretations of their responses. All researchers participated in the data analysis process and discussions on the main findings, to mitigate the bias of relying on the interpretations of a single person. Nonetheless, there may be several other important issues in the data collected, not yet discovered or reported by us.

Interview scripts. We used an interview script to ensure that all participants were asked the same base questions. The interview script was developed in stages. The script was first piloted in three interviews. After these pilot interviews, we reformulated the script to ensure that the questions were sufficient to generate data to answer our research questions. Only then, we invited the other participants to the study. The use of semi-structured interviews was also relevant for allowing the on-the-fly adaptation of questions, in case the interviewer noticed any possible misunderstanding of questions. After the conclusion of the analysis, the final manuscript related to the results of the interviews were sent to the participants for validation. This validation did not result in change requests from participants.

Number of participants. A larger number of participants should be interviewed to capture the general view of a broader audience. However, this type of study is limited by the availability of practitioners willing to participate in a study without any type of reward or compensation for their time. Therefore, we sent 80 invitations by e-mail. Consequently, we cannot generalize the results of the study. Nonetheless, we found some consensus in a random sample with participants from different projects, which may depict perceptions of the community regarding how collaboration happens in fork-based development projects.

Brazilian participants. We interviewed twelve Brazilian developers in order to avoid communication issues. However, it could have the chance of limiting the generalization of our subjects. To mitigate this limitation, we were careful to invite active and experienced contributors to the open source project (top-eighty contributions on GitHub). In fact, before starting the interview, all interviewees had the opportunity to talk about the main open source projects they contributed and their current occupation. Most of them have worked as senior developers not only in Brazil but also in other countries (England, the United States, and Germany). Furthermore, they have contributed to projects with a higher number of stars, which attract collaborators from several places around the world. All these factors corroborate with the interviewees' expertise in global collaboration with open source development.

7 CONCLUSION AND FUTURE WORK

The importance of collaboration in globally distributed software development is no novelty. However, the collaboration process evolves continuously, responding to changes in development methodologies and technologies, and taking advantage of communication technologies. In the specific case of fork-based development projects, collaboration is a key factor. In this study, our main findings include: (i) collaboration transcends coding, and includes documentation and management tasks; (ii) the collaboration process has different nuances and challenges when considering members of the core team interacting with each other, and members of the team interacting with peripheral contributors; collaboration is heavily driven by issue management, and it is impacted by management skills in defining, categorizing, and sizing tasks accordingly, in such way that the community (including newcomers) can collaborate independently; (iii) knowledge management is a challenge in collaboration, and it is important to carefully define communication policies in order to mitigate and avoid problems related to knowledge retention and decentralization.

Future Work. Our results could be compared with direct observations of developer activity in forking projects using a survey. Also, the identification of the factors which may impact collaboration among developers may be considered as future work. Hence, if we identify these factors, we can calibrate them to connect the desired collaboration team naturally. Besides, we are working on in a Web-tool to connect developers based on their similarities to improve collaboration among developers. These works would be essential steps to improve collaboration in the context of fork-based development.

8 ACKNOWLEDGMENTS

Many thanks to participants of our interviews and reviewers. This research was partially supported by Brazilian funding agencies: CNPq (Grant 424340/2016-0), CAPES (88881.189537/2018-01), and FAPEMIG (Grant PPM-00651-17).

REFERENCES

[1] M. Arciniegas-Mendez, A. Zagalsky, M. Storey, and A. F. Hadwin. 2017. Using the model of regulation to understand software development collaboration practices and tool support. In *Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*. 1049–1065.
[2] C. Bird. 2011. Sociotechnical Coordination and Collaboration in Open Source Software. In *International Conference on Software Maintenance (ICSM)*. 568–573.
[3] T. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon. 2013. Got Issues? Who Cares About it? a Large Scale Investigation of Issue Trackers From Github. In *International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 188–197.
[4] P. Bjorn, J. Bardram, G. Avram, L. Bannon, A. Boden, D. Redmiles, C. De Souza, and V. Wulf. 2014. Global Software Development in a CSCW Perspective. In *Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*. 301–304.
[5] B. Butler, L. Sproull, S. Kiesler, and R. Kraut. 2002. Community Eeffort in Online Groups: Who Does the Work and Why. *Leadership at a Distance: Research in Technologically Supported Work* 1 (2002), 171–194.
[6] A. Capiluppi, P. Lago, and M. Morisio. 2003. C. In *European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 317–327.

[7] J. Corbin and A. Strauss. [n. d.]. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*.
[8] J. W Creswel. 2009. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. *Los angeles: University of Nebraska–Lincoln* (2009).
[9] K. Crowston, K. Wei, J. Howison, and A. Wiggins. 2008. Free/Libre Open-source Software Development: What We Know and What We Do Not Know. *ACM Computing Surveys (CSUR)* 44, 2 (2008), 1–35.
[10] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Conference on Computer Supported Cooperative Work (CSCW)*. 1277–1286.
[11] J. Gamalielsson and B. Lundell. 2014. Sustainability of Open Source Software Communities Beyond a Fork: How and Why Has the LibreOffice Project Evolved? *Journal of Systems and Software* 89 (2014), 128 – 145.
[12] G. Gousios, M. Pinzger, and A. Deursen. 2014. An Exploratory Study of the Pull-Based Software Development Model. In *International Conference on Software Engineering (ICSE)*. 345–355.
[13] G. Gousios, M. Storey, and A. Bacchelli. 2016. Work Practices and Challenges in Pull-based Development: The Contributor's Perspective. In *International Conference on Software Engineering (ICSE)*. 285–296.
[14] G. Gousios, A. Zaidman, M. Storey, and A. Van Deursen. 2015. Work Practices and Challenges in Pull-based Development: The Integrator's Perspective. In *International Conference on Software Engineering (ICSE)*. 358–368.
[15] J. D. Herbsleb and D. Moitra. 2001. Global Software Development. *IEEE Software* 18, 2 (2001), 16–20.
[16] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. German. 2015. Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub. In *International Conference on Software Engineering (ICSE)*. IEEE, 574–585.
[17] K. R. Lakhani and E. Von Hippel. 2004. How Open Source Software Works:"Free" User-to-User Assistance. In *Produktentwicklung mit virtuellen Communities*. Springer, 303–339.
[18] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaíno. 2010. Collaboration Tools for Global Software Engineering. *IEEE software* 27, 2 (2010), 52–55.
[19] A. Laurent. 2004. *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software*. "O'Reilly Media, Inc.".
[20] A. Lee and J. C. Carver. 2017. Are One-Time Contributors Different? A Comparison to Core and Periphery Developers in Floss Repositories. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–10.
[21] A. Lee, J. C. Carver, and A. Bosu. 2017. Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. In *International Conference on Software Engineering (ICSE)*. 187–197.
[22] J. Linåker, H. Munir, K. Wnuk, and C. Mols. 2018. Motivating the Contributions: An Open Innovation Perspective on What to Share as Open Source Software. *Journal of Systems and Software* 135 (2018), 17–36.
[23] J. Marlow, L. Dabbish, and J. Herbsleb. 2013. Impression Formation in Online Peer Production: Activity Traces and Personal Profiles in GitHub. In *Conference on Computer Supported Cooperative Work (CSCW)*. 117–128.
[24] N. McDonald and S. Goggins. 2013. Performance and Participation in Open Source Software on GitHub. In *Conference on Human Factors in Computing Systems (CHI)*. 139–144.
[25] C. Miller, D. Gray Widder, C. Kästner, and B. Vasilescu. 2019. Why do People Give Up Flossing? A Study of Contributor Disengagement in Open Source. In *International Conference on Open Source Systems*. Springer, 116–129.
[26] K. W. Miller, J. Voas, and T. Costello. 2010. Free and Open Source Software. *IT Professional* 12, 6 (2010), 14–16.
[27] S. Minto and G. C. Murphy. 2007. Recommending Emergent Teams. In *International Conference on Mining Software Repositories (MSR)*. 5–5.
[28] R. Mokarzel Filho, M. de Souza Pereira, C. Faria, and G. Lemos. 2019. Collaboration Tool for Distributed Open Source Verification. In *International Conference on Global Software Engineering (ICGSE)*. IEEE, 139–142.
[29] J. Oliveira, E. Fernandes, G. Vale, and E. Figueiredo. 2017. Identification and prioritization of reuse opportunities with JReuse. In *International Conference on Software Reuse (ICSR)*. Springer, 184–191.
[30] J. Oliveira, M. Viggiato, and E. Figueiredo. 2019. How Well Do You Know This Library? Mining Experts from Source Code Analysis. In *Brazilian Symposium on Software Quality (SBQS)*. 49–58.
[31] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. 2013. Creating a Shared Understanding of Testing Culture on a Social Coding Site. In *International Conference on Software Engineering (ICSE)*. IEEE, 112–121.
[32] G. Pinto, I. Steinmacher, and M. Gerosa. 2016. More Common Than You Think: An In-Depth Study of Casual Contributors. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 112–123.
[33] H. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu. 2019. Going Farther Together: The Impact of Social Capital on Sustained Participation in Open Source. In *International Conference on Software Engineering (ICSE)*. IEEE, 688–699.
[34] I. Qureshi and Y. Fang. 2011. Socialization in Open Source Software Projects: A Growth Mixture Modeling Approach. *Organizational Research Methods* (2011), 208–238.
[35] A. Rastogi and N. Nagappan. 2016. Forking and the Sustainability of the Developer Community Participation–An Empirical Investigation on Outcomes and Reasons. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 102–111.
[36] D. Riehle, J. Ellenberger, T. Menahem, B. Mikhailovski, Y. Natchetoi, B. Naveh, and T. Odenwald. 2009. Open Collaboration within Corporations using Software Forges. *IEEE software* 26, 2 (2009), 52–58.
[37] S. Schulze S. Stănciulescu and A. Wąsowski. 2015. Forked and Integrated Variants in an Open-Source Firmware Project. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 151–160.
[38] A. Schilling. 2014. What Do We Know About Floss Developers' Attraction, Retention, and Commitment? A Literature Review. In *Hawaii International Conference on System Sciences (HICSS)*. 4003–4012.
[39] A. Schilling, S. Laumer, and T. Weitzel. 2012. Who will Remain? an Evaluation of Actual Person-Job and Person-Team Fit to Predict Developer Retention in FLOSS Projects. In *Hawaii International Conference on System Sciences (HICSS)*. IEEE, 3446–3455.
[40] I. Steinmacher, A. P. Chaves, and M. A. Gerosa. 2010. Awareness Support in Global Software Development: a Systematic Review based on the 3C Collaboration Model. In *International Conference on Collaboration and Technology (CRIWG)*. Springer, 185–201.
[41] I. Steinmacher, G. Pinto, I. Wiese, and M. Gerosa. 2018. Almost There: A Study on Quasi-Contributors in Open-Source Software Projects. In *International Conference on Software Engineering (ICSE)*. IEEE, 256–266.
[42] I. Steinmacher, C. Treude, and M. A. Gerosa. 2018. Let Me In: Guidelines for the Successful Onboarding of Newcomers to Open Source Projects. *IEEE Software* 36, 4 (2018), 41–49.
[43] M. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. 2014. The (r) Evolution of Social Media in Software Engineering. In *Proceedings of the on Future of Software Engineering*. 100–116.
[44] J. Tsay, L. Dabbish, and J. Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *International Conference on Software Engineering (ICSE)*. 356–366.
[45] Y. Tymchuk, A. Mocci, and M. Lanza. 2014. Collaboration in Open-Source Projects: Myth or Reality?. In *International Conference on Mining Software Repositories (MSR)*. 304–307.
[46] B. Ulziit, Z. A. Warraich, C. Gencel, and K. Petersen. 2015. A Conceptual Framework of Challenges and Solutions for Managing Global Software Maintenance. *Journal of Software: Evolution and Process* (2015), 763–792.
[47] M. Viggiato, J. Oliveira, E. Figueiredo, Pooyan Jamshidi, and C. Kästner. 2019. How Do Code Changes Evolve in Different Platforms? A Mining-based Investigation. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 218–222.
[48] M. Viggiato, J. Oliveira, E. Figueiredo, P. Jamshidi, and C. Kästner. 2019. Understanding Similarities and Differences in Software Development Practices Across Domains. In *International Conference on Global Software Engineering (ICGSE)*. IEEE, 84–94.
[49] G. Von Krogh, S. Haefliger, S. Spaeth, and M. Wallin. 2012. Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS quarterly* (2012), 649–676.
[50] J. Whitehead. 2007. Collaboration in Software Engineering: A Roadmap. In *Future of Software Engineering*. IEEE, 214–225.
[51] Y. Yue, I. Ahmed, Y. Wang, and D. Redmiles. 2019. Collaboration in Global Software Development: An Investigation On Research Trends and Evolution. In *International Conference on Global Software Engineering (ICGSE)*. IEEE Press, 68–69.
[52] S. Zhou, B. Vasilescu, and C. Kästner. 2019. What the Fork: A Study of Inefficient and Efficient Forking Practices in Social Coding. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE/ESEC)*. 350–361.