Received 08 January, 2021; | Revised <day> <Month>, <year>; | Accepted <day> <Month>, <year>

DOI: xxx/xxxx

Perceptions of Open-Source Software Developers on Collaborations: An Interview and Survey Study

Kattiana Constantino[1] | Mauricio Souza[2] | Shurui Zhou[3] | Eduardo Figueiredo[1] | Christian Kästner[4]

[1]Federal University of Minas Gerais, Minas Gerais, Brazil
[2]Federal University of Lavras, Minas Gerais, Brazil
[3]University of Toronto, Ontario, Canada
[4]Carnegie Mellon University, Pennsylvania, United States

Correspondence
*Kattiana Constantino Email: kattiana@dcc.ufmg.br

Abstract

With the emergence of social coding platforms collaboration has become a key and dynamic aspect to the success of software projects. In such platforms, developers have to collaborate and deal with issues of collaboration in open-source software development. Although collaboration is challenging, collaborative development produces better software systems than any developer could produce alone. Several approaches have investigated collaboration challenges, for instance, by proposing or evaluating models and tools to support collaborative work. Despite the undeniable importance of the existing efforts in this direction, there are few works on collaboration from developers' perspectives. In this work, we aim to investigate the perceptions of open-source software developers on collaborations, such as motivations, techniques, and tools to support global, productive, and collaborative development. Following an ad hoc literature review, an exploratory interview study with 12 open-source software developers from GitHub, our novel approach for this problem also relies on an extensive survey with 121 developers to confirm or refute the interview results. We found different collaborative contributions, such as managing change requests. Besides, we observed that most collaborators prefer to collaborate with the core team instead of their peers. We also found that most collaboration happens in software development (60%) and maintenance (47%) tasks. Furthermore, despite personal preferences to work independently, developers still consider collaborating with others in specific task categories, for instance, software development. Finally, developers also expressed the importance of the social coding platforms, such as GitHub, to support maintainers and contributors in making decisions and developing tasks of the projects. Therefore, these findings may help project leaders optimize the collaborations among developers and reduce entry barriers. Moreover, these findings may support the project collaborators in understanding the collaboration process and engaging others in the project.

KEYWORDS:
Open-Source Software Projects; Collaborative Software Development; Collaboration in Software Development; Distributed Collaboration; Sustained Developer Community Participation

1 | INTRODUCTION

Collaboration has always been important to the success of large software systems [1]. With the emergence of social coding platforms, such as GitHub (*https://github.com/*), collaboration has become even more important in open-source software development. Social coding platforms often involve globally distributed developers collaborating to develop software systems more dynamically. In this context and in the social coding networking environment, many software engineers have to collaborate and deal with issues from geographical, temporal, cultural, and language diversity [1,2]. Collaboration is key to promote sustainability, for instance, in terms of technical and longevity [3,4] by enhancing the motivation, engagement, and retention of developers in the project [5,6,7]. However, collaboration is challenging since there are many barriers to collaborative development in open-source software projects. As a result, this area has attracted increased interest from researchers and practitioners in recent years [8] .

Several previous studies [8,9,10,11,12,13] tried to understand collaboration practices and communication patterns in software development. Others [14,15,16,17] investigated models 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 software engineering research and practice. In a different direction, other previous work proposes and evaluates models [18,19], theories [20,21], and tools [14,22,23] to support collaboration and to help developers in collaborative development tasks. For instance, Lanubile et al. [14] enumerated several existing tools to support collaborative work along the software product life-cycle. Despite the undeniable importance of the existing efforts in this direction, there are few works to understand how and why collaboration happens in open-source software development from the perception of developers. That is, it is necessary more studies with developers to understand the reasons and barriers involved in collaborative software development. It is therefore important for researchers and practitioners to understand these challenges and barriers to provide suitable support and tools for their collaborators in open-source software development.

In this work, we aim to understand how and why collaboration happens from perceptions of developers in open-source software development projects. More precisely, this study investigate the motivations, processes, interactions, and barriers involved in collaboration during open-source software development to provide strategies to retain developers in the projects and, consequently, guarantee their sustainability. To achieve this aim, first, we performed an *ad hoc* literature review to identify gaps and motivate studies. Besides, following an exploratory interview study with 12 active software developers of large collaborative projects, our novel approach for this problem also relies on an extensive survey with 121 developers to confirm or refute the interview results.

In the interview study, we designed a semi-structured interview protocol and interviewed twelve experienced software developers. Each interview lasted between 30 to 60 minutes and was guided by three main questions about (i) what motivates developers to collaborate, (ii) the collaboration process adopted, and (iii) challenges and barriers involved in collaboration. Our key 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 interview study are related to non-technical, rather than technical issues.

We cross-validated our results aiming to obtain a better understanding of how collaboration happens in open-source software development, based on developers' perceptions. As a result, we identify and check the main tasks and opportunities for collaborations. We designed and performed an opinion survey that was emailed to 1,113 developers from 50 open-source projects hosted on GitHub. As a result, we received 121 answers from developers (response rate around 11%). All participants are developers familiar with their projects and made their last commit within the last year. Our survey analysis indicates that most developers prefer to work collaboratively with the core team. Moreover, software development (60%) and maintenance (47%) tasks are the main reasons to work collaboratively with other developers. Furthermore , despite personal preferences to work independently, developers still consider collaborating with others in some tasks, such as software development and maintenance tasks. Finally, developers also expressed the importance of the social coding platforms, such as GitHub, to support leaders and contributors in making decisions and developing tasks of the projects.

In this paper, we build in top of our previous work [24] for improving developers' collaboration and extend it with a quantitative study to cross-validate how collaboration happens. Our main contributions can be summarized as follow.

1. We conducted an *ad hoc* literature review to identify gaps, motivate and evidence, when suitable, statements of the many developers;
2. We designed and performed a qualitative to investigate how collaboration happens in open-source development;
3. We designed and performed a quantitative study with 121 developers to cross-validate the interview results;

4. We provide empirical evidence about collaboration in open-source software development faced by active and open-source software developers;

5. We highlight the opportunities for improving collaboration and discuss its benefits in highly collaborative environment, such as GitHub;

6. We obtain insights into the contributors' work practices and encountered opportunities to improve the collaborations based on collaborative tasks and contributor preferences.

We hope the open-source software communities and researchers could take advantage of this paper to better comprehend the opportunities of collaboration among developers and design strategies to make greater use of them. We believe that these findings may help project maintainers optimize the collaboration among developers, reduce the barriers to entry, and support the engagement of collaborators in the project. The rest of this paper is organized as follows. Section 2 motivates this paper with some background information and related work. Section 3 presents the protocol we followed to conduct our research methods. Section 4 reports and discusses the main results of the interview study. Section 5 presents the results of the survey study. Section 6 discusses our main findings and implications from both studies. Section 7 explains some potential threats to the study validity and decisions we made to mitigate them. Finally, Section 8 concludes this paper by pointing out directions for future work.

2 | BACKGROUND AND RELATED WORK

In this section we present the literature discussion on open-source software (OOS) and its development model. In developing a theoretical basis for this paper, we have drawn from collaboration literature [25,26,27] to discuss how collaboration occurs in the context of collaborative software 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 [28,29,30]. 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 [20,21]. 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 [29].

In this research, we focus on open-source software development specially in projects hosted on GitHub. GitHub is a Web-based code-hosting service that uses Git distributed version control system [31]. Furthermore, GitHub is available for free and aims to promote collaboration in software development. As of June 2021, GitHub reports having over 65 million users and more than 200 million repositories [1]. GitHub has become an essential tool in technology areas that demand collaboration, such as globally distributed software development [32]. 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 [9]. In this model, developers "fork" the repository to create an own copy of the source code and make changes without affecting the upstream development. Next, developers 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 code changes.

2.2 | Related Work

Several studies have discussed diverse aspects of collaboration in open-source software development projects [9,10,33,34,35,36]. However, each one focuses on distinct aspects and perspectives. For instance, two studies investigated the "fork & pull" development model from the integrators' [9] and contributors' [10] perspectives. Both studies investigated practitioners' working habits and challenges alike. Gousios et al. [9] investigated which motivation keep contributors working on the project and how to not discourage them. In a complementary study, Gousios et al. [10] 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

[1] https://github.com/about

experienced developers of open-source software projects hosted on GitHub. We aim to know how the community supports growing supplies of developers through spontaneous collaboration.

Zhou et al.[33] 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 and survey with experienced software developers to understand the motivations and barriers related to the collaboration process in fork based development, and how these factors impact developers' willingness to collaborate. Therefore our research aims at understanding how and why collaboration occurs, in the perspective of 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[36,37]. Some studies investigate the use of GitHub's social features to understand developer behavior, to evaluate projects' success, and to identify potential collaboration opportunities. For instance, Marlow et al.[38] 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.[39] found that both technical and social information influence the evaluation of contributions. McDonald et al.[40] noted that one of the main reasons for the increasing number of contributions and developers to a project are features provided by GitHub.

In the present study, we focus on how the project developers perceive the collaboration process. This work may include but does not focus on the motivations and strategies to find new developers to a project. Additionally, while Tsay's work[39] focuses on how technical and social information influence the evaluation of contributions, our study focuses on identifying social aspects that impact on the developers' willingness to contribute.

3 | RESEARCH METHOD

This section presents the research methods we followed in both literature review, interview and survey studies.

3.1 | Goal and Study Phases

The goal of this research work is to understand how and why collaboration happens from developers' perspectives in open-source software development projects. To achieve this goal, we design a research study divided into three phases as shown in Figure 1. In phase 1, we conducted an *ad hoc* literature review to identify gaps and motivate studies of the next phases. In phase 2, we planned and executed an interview study with developers. In phase 3, we designed and performed a survey study with developers of open-source software projects. We discuss each study phase as follows.

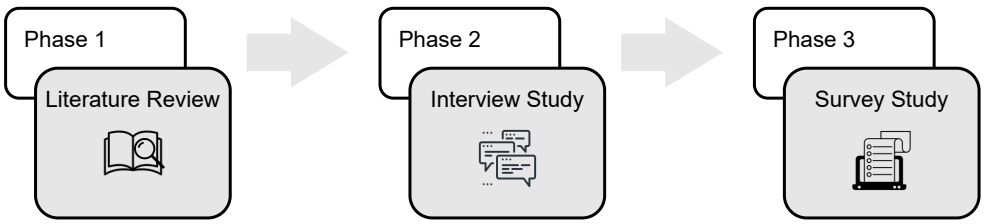


FIGURE 1 Study phases.

Phase 1. At first, as presented in Figure 1, we performed an *ad hoc* literature review on collaboration in open-source software projects. The goal was to understand the conceptual background and emergent issues for further investigation (Section 2). Furthermore, the literature analysis also helps us to illustrate, when appropriate, statements of the various open-source developers from interview and survey studies.

Phase 2. In this phase, we performed an interview study (Figure 1). We carried out an interview study to explore the collaborations, processes, communication channels, and barriers and challenges faced by developers in open-source software development. The interview study was focused on understanding (i) what motivates developers to collaborate, (ii) the collaboration process adopted, and (iii) challenges and barriers involved in collaboration. For this study, we invited GitHub developers and performed individual interviews that were conducted face to face or with Skype.

Each interview was transcribed verbatim. For the data analysis, we applied standard coding techniques for qualitative research [41,42]. Section 3.2 explain the details of our method in this phase.

Phase 3. In our survey study (Figure 1), we performed an opinion survey to cross-validate the interview results to understand how open developers are to work collaboratively based on their behaviors and identify and check the main tasks and opportunities for collaborations. To this end, we selected projects hosted on GitHub and extracted data from the original repository and its developers (contributing forks). Additionally, we choose the surveyed participants and their similar developers based on the same interest in code changes. We sent personalized emails to 1,113 developers from 50 projects. In total, 121 participants answered the survey (response rate around 11%). We executed the survey study the steps detailed in Section 3.3.

3.2 | Interview Study Design

This section describes the goals, research questions, and methodology for the interview study. For the data collection, we used semi-structured interviews conducted face to face or with Skype and applied standard coding techniques for qualitative research as presented by Figure 2.

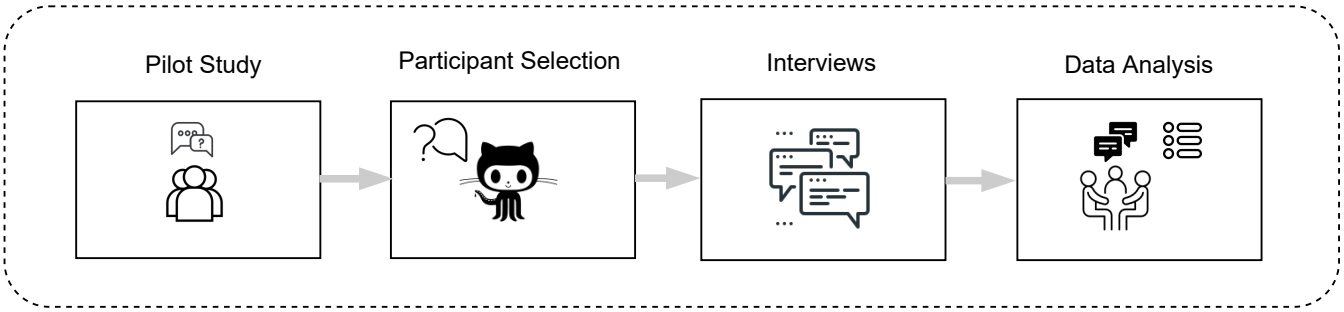


FIGURE 2 Interview study overview.

Goal and Research Questions. 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 [43,44]. Developers may have expectations concerning the kinds of contributions they want and concerning the roles and responsibilities of each party [45]. Sometimes, the term "collaboration" may have different meanings to the developers and others who may be directly or indirectly involved. Therefore, *the main goal of this interview study is to understand how and why collaboration happens from developers' perspectives in open-source software development projects hosted on* GitHub. We need to identify which possible motivations, processes, interactions are common in software development projects. Furthermore, we aim to comprehend the challenges and barriers faced by developers. We expect that it could help project maintainers to optimize collaboration opportunities and attract more community members. To obtain a understanding on how collaboration in software development projects happens, we interviewed twelve experienced developers 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? With RQ1, we want to investigate the individual motivation of developers. We also discuss the reasons for working independently mentioned by participants.

RQ2 - How does the collaboration process occur? To make a better analysis, we refine the RQ2 in the following sub-questions.

RQ2.1 - What are the collaborative contributions? 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? With RQ2.2, we want to comprehend how the developers organize themselves, in particular, considering their roles and interest in different types of collaborative contributions.

RQ2.3 - What communication channel do developers use to support collaboration among them, and how? With RQ2.3, we want to know what communication tools are often used by developers. Besides, we want to understand how communication occurs to support the collaborative contributions.

RQ3 - What are the challenges and barriers in working collaboratively? With RQ3, we are interested to know which challenges and barriers are faced by developers when working collaboratively. Besides, we want to know what are the options we could pursue in order to optimize collaboration.

Pilot. We met with all researchers involved in this study to discuss an interview protocol. As shown in Figure 2, we first decided to run a pilot study to select three developers of open-source projects by convenience. With these pilot interviews, we refined the protocol, questions, focus, and time for interviews interactively. All participants in the pilot interviews are Ph.D. candidates in Computer Science of our research group, while they are also software developers and technical developers 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.

Participant Selection. In order to select participants, we first mined Portuguese speakers (the language of the first author) and frequent GitHub developers with more than 500 commits in the last three years using GitHub's REST API v3[2], in line with prior studies on GitHub [46,47,48,49]. With this last selection criterion, we try to guarantee that the developer has experienced in open-source development and is currently involved in at least one project. Therefore, we can confidently state our interview questions related to collaboration among developers on open-source development. Next, we sent e-mails to invite the top eighty developers for an interview (see Figure 2). A total of eighteen developers replied to the invitation. However, six of them later cancelled the interviews. Therefore, twelve developers 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#). Ten participants volunteered their time to contribute to their respective project and the other 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. To facilitate and better contextualize the questions for the participants, we focus the discussions based on the project to which the participant contributes the most (see Figure 2). Moreover, the project could help them remembering or focusing answers on their collaborative experience into a project. When the participant is active in multiple projects, we pick the most popular one in terms of stars and forks. Each participant (P#) answered the interview questions focusing on the project to which they collaborated.

TABLE 1 Participants Demographics. Participants' demographic information concern gender, last education status, years of OSS development experience, number of contributions on GihHub for the past three years, previous or current roles in the project, and their project domains.

ID	Gender	Education	OSS Experience	GH Contributions* (last 3 years)	Roles**	Project Domains
P01	M	B.S.	4	>870	TC/U	Compilers/E-business
P02	M	B.S.	2	>1,800	TC	E-commerce
P03	F	B.S.	4	>2,740	TC/SEO	Artificial Intelligence/Education
P04	M	M.S.	5	>600	TC/ME	E-business
P05	M	B.S.	9	>4,580	TC	E-business Infrastructure
P06	M	B.S.	11	>3,900	TC	E-Collaboration Tools
P07	F	M.S.	7	>2,550	FPM/TC/SEO	Civil Participation/Datamining
P08	M	Ph.D.	10	>3,500	CM, SEO/M/TC	Civil Participation/Datamining
P09	M	M.S.	11	>3,950	ME/TC/SEO	E-commerce/WebSecurity
P10	F	M.S.	4	>2,500	CM/ME/TC/SEO	Datamining
P11	M	B.S.	8	>4,300	FM/TC/SEO	Datamining/E-commerce
P12	M	M.S.	5	>530	PM/TC/M/U	Artificial Intelligence

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

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

TABLE 2 Interview questions.

ID	Questions
IQ1	In the project hosted on GitHub which you are mostly involved in, do you prefer to work alone or with other developers? Why?
IQ2	In the *<project name>* project, what kind of collaboration is the most common among developers?
IQ3	Are there other collaborations that are important, but little explored in the *<project name>* project that you participate in?
IQ4	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?
IQ5	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?

Data Analysis. As detailed in Figure 2, we qualitatively analyzed the interview transcripts using standard coding techniques for qualitative research [41,42]. Two authors analyzed the responses individually and marked relevant segments with codes (tagging with keywords). Later, the authors compared their codes to reach consensus and tried to group these codes into relevant categories. With the support of the other three authors, the codes and categories were discussed to extract key findings and theories.

3.3 | Survey Design

In order to study how the collaborations happen among developers, we performed an opinion survey to get more insight into the results from the interview study (Section 4). In the following sections, we describe our goal and survey research questions, and the steps of our survey study, as summarized in Figure 3.

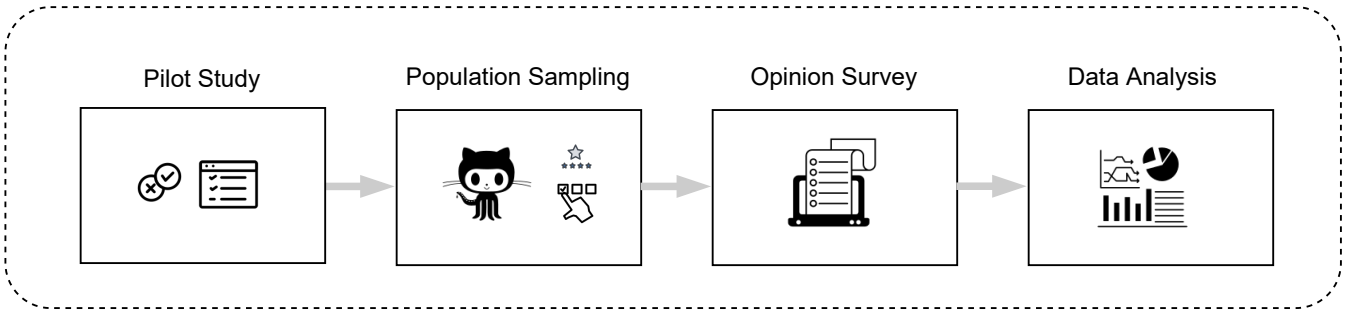


FIGURE 3 Survey study overview.

Goal and Survey Research Questions. In the Section 4, some interviewees (P01, P02, P06, and P07) reported they have a tendency to mutual interactions with the senior developers of the project instead of interactions with other developers. The main reason is that the core team members have a more significant level of involvement with the project. Experienced community members are also more prone to sharing knowledge and mutual interactions with other members than other non-core team developers. However, all collaboration is essential for the sustainability of the project [36]. Hence, all contributions should be valuable and encouraged [44,50,51]. For instance, casual developers or newcomers also contribute with bug fixes, new features, documentation, user support, and other important tasks. Therefore, we performed a survey study aiming to cross-validate our results and understand how collaboration happens in software development projects. Based on developers' behavior, how open they are to work collaboratively with others, and which main tasks are to further opportunities of collaborations in the software development project. To achieve this goal, we consider the following survey research questions (SRQs):

SRQ1 - How open are developers to work collaboratively? With SRQ1, we want to know how developers prefer to work and how they actually work, in respect to: working alone, working with the core team, or working with other developers.

SRQ2 - What types of activities do developers prefer to collaborate? With SRQ2, we aim to investigate the types of activities that are more likely to improve collaboration among developers.

SRQ3 - What other perceptions do developers provide about collaborations? With SRQ3, we aim to know which are the observations or developers' perceptions provided in the open questions of the survey questionnaire.

Pilot Study. We executed a pilot study in order to validate the study protocol and the questionnaire. For the pilot study, we sent 158 invitation emails to answer our survey. In total, 10 participants completed the survey in our pilot study. The main improvement from the pilot study was related to the questionnaire. Initially, we asked participants about their current collaboration practices (questionnaire item SQ1, in Table 4). However, during the pilot study, we noticed that the current work practice may not reflect the participant preferences. Therefore, we added the questionnaire item SQ2. Another change is related to the number of recommended developers presented for each participant. Firstly, we started with three developers for each participant. However, we realized that it turns the questionnaire too long and very complicated for participants to answer. For each of the recommended developers, the participant should check the relevant files for both and answer the questions. Thus, participants gave signals indicating giving up on completing the survey, for instance, leaving the questions blank related to the last recommended developers. Thereby, we decided to reduce for one recommended developer per participant. We also decided to use developer recommendations based on similar code changes [52,53,54] to obtain concrete collaboration scenarios across developers. Accordingly, the developer would collaborate with existing or possible concrete situations.

Population Sampling. The target population sample is composed of developers working in contributing forks of open-source projects hosted on GitHub, as presented by Figure 3. Therefore, we first selected candidate open-source projects from GitHub according to the following criteria: they must be public, have at least 1K stars, and must active with ongoing development. This yielded 3,464 repositories, from which we randomly selected 50. The selected repositories include many famous and popular projects, such as MongoDB [3] (Version 4.4, 19K stars, 7K forks and C++ as predominate language), Eclipse Deeplearning4J [4] (Version 1.0.0, 12K stars, 5K forks and Java as predominate language), Pandas [5] (Version 1.1.3, 27K stars, 11K forks and Python as predominate language), and Vue [6] (Version 3.0, 16K stars, 3K forks and TypeScript as predominate language). Afterward, we automatically collected the name and emails of project developers with at least four accepted commits (with their last commit within the last year). Based on the last commit date, we try to guarantee that the developers may be currently involved or still familiar with the project [55,56].

Survey Demographic Information. Through the GitHub REST API[7], we collected the public and available demographic information of the survey participants in their GitHub profiles as follows. Table 3 show the other demographic information such as the number of followers and following as indicators of social interaction and popularity of the participants. They have a median of 20 followers and 4 following. Additionally, we collected the number of public repositories they have interested to follow or participated in, and the number of the contributions in the last year. Likewise, they have a median of 31 of interest in public repositories and 797 commits in the past three years. Moreover, Figure 4 presents the location of the survey participants that many of them were from the USA, UK, and India, but there was also a wide distribution among other unspecified countries, because this information was not available (total of 40). Finally, we mined the public and available roles and technologies information auto declared by participants in their bios. The roles are Compiler Developer, Community Team Manager, Data Engineer, Data Scientist, Front-End Developer, Maintainer, Programmer, Researcher, Software Engineer, Software Developer, Tutor, UI Designer, and Web Developer. The Languages used are Java, JavaScript, MATLAB, Python, Rust, and TypeScript. Besides, the technologies they use involve Apache Framework, API maker, BootStrap, Build tools, CLI, Dataviz, Gatsby, Git, JIT compilation, MongoDB, Node.js, Plugins, and React/Redux.

TABLE 3 Participants demographics.

	Mean	St. Deviation	Min	Median	Max
Followers	77	1,983	0	20	1,505
Following	13	20	0	4	116
Public Repos	54	67	2	31	512
Contributions	1,715	2,249	38	797	11,093

**We recorded the values in July of 2021.*

[3] https://github.com/mongodb/mongo
[4] https://github.com/eclipse/deeplearning4j
[5] https://github.com/pandas-dev/pandas
[6] https://github.com/vuejs/vue-next
[7] https://docs.github.com/en/rest

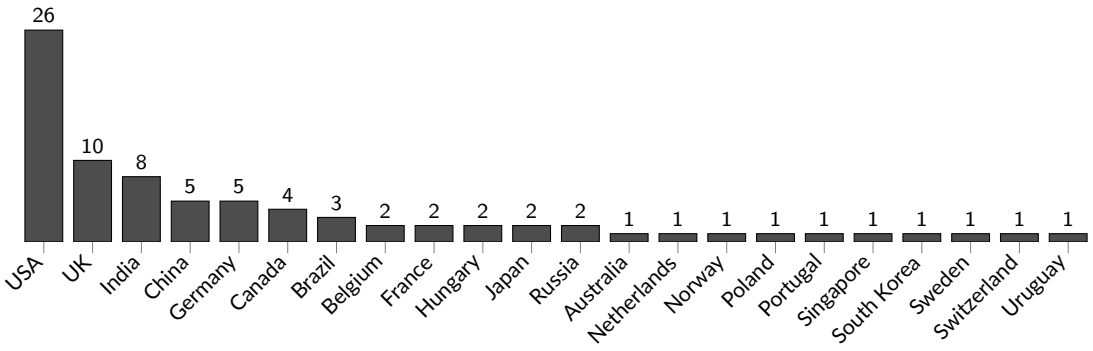


FIGURE 4 Location of the survey participants.

TABLE 4 Survey questions.

ID	Questions
SQ1	How are you working on the project? [ ] In collaboration with the core team [ ] In collaboration with owners of other forks [ ] Independently
SQ2	How do you prefer to work on the project? [ ] In collaboration with the core team [ ] In collaboration with owners of other forks [ ] Independently
SQ3	I worked or may work in partnership with the owner of this fork on some tasks of the project, such as: [ ] software development tasks (e.g., feature or test suites developing, or code review) [ ] issues management tasks (e.g., reporting, triaging, or solving issues) [ ] community building (e.g., motivating/recruiting collaborators, or promoting/directing the project) [ ] maintainability (e.g., improving code/project quality) [ ] mentorship/knowledge sharing (e.g., for giving/asking help to develop a new feature or fix an issue) [ ] repository management tasks [ ] I did not work or may not work in partnership with the owner of fork [ ] other (open question)
SQ4	Other important observations or suggestions (open question)

Opinion Survey Development. We created a questionnaire on Google Forms composed of questions about the project developers' perception of their collaborative works with other project community members (see Figure 3). Table 4 shows the list of questions for the final version (after the pilot study). Finally, we seek any additional information that participants would like to add. We ask two open questions. First, what additional collaborative tasks that participants may wish to work in work collaboratively with. Second, in the last question, the participants could add any comments to this study. These additional questions aim to catch any important issues to participants not asked in the questionnaire. We sent personalized emails to 1,113 developers from 50 projects. In total, 130 participants completed the opinion survey. We analyzed their responses, and filtered incomplete responses, excluding 9 participants. Finally, we nicknamed the surveyed participants from S01 to S121. Our goal is to use these nicknames while keeping the anonymity of the participants, for analyzing their feedback from open questions.

Data Analysis. As detailed in Figure 3, we collected quantitative and qualitative data from the online survey (concerning developer opinions and preferences) and objective data such as demographic information. Section 5 presents descriptive and qualitative analysis to answer the research questions of this work.

4 | RESULTS OF THE INTERVIEW STUDY

This section presents the results of the interviews according to each research question proposed in Section 3.2. Figure 5 summarizes these results from the perceptions of developers on collaborations that are detailed as follows.

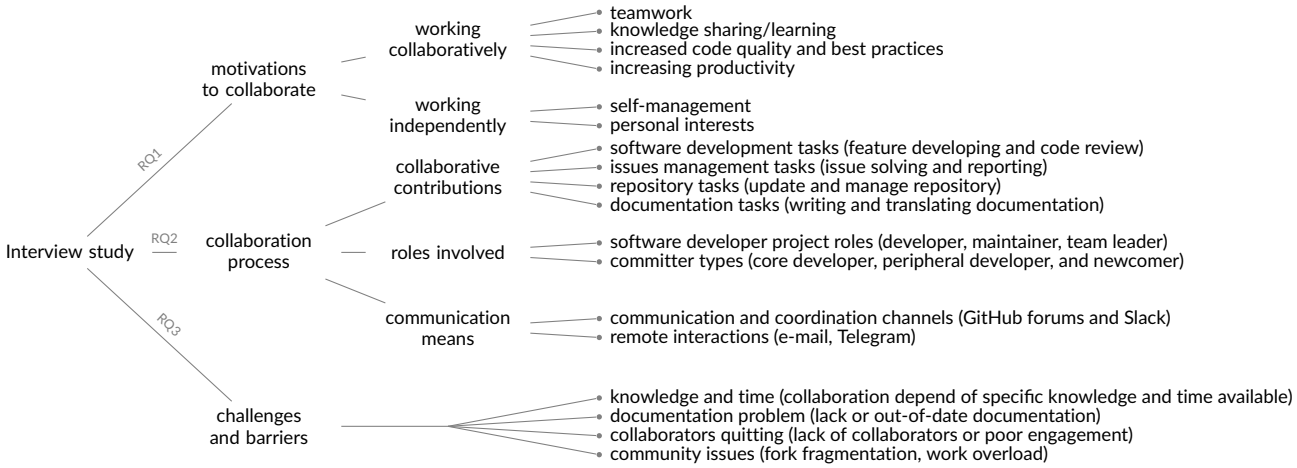


FIGURE 5 The main findings identified in the interview study.

4.1 | RQ1 - What are the motivations to work collaboratively?

The first research question aims to investigate the individual motivation of open-source software developers. 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 5, we present the reasons why developers 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., strengthening the synergy of teamwork, increasing the quality of the code and reinforcing best practices, benefits from sharing knowledge and learning, and increasing productivity). 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 5, we also show the reasons why developers 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 to working in groups, such as the dependence of other developers and the time-consuming nature of collaborative work, for example, when developers need to make a joint decision to solve an issue or about a new release. These drawbacks could postpone the project results. P10 reported: *"You depend on the result of other people and of more time to solve an issue or make a decision. It can be a problem."* P08 was one of the participants who remarked that sometimes they like to work with other developers, like *"pair programming,"* and other moments they prefer to work in own pace: *"I prefer to work alone, in my own time, for fun, without any pressure."* Also, 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 [44,57,58,50]. We realized that the participants reported some of their experiences and possible outcomes of the collaborations in the project. Besides, they have appreciation expectations and want their contributions to be valued and recognized with financial benefit or not.

TABLE 5 Reasons for working collaboratively and independently.

Category	Codes	Part.(#)	Freq.
Working Collaboratively	Teamwork	5	15
	Knowledge sharing/learning	4	6
	Increased code quality and best practices	4	16
	Productivity	2	2
	Professional activities	1	1
Working Independently	Self-management	4	7
	Personal interests	2	2
	Different timezone	2	2
	Independent tasks	2	2
	Reaching consensus can be time-consuming	2	2
	Dependence of others	1	1
	Collaboration for demand	1	1
	For non-core contributor	1	1

`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?

The second research question aims at characterizing the collaboration process in collaborative software development projects. Therefore, we aim to understand the collaborative contributions, the involved people, the collaboration channels, and the collaboration process.

RQ2.1 - Collaborative Contributions. In Table 6, 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, a previous study [59] confirms that issue fixing ("fix bug") and feature development ("add new feature") are the top motivations for developers. 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 6. The pull-based development model helps the control of the contributed code quality by selecting new contributions, and incremental code reviews [39,51]. Thereby, developers perceive each task as a contribution to the projects, not only the code itself. We observe that some tasks are suitable for a developer to do alone. However, some developers prefer looking for collaboration with other developers to perform these tasks. As seen in Table 6, `Software Development Tasks` and `Issues Management Tasks` are the most prominent categories regarding what participants understand as collaborative contributions into projects.

RQ2.2 - Roles Involved. In Table 7, we show the roles of developers involved in collaborations. The first column of Table 7 presents the two categories we identity: `Software Developer Project Roles` and `Committer Type`. The analysis of Table 7 revealed four perspectives on the roles involved in collaborative software 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 developer", "core developer", and "newcomers" as presented in Table 7. Interestingly, these types of committers match the terminology used in recent research paper [60]. This finding indicates an alignment between software development research and practice. There are a set of responsibilities attributed to these developers, as stated by P05 and supported by P03 and P08: *"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 developers. The main desired characteristics are "availability to collaborate" and "engagement", also cited in literature [43,55,59]. For instance, availability includes the developer's ability to conciliate the volunteer aspect of collaborating in collaborative software development projects with their formal employment schedule. P07 affirmed (supported by P02): *"developers need to be very engaging to work on an open-source project."*

TABLE 6 Categories and codes for the types of collaborative contributions in the 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	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 7 Categories and codes for the people roles involved in collaboration in collaborative software 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

RQ2.3 - Collaboration Means. In Table 8, we present the codes related to the communication channels that developers used to support collaboration among them in open-source development projects. GitHub issues (forum) was the most cited. More than one participant also mentioned Slack, GitHub Pull Requests, e-mail, Telegram, and Gitter as communication channels for collaboration. Prior works reported similar findings for open-source software projects [15,43] and commercial projects [61]. They pointed out that developers usually communicate through text messages.

The developers stated that each project is free to communicate, and in some cases, without clear rules regarding communication tools usage. For instance, participant P12 declared, *"Sometimes, I start chatting on the forum, but if the conversation extends, I switch to email."* Besides, participant P08 stated, *"If I know the developer, I talk directly. However, sometimes I open the issue."* Finally, P10 explained that all communication channels used during their works: *"Large projects that I worked on already have a specific communication channel, if they do not have, I use* GitHub *Issues. I also use Gitters as a chat tool, Discuss as a tool for deeper discussions,* GitHub *Issue as a way to record bugs and improvements, and some other projects I used slack as well."* While it could be advantageous to have several communication tools, it could also cause data inconsistency. These findings align with the importance of defining the role of the communication channels to use them for different purposes within a team to improve collaboration [62,63].

TABLE 8 Categories and codes for the collaboration channels.

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

Considering that all participants are GitHub users, it is not surprising that they mention this platform as their primary collaboration channel. However, it is interesting to notice that the issue system is an essential element in open-source software collaboration. Thus, it complies with the participants' perception that issue solving contributes significantly to collaborative software development. Bissyandé et al.[64] found a considerable correlation between the number of forks and the number of issues that bring a positive impact on the project. Indeed, through issue reporting, developers help identify and fix bugs, document software code, and enhance the software system. Participants pointed out that they use communication tools in several contribution tasks in the life cycle of the open-source project. For example, they use these tools to direct and promote the project, recruit new developers, ask for help to develop a new feature, discuss solving an issue, 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 being members of the repository's maintainer group."*

`RQ2 - Collaborative Contributions:` 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. `Roles Involved:` 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 developers commit to the project. `Communication Means:` GitHub issues (forum) was the most cited communication tool by developers. Moreover, they usually communicate with text messages. Furthermore, participants pointed out the communication tools used in several contribution tasks in the life cycle of the open-source project. For instance, to ask for help with a new feature, to fix an issue, or to manage a repository.

4.3 | What are the challenges and barriers in working collaboratively?

In order to answer RQ3, we identify the challenges and barriers that would impact on decisions into collaborative software development as presented in Table 9.

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 developers answering questions..."* for the project to survive because without the developers answering questions, the project fails in its first goal, which is to attract developers or maintainers[52]. P01 agrees with P07, but looks for knowledge between the project's developers senior (supported by P02 and P06). *"I have a greater tendency to ask for help from a developer who has contributed for a longer time and who has more excellent knowledge."* Thus, when the core team is aware of these matters, it has the challenge of being available to meet the demands of peripheral and newcomer developers who want to participate in the project. For instance, it can happen as mentoring in a forum or the code review phase. However, developers 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 developer needs) to have time to contribute, after all, everyone has limited time."*

Moreover, P08 expressed their concern related to specific knowledge from non-technical collaborators: "*In my project, I have collaboration not only about coding but also the interpretation of legal laws by lawyers. Then, thinking that* GitHub *is a tool only for the developer community, we have*

TABLE 9 Challenges and barriers faced by participants. We present the categories as follow: `Knowledge and Time`, `Documentation Problem`, `Collaborators Quitting`, 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	Part.(#)	Freq.
Knowledge and Time	Clarifying any questions	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 Problem	Lack of documentation	5	6
Collaborators Quitting	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

extra work in joining lawyers' knowledge to our collaborators. Thus, when a developer analyzes an issue to solve it, they could have that specific knowledge (technical or non-technical)". Indeed, GitHub is a tool for technical and non-technical professionals. Project leaders need to pay attention to that kind of collaboration in their projects. Furthermore, when the developers 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 developers to share their experience. For instance, whether the developer knows a specific functionality, they could develop another 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 developers, and with relevant knowledge. Thus, some strategies to retain these developers need to be done, as P11 explains (supported by P07): *"The networks in OSS projects are essential. Usually, I always try to keep in contact with everyone who collaborates with the project. So, there are punctual contributions, that sometimes we do not talk to the contributor as much. However, when the contribution is more significant, we try to keep talking to get closer because the contributor may have knowledge that can help at another time."* The lack of experience of the quasi-contributors deemed the work unacceptable [57]. The lack of time is one of the reasons for developers to stop contributing to GitHub projects [43]. Lack of time was also identified as the most common barrier to participation faced by peripheral developers [44] and "no time" was one of the reasons for disengagement in open-source software projects [55].

Documentation Problem. 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 developer 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 [50,65,66] 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 overloading the discussion forums with questions quickly clarified in documentation.

Collaborators Quitting. 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 developers 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 developer worked a lot, and we did not even know about it."* Such a problem likely happens because of carelessness with the developers. Another possible explanation concerns the high number of forks in the project, which makes it difficult to know who is working and in what [7]. Hence, to avoid losing developers or collaborations, P07 alerts: *"If volunteer developers 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 centralized only on one core team member. 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 developers have a great challenge to bring these developers back (if possible). For P07, this depends more on the motivation of the developers than on the efforts of the core team: *"If after a while, someone (from the team) gives feedback to the developer again, he will have to be very motivated to return working on the issue. After a while, without working on the code, the developer does not remember the entire context anymore. So, it's too difficult. The developer has to be really motivated to keep contributing with the project"*

Since it may not be possible to have the "lost" developer 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 collaborative software development and requires collaboration. Many development efforts over multiple project versions are wasted, and many bug-fixes are not propagated [33,67]. Consequently, it makes a great set of specialized forks. For projects with a large number of forks, it becomes impracticable [67].

Community Issues - Failing to Comply with the Project's Contribution Guidelines. In order to attract developers and keep them active, project maintainers usually have their code of conduct and guidelines of best practices for contributions. This documentation drives new developers on the community's expectations regarding posture, commitment, and quality of work. However, we see reports on some procedures that developers 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 could be the creation of a set of tests. The developers send the features, but the tests are missing. This situation can cause a delay in the project."* That is, not all developers have a testing culture. Besides, some tests require more time and effort from casual and volunteer contributors. Project owners are aware of these restrictions [50]. Therefore, volunteers for this specific contribution are welcome.

Community Issues - Work Overload. Besides, some contributions demand great efforts from both the newcomer or peripheral developers 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 because of the task complexity."* 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."* This finding reinforces the importance of attracting developers to the sustainability of the projects, as newcomers as experienced developers [5,6,7,12].

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

5 | RESULTS OF SURVEY STUDY

This section describes the results of our survey regarding the three research questions introduced in Section 3.3. Figure 6 summarizes these results that are detailed as follows. We analyzed 121 responses from 130 participants. We decided to exclude 9 participants that left some questions without answers, i.e., incomplete questionnaires. We also do not include any responses from our pilot study.

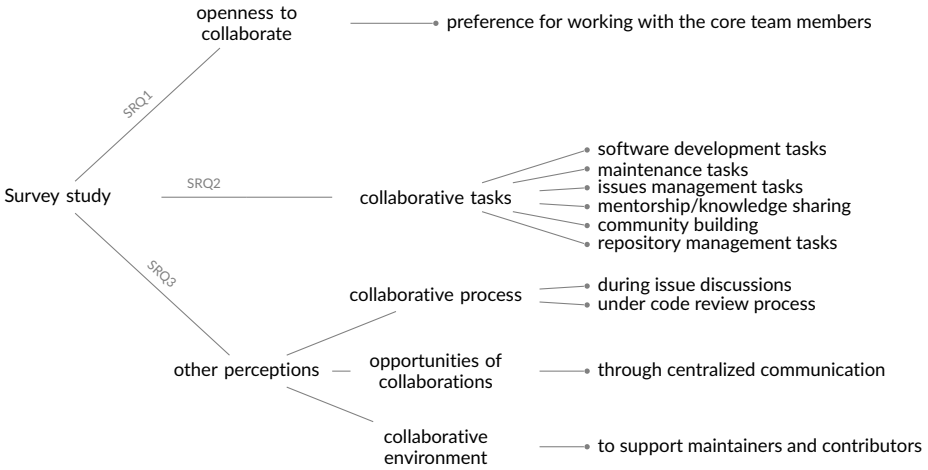


FIGURE 6 The main findings identified in the survey study.

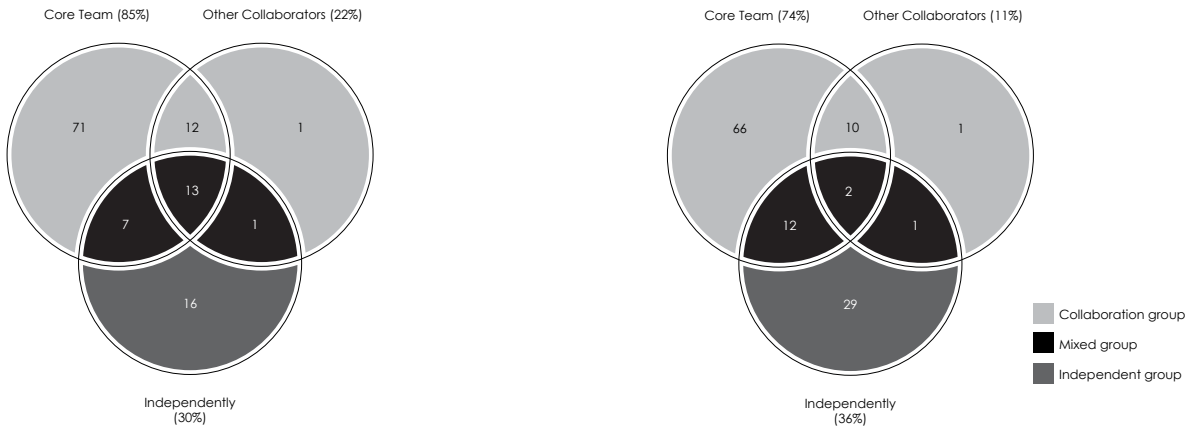
5.1 | SRQ1 - How open are developers to work collaboratively?

To answer the SRQ1, we used the participants responses for the survey items SQ1 (*How are you currently working on the project?*) and SQ2 (*How do you prefer to work in the project?*) (see Table 4). They had the following options to answer (more than one option is allowed): (i) `In collaboration with the core team`, (ii) `In collaboration with the other developers (owners of forks)`, and (iii) `Independently`. Figures 7a and 7b use Venn diagrams to present the responses for their preferences and the actual practice of collaborative work, respectively. The intersections illustrate participants that choose more than one options. For their preferences, Figure 7a indicates that most participants (85%) prefer to work collaboratively with the core team, 30% prefer to work in independent tasks, and 22% prefer to collaborate with the other developers (owners of forks). Regarding their actual practices, Figure 7b indicates 74% of the participants actually work collaboratively with the core team, 36% work in independent tasks, and 11% work in collaboration with the other developers (owner of forks).

Figures 7a and 7b also present the participants mapped into groups, based on collaborative work categories, as follows: Collaboration Group, Independent Group, and Mixed Group. Participants from `Collaboration Group` work or prefer to work strictly in collaboration, and participants from `Independent Group` work or prefer to work strictly independently. The other group work or prefer to work independently or collaboratively (`Mixed Group`). The significant differences between the results (preference versus reality) are between the subsets that involve most of those chosen for mixed and independent works. Figure 8 presents the matched and mismatched expectations of each group. Most of the developers of the `Collaboration` and `Independent` Group are working according to their stated preferences.

Considering these results, we can observe some indicators about which could attract (or not) more collaboration for a project. For example, Farias et al.[68] found that collaborators that contribute regularly and that active participation and long-time interaction with a project are drivers for collaboration. Similarly, Blincoe et al.[69] found that developers are also likely to contribute to new projects after a popular user whom they are following performs any activity on that project. Cai and Zhu[70] argued that experienced developers produce quality codes that could attract more developers to the project. Therefore, developers are interested in improving the project or their knowledge and reputations by interacting or following core team members. Another possible point is that sponsored projects have dedicated developers full-time for the project. Hence, it facilitates the interaction and synergy among these project developers. This finding is also consistent with prior findings that collaborators contributing regularly and actively and long-time participating in a project may attract or engage other collaborators[68,69].

Simultaneously, the `Mixed` Group has the highest number of developers with mismatched expectations (52%). Indeed, developers who are open to all collaboration possibilities have mismatched expectations. In this regard, aside from collaboration barriers, such as lack of interaction



(a) Developer expectations (preference).

(b) Developer reality.

FIGURE 7 Developers' expectations and reality about working collaborative and independently in the software development project.

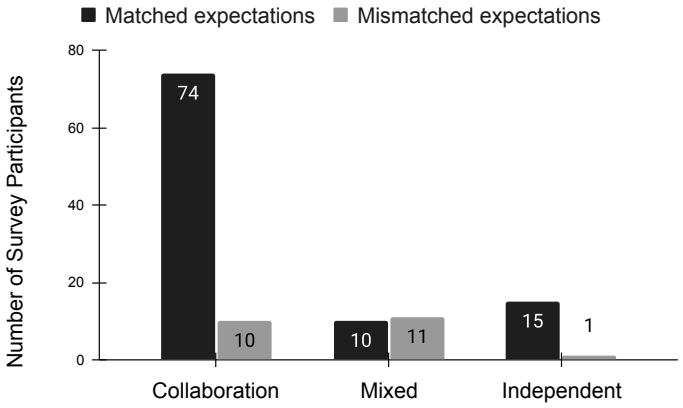


FIGURE 8 Participants clustered in the three groups: `Collaboration`, `Mixed`, and `Independent`.

with project members[21,71], lack of knowledge codebase[10], and others[66], not everyone works well with everyone, conflicting their preferences and styles[72]. Although developers are open to work collaboratively, this situation is not always easy when faced with technical or social barriers.

SRQ1 - The survey results show that most participants (85%) prefer to work collaboratively with the core team, 30% prefer to work in independent tasks, and 22% to value the collaborations with the other developers. Besides, when we asked about how they are working on the project (reality), 74% of those surveyed claimed that they work collaboratively with the core team, 36% work in independent tasks, and 11% answered that they work in collaboration with other developers. The most developers from the `Collaboration` and `Independent Groups` have their expectations matched as working with the same group of their preference. In contrast, most of the developers from the `Mixed` group had their mismatched expectations.

5.2 | SRQ2 - What types of activities do developers prefer to collaborate?

To answer the SRQ2, we used the participants' responses for the item SQ3 of the questionnaire (*I worked or may work in partnership with the owner of this fork on some tasks of the project, such as...*). For each participant, we selected a developer (they may have not contributed with) of a project they already work on. The developer was selected based on the similarities between modifications made in its forks and modifications made by the

participant in its contributions. Thus, we asked participants which types of activities they could work collaboratively with the selected developer. Figure 9 summarizes the responses of participants. The task categories are: software development tasks (e.g., feature or test suites developing, or code review), maintainability (e.g., improving code/project quality), issues management tasks (e.g., reporting, triaging, or solving issues), and mentorship/knowledge sharing (e.g., for giving/asking help to develop a new feature or fix an issue), community building (e.g., motivating/recruiting developers, or promoting/directing the project), and repository management tasks.

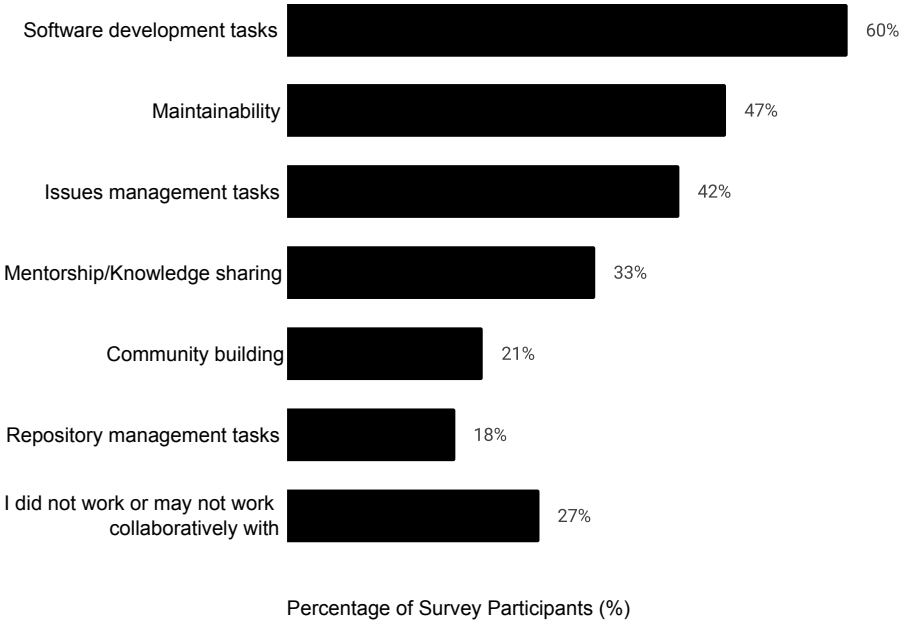


FIGURE 9 Survey results on the developers' prominent task categories to work collaboratively with other in the project. The task categories are: software development tasks, maintainability, issues management tasks, and mentorship/knowledge sharing, community building, and repository management tasks.

The majority of the developers collaborate in software development tasks (60%). Maintainability (47%), issues management tasks (42%), and mentorship/knowledge sharing (33%) are also prominent tasks to work collaboratively with other developers. Community building and repository management are the least selected tasks for collaborative work. On the other hand, 27% of them claimed that they would not work collaboratively on any task with the presented developer. Participants further had the opportunity to provide an optional comment within the survey. In this optional answer, they commented that Documentation is another category for collaborative tasks. Cross-validating these results with the interview study results (see Section 4), tasks related to software development and issues management were the most preferred. Furthermore, in both studies, the developers mentioned that the Mentorship/Knowledge Sharing and Documentation tasks are opportunities for collaboration in the project.

To investigate which task categories were more appropriate to work in a group or independently in the project, we matched these task categories with the population sample based on their preferences of working (Figure 7a). Figure 10 shows the detailed analysis for each type of task, considering two groups from the results of SRQ1: (i) participants who prefer to work strictly in the `Collaboration Group` and participants who prefer to work strictly independently (`Independent Group`), we do not consider the Mixed group for this analysis because they are openness for any opportunity of collaboration or in a group or independently. Moreover, it is important to notice that we excluded the answers of 3 participants of the collaboration group who marked all options in the questionnaire item SQ3, rendering their response inconsistent (they would not work collaboratively in any task and also would work collaboratively in all tasks). Thus, the `Collaboration Group` has 81 participants, and the `Independent Group` has 16 participants.

Starting the analysis with the `Collaboration Group` that declared themselves open to collaborative work with the project developers, the tasks most selected by them are software development tasks, maintainability, and issues management tasks with 70%, 53%, and 52%, respectively. These results match with other works[39,51,57] that also show that these tasks are preferred among the projects' developers. On the other hand, the tasks of repository management (80%), community building (77%), and mentorship/knowledge sharing (63%) were the least considered. This

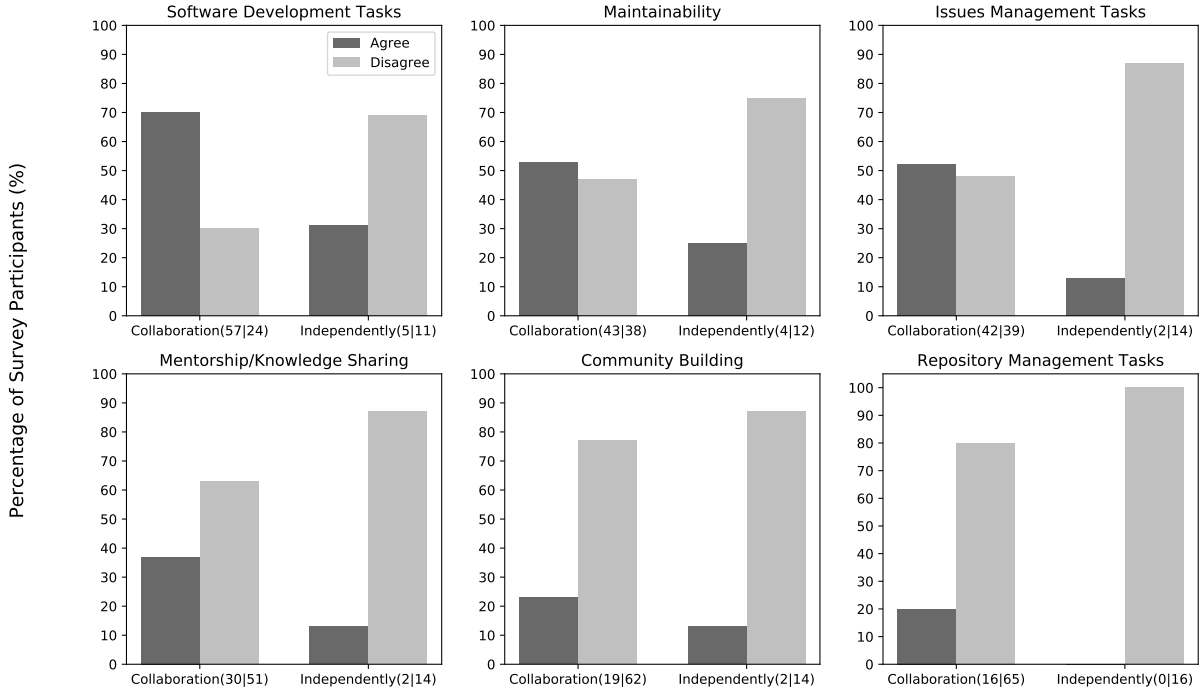


FIGURE 10 Each task category has two groups: collaboration and independent group. The number of participants for each group is in parentheses. However, they are subdivided based on their choice ("Agree" | "Disagree"). Participants who agreed to work collaboratively with another developer randomly selected ("Agree") in that task category, and participants who disagree or did not choose the task category ("Disagree").

result is somewhat surprising. We expected that this group would be more open to these tasks (community building and mentorship/knowledge sharing, mainly) to strengthen ties among developers and, consequently, the project community. Indeed, some works [73,74] encourage mentoring to support the integration of the newcomers into projects, and others [75,76] present the difficulty faced by developers to obtain effective mentoring. Cross-validating these results, in our interview study (see Section 4), few developers also chose the repository management tasks. However, some developers have emphasized that the projects should have collaborators available to support developers who need helps.

Some developers who declared a preference to work independently in the project claimed they were not open to creating partnerships with another developer. For instance, for the participant S63 the Pull Request is the event that implies that an author is ready for their code to be looked at. Before that point, any code on other forks/branches is likely to be underdeveloped, and review/input from other developers may be detrimental. For example, premature input on a vague idea can waste a disproportionate amount of time on a trivial or non-critical issue. Participant S67 completed: *"I think the distributed version control model has this right in the sense that there is a way to indicate to others that you are ready to discuss your ideas by creating PRs, either to the main repo or to any other fork if you want to have a more focused discussion with a specific individual"*. On the other hand, just a few of these developers (`Independent Group`) still consider working collaboratively. One possible reason for this could be that developers can see the benefits of their contributions to the project community. Although they prefer to have their tasks well defined to work independently, they are willing to work collaboratively on specific tasks with other developers, such as code development and maintainability. In our interview study (see Section 4), developers pointed out the motivations/reasons to work independently. However, they are open to working collaboratively if they need some help.

SRQ2 - When exposed to the project's collaborative scenario, the majority of participants selected the category related to software development (60%), maintenance (47%), issues management (42%), and mentorship/knowledge sharing (33%) as prominent tasks to work collaboratively with other developers. On the other hand, 27% of them claimed that they did not work or may not work collaboratively with a randomly selected developer. Furthermore, participants added freely Documentation as a category for collaboration.

5.3 | SRQ3 - What other perceptions do developers provide about collaborations?

In this section, we address the results for the third research question related to other surveyed participants' perceptions. To this end, we analyzed the answers from the open questions used for the survey to extracting new insights, suggestions, or criticism. Indeed, participants had this field to elaborate or clarify their points, helping us to understand information or situations from a participant's perspective as we get feedback in their own words, the most of them will be discussed in more detail as follows.

Participants mentioned their perceptions of the collaborative process and the main beneficiaries in this context of projects hosted on GitHub. Contributors copy the original version and make their changes. When their works are ready, they have an option of making back these changes to the original repository contributing to the project and the entire community or keeping them into their own fork for their personal benefits, as explained in Section 2.1 and by participant S32: *"All collaboration occurs on the original repository, and there would be little to no benefit for users to collaborate away from the main repository. So, any work off of the main repo would only serve for personal purposes."* Participant S37 also detailed how this collaborative process: *"When making changes, I code independently in my own fork of the project. Collaboration occurs when I submit a pull request into the main repository of the project and those code changes undergo review. Collaboration also occurs when issues are discussed, again in the main repository."* Contributions in the copy of contributors and without backing to the original repository also happen for their benefit initially. However, in some cases, it may generate a new project or product [77]. The opposite also applies, i.e., when a project merges with another one. Participant S31 explained *"This was a special case of a project developed independently, <project 1>, being moved into <project 2>. I worked with the maintainer (of project 2) on his fork to move the project over and then we finally moved our combined effort into <project 2>."* Participants detailed these types of contributions and their advantages as we also presented in the interview study (Section 4).

Participants expressed the importance of communication between maintainers, core team members, and other contributors focused discussions around the subject, such as new insights/ideas about an issue or new feature, software development, and code review. Participants also listed many communication channels used to interact with the project contributors, such as GitHub Issues, GitHub Pull Request, Slack tool, community meetings, and mailing list discussions. For instance, surveyed participant S42 mentioned: *"In general, on this project, we often identify potential collaborations through centralized communication, such as the assignment of* GitHub *issues on <project name> or through questions and comments on the <project name> Slack space."* Participant S45 also mentioned *"Usually, I discuss my ideas and thoughts in a designated* GitHub *Issue/PR with the core team and whoever joins in the discussion in the form of comments. Technically, anyone can join the discussion or review my code, but typically only members of the core team do."* In our interview study (Section 4), developers pointed out that the value of communication among developers and the tools to support them. They also mentioned that projects are open for anyone. However, usually, core team member become more required.

Concerning effective coordination of collaborations, participants expressed the importance of the GitHub environment to support maintainers and contributors in the making decision and development tasks of the projects. For example, participant S80 clarified: "Most collaboration happens in GitHub issues or pull requests. It's rare to work directly on a fork together unless we're prototyping a big feature." Moreover, some participants mentioned that due to large projects with too many developers (forks), it becomes difficult to keep track of their contributions or know who they are. Participant S17 declared (supported by S37): *"There are a LOT of forks of <project name> out there. While I'm very tied into the main repo they're ALL forked from, I don't spend any effort tracking other people's forks."* Participants also declared the main purpose of their fork version. They also declared that use their copy of the original repository to make contributions back, as stated by participant S121 (supported S73, S75, S96, and others): *"My fork's only purpose is to fix bugs and merge back into the main repository"*. In general, GitHub encourages the participation of anyone who wants to participate. For example, the participant declared that it is limited to contribute to the project. However, GitHub can support them: *"The general development of this project is interesting and important, but my ability to contribute is limited. That makes working in a* GitHub *setting very practical."*

SRQ3 – Participants stated their perceptions of the collaborative process. For example, collaboration occurs mainly during issue discussions and when the code changes undergo review. Besides, they also mentioned that they often identify potential collaborations through centralized communication, such as the assignment of issues, questions, or comments. Finally, they expressed the importance of the GitHub environment to support maintainers and contributors in making decisions and in the development tasks of the projects.

6 | DISCUSSIONS AND IMPLICATIONS

6.1 | Discussions

In this section, we discuss and summarize some of our main findings obtained in the interview study and the opinion survey.

Motivation to Collaborate with Specific Groups. Our research goal was to better understand how and why collaboration occurs from the perspective of developers. Therefore, we investigated the motivations they have to collaborate with others and their preferences. As a result, most developers declared their preference to work with core team members (Sections 4.3, 5.1, 5.2). One reason could be that some collaborators feel

motivated to follow the most experienced and involved developers of the projects. In addition, they can have particular interests in this partnership, for example, improving their reputation. Another possible reason can be when the project has sponsors (Section 5.1). Hence, their more active developers receive financial benefits. Consequently, they could interact more with the core team members for being exclusively dedicated to the project. Therefore, maintainers need to pay attention to providing the power that core team members have to attract and engage other collaborators to the project. Besides, maintainers need to keep everyone aware of the importance of appreciating and encouraging all contributions to the project. Last but not least, the benefits of mentoring should be considered as newcomers are interested and motivated to participate in the projects and active participation, and long-time developers can follow them through mentoring and help them do quality codes and contributions (Section 5.2). On the other hand, most of the developers who have been more open to collaborating with other groups and independently had their mismatched preferences (Section 5.1). It is not easy to work collaboratively with other groups; most of them only worked independently. Finally, developers who prefer independent tasks have the most matched expectations. It seems that punctual contributions without pressure and their own time are attractive to this developer profile.

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 (Section 4.2). 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 (Section 4.3). Indeed, some issues require the experience and knowledge of the core team. However, the core team may be overwhelmed with project direction and other responsibilities of more priorities. Properly, the project maintainer could manage these issues for active collaborators (peripheral or even newcomers that has expertise from previous projects). Besides, some collaborations that are reserved for the core team, which require administrative privilege, could be delegated to other collaborators able to help them. As future work, we want to investigate which type of collaborations the maintainers could outsource. It may help to avoid the overwork of the core team, motivate, and to engage other collaborators in the project.

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, carelessness 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. 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 (Section 4.2).

Collaboration from 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 (Section 4.3). 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.

Collaboration Opportunities based on Profile of Developers. Despite personal preferences to work independently, developers still admit the possibility of collaborating with others in some scenarios, especially in software development and maintenance tasks (Section 5.2). Similarly, developers who prefer to work collaboratively are not necessarily inclined to collaborate with others in any given project. Therefore, it is important to provide developers with meaningful recommendations for collaboration that addresses specific motivations. Furthermore, it is essential to investigate what those motivations are.

6.2 | Implications

The results of this work can have several implications for researchers, practitioners, and tool developers in the area of open-source software development.

1. Practitioners can use the findings of this work to better understand the available knowledge-sharing practices and their suitability for different contexts to apply practices to address their developer attracting and motivating challenges. For example, practitioners need to know that a lack of mentors can inhibit newcomers from participating in the project or provide them with quality code. Therefore, we recommend that practitioners pay attention to provide a receptive environment, follow and appreciate the work of developers, especially newcomers. It is essential to keep the collaborators engaged in the project to deliver better results and better quality software products.

2. This work has presented some communications means to enable collaboration in the project. In general, the tools concentrate the issues of the project, share the knowledge and collective help, support the development of tasks, and facilitate the inclusion of newcomers in

the project. Thus, we suggest that researchers, practitioners, and tool developers cooperate closely to enhance communication means to support collaborators in sharing knowledge and making partnerships to improve the collaborations in open-source development.

3. The results also showed the main tasks of the project in which developers prefer to collaborate with others or independently. Even contributors who are more open to working collaboratively with other developers have expressed their preferences for specific tasks. In contrast, developers who expressed strict preferences for independent tasks were still ready to work collaboratively with others on behalf of the project. These situations require special attention on behalf of researchers and practitioners as the preference for the tasks and interaction ways among developers, since disappointments related to developers' preferences may result in some loss of collaborations or abandonment projects.

7 | THREAT TO VALIDITY

In this section, we clarify potential threats to the study's credibility and discuss some bias that may have affected our research findings. The main threats and our respective actions to mitigate them are discussed below based on the proposed categories of Wohlin et al. [78].

Construct Validity. Construct validity reflects what extent the operational measures that are studied represent what the researchers have in mind and what is investigated according to the RQs [78]. 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. In our survey study, we were aware that the results could be affected by the quality of the questions. Then, we were careful about this threat by preparing a pilot questionnaire before making them available to the participants. Also, we reviewed the questionnaire related to their contents and format to avoid misunderstanding and provide the necessary definitions. We include the question 'Other' as one of the answer options and associated free-text fields in order to capture more clarification from participants.

Internal Validity. The internal validity is related to uncontrolled aspects that may affect the study results [78]. 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. Nonetheless, the number of interviewees is in accordance with the literature on lived experience (minimum 10 interviewees) [79] and phenomenological studies (minimum 6 interviewees) [80]. Hence, 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 collaborative software development projects. In our survey study, the target population consisted of developers in collaborative software development. A participant may answer the questionnaire without the required knowledge about the project. We addressed this concern in the protocol: we explicitly required a developer still involved with the project to fill in the questionnaire. A knowledgeable respondent should have more than four commits and their last commit within the last year.

External Validity. External validity concerns the ability to generalize the results to other environments, such as to industry practices [78]. In our interview study, we interviewed twelve Portuguese-speaker 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 developers 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 in several countries (e.g., Brazil, England, the United States, and Germany). Furthermore, they have contributed to projects with a higher number of stars, which attract developers from several places around the world. All these factors corroborate with the interviewees' expertise in global collaboration with open-source development.

Conclusion Validity. The conclusion validity concerns with issues that affect the ability to draw the correct conclusions from the study [78]. The results presented in the study are first and foremost observations and interpretations of the authors from the interview and survey studies. These results reflect our individual perceptions of practitioners, and our interpretations of their responses (interviews and surveys). 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.

8 | CONCLUSION AND FUTURE WORK

The importance of collaboration in open-source 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 collaborative software development projects, collaboration is a key factor. In this paper, we analyzed data collected through interviews and an opinion survey conducted with developers from different open-source software communities to know how collaborations happen, the barriers and challenges faced by developers. Our main findings from interview study 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 developers; 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.

In our survey study, we cross-validated the interview results to understand better how collaboration happens in software development projects based on developers' behavior. Especially how open they are to work collaboratively with others and the main tasks that increase collaboration opportunities. Our analysis revealed that most participants (85%) prefer to work collaboratively with the core team, 30% prefer to work in independent tasks, and 22% to value the collaborations with the other developers. Besides, when we asked about how they are working on the project (reality), 74% of those surveyed claimed that they work collaboratively with the core team, 36% work on independent tasks, and 11% answered that they work in collaboration with other developers. Furthermore, when exposed to the project's collaborative scenario, the majority of participants selected the category related to software development (60%), maintenance (47%), issues management (42%), and mentorship/knowledge sharing (33%) as main tasks to work collaboratively with other developers. Finally, despite personal preferences to work independently, developers still consider collaborating with others in some scenarios, especially in development tasks.

As future work, we want to identify the factors which may impact collaboration among developers. It also includes the cultural aspects and OSS ecosystem context. Hence, if we identify these factors, we can naturally calibrate them to connect the desired collaboration team. We also want to investigate whether developers feel more comfortable working with people they know before or do not mind working with "strangers." It could help us find which type of collaborations would be approved to be outsourced and which type of collaborations require a certain level of trust. Moreover, it may help to avoid the overwork of the core team, motivate, and engage other collaborators in the project. Besides, we are working on 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 collaborative software development.

9 | ACKNOWLEDGMENTS

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

References

1. Herbsleb JD, Moitra D. Global software development. *IEEE Software* 2001: 16–20.

2. Ulziit B, Warraich ZA, Gencel C, Petersen K. A conceptual framework of challenges and solutions for managing global software maintenance. *Journal of Software: Evolution and Process* 2015: 763–792.

3. Venters CC, Capilla R, Betz S, et al. Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software* 2018: 174–188.

4. Condori-Fernandez N, Lago P. Characterizing the contribution of quality requirements to software sustainability. *Journal of Systems and Software* 2018: 289–305.

5. Qureshi I, Fang Y. Socialization in open source software projects: a growth mixture modeling approach. *Organizational Research Methods* 2011: 208–238.

6. Schilling A, Laumer S, Weitzel T. 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. ; 2012: 3446–3455.

7. Rastogi A, Nagappan N. 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. ; 2016: 102–111.

8. Steinmacher I, Chaves AP, Gerosa MA. Awareness support in global software development: a systematic review based on the 3C collaboration model. In: International Conference on Collaboration and Technology (CRIWG). Springer. ; 2010: 185–201.

9. Gousios G, Zaidman A, Storey MA, Van Deursen A. Work practices and challenges in pull-based development: the integrator's perspective. In: International Conference on Software Engineering (ICSE). IEEE. ; 2015: 358–368.

10. Gousios G, Storey M, Bacchelli A. Work practices and challenges in pull-based development: the contributor's perspective. In: International Conference on Software Engineering (ICSE). IEEE. ; 2016: 285–296.

11. Capiluppi A, Lago P, Morisio M. Characteristics of open source projects. In: European Conference on Software Maintenance and Reengineering (CSMR). IEEE. ; 2003: 317–327.

12. Morrison P, Pandita R, Murphy-Hill E, McLaughlin A. Veteran developers' contributions and motivations: an open source perspective. In: IEEE. ; 2016: 171–179.

13. Yue Y, Ahmed I, Wang Y, Redmiles D. Collaboration in global software development: an investigation on research trends and evolution. In: International Conference on Global Software Engineering (ICGSE). IEEE. ; 2019: 68–69.

14. Lanubile F, Ebert C, Prikladnicki R, Vizcaíno A. Collaboration tools for global software engineering. *IEEE Software* 2010: 52–55.

15. Dabbish L, Stuart C, Tsay J, Herbsleb J. Social coding in GitHub: transparency and collaboration in an open software repository. In: ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW). ; 2012: 1277–1286.

16. McDonald N, Blincoe K, Petakovic E, Goggins S. Modeling distributed collaboration on Github. *Advances in Complex Systems* 2014: 1–24.

17. Tsay J, Dabbish L, Herbsleb J. Let's talk about it: evaluating contributions through discussion in GitHub. In: International Symposium on Foundations of Software Engineering (FSE). ; 2014: 144–154.

18. Arciniegas-Mendez M, Zagalsky A, Storey M, Hadwin AF. Using the model of regulation to understand software development collaboration practices and tool support. In: ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW). ; 2017: 1049–1065.

19. Steinmacher I, Treude C, Gerosa MA. Let me in: guidelines for the successful onboarding of newcomers to open source projects. *IEEE Software* 2018: 41–49.

20. Tymchuk Y, Mocci A, Lanza M. Collaboration in open-source projects: myth or reality?. In: International Conference on Mining Software Repositories (MSR). ; 2014: 304–307.

21. Bird C. Sociotechnical coordination and collaboration in open source software. In: International Conference on Software Maintenance (ICSM). ; 2011: 568-573.

22. Bjorn P, Bardram J, Avram G, et al. Global software development in a CSCW perspective. In: ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW). ; 2014: 301–304.

23. Mokarzel Filho R, Souza Pereira dM, Faria C, Lemos G. Collaboration tool for distributed open source verification. In: International Conference on Global Software Engineering (ICGSE). IEEE. ; 2019: 139–142.

24. Constantino K, Zhou S, Souza M, Figueiredo E, Kästner C. Understanding collaborative software development: an interview study. In: International Conference on Global Software Engineering (ICGSE). ; 2020: 55–65.

25. Schilling A. What do we know about FLOSS developers' attraction, retention, and commitment? a literature review. In: Hawaii International Conference on System Sciences (HICSS). ; 2014: 4003–4012.

26. Crowston K, Wei K, Howison J, Wiggins A. Free/Libre open-source software development: what we know and what we do not know. *ACM Computing Surveys (CSUR)* 2008: 1–35.

27. Miller KW, Voas J, Costello T. Free and open source software. *IT Professional* 2010: 14-16.

28. Laurent A. *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software*. O'Reilly Media . 2004.

29. Whitehead J. Collaboration in software engineering: a roadmap. In: IEEE. ; 2007: 214–225.

30. Riehle D, Ellenberger J, Menahem T, et al. Open collaboration within corporations using software forges. *IEEE Software* 2009: 52–58.

31. Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D. The promises and perils of mining GitHub. In: International Conference on Mining Software Repositories (MSR). ; 2014: 92–101.

32. Storey M, Singer L, Cleary B, Figueira Filho F, Zagalsky A. The (r) evolution of social media in software engineering. In: Proceedings of the on Future of Software Engineering. 2014 (pp. 100–116).

33. Zhou S, Vasilescu B, Kästner C. What the fork: a study of inefficient and efficient forking practices in social coding. In: ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE/ESEC). ACM. ; 2019: 350–361.

34. Linåker J, Munir H, Wnuk K, Mols C. Motivating the contributions: an open innovation perspective on what to share as open source software. *Journal of Systems and Software* 2018: 17–36.

35. Lakhani KR, Von Hippel E. *How open source software works: "free" user-to-user assistance*. Springer . 2004.

36. Gamalielsson J, Lundell B. Sustainability of open source software communities beyond a fork: how and why has the LibreOffice project evolved?. *Journal of Systems and Software* 2014: 128 - 145.

37. Butler B, Sproull L, Kiesler S, Kraut R. Community effort in online groups: who does the work and why. *Leadership at a Distance: Research in Technologically Supported Work* 2002: 171–194.

38. Marlow J, Dabbish L, Herbsleb J. Impression formation in online peer production: activity traces and personal profiles in GitHub. In: ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW). ; 2013: 117–128.

39. Tsay J, Dabbish L, Herbsleb J. Influence of social and technical factors for evaluating contribution in GitHub. In: International Conference on Software Engineering (ICSE). ; 2014: 356–366.

40. McDonald N, Goggins S. Performance and participation in open source software on GitHub. In: Conference on Human Factors in Computing Systems (CHI). ; 2013: 139–144.

41. Corbin J, Strauss A. *Basics of qualitative research: techniques and procedures for developing grounded theory*. SAGE Publications . 2014.

42. Creswell JW, Creswell JD. *Research design: qualitative, quantitative, and mixed methods approaches*. SAGE Publications . 2017.

43. Qiu H, Nolte A, Brown A, Serebrenik A, Vasilescu B. Going farther together: the impact of social capital on sustained participation in open source. In: International Conference on Software Engineering (ICSE). IEEE. ; 2019: 688–699.

44. Pinto G, Steinmacher I, Gerosa M. More common than you think: an in-depth study of casual contributors. In: International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE. ; 2016: 112–123.

45. Onoue S, Hata H, Matsumoto Ki. A study of the characteristics of developers' activities in GitHub. In: IEEE. ; 2013: 7–12.

46. Viggiato M, Oliveira J, Figueiredo E, Jamshidi P, Kästner C. How do code changes evolve in different platforms? a mining-based investigation. In: International Conference on Software Maintenance and Evolution (ICSME). IEEE. ; 2019: 218–222.

47. Viggiato M, Oliveira J, Figueiredo E, Jamshidi P, Kästner C. Understanding similarities and differences in software development practices across domains. In: International Conference on Global Software Engineering (ICGSE). IEEE. ; 2019: 84–94.

48. Oliveira J, Fernandes E, Vale G, Figueiredo E. Identification and prioritization of reuse opportunities with JReuse. In: International Conference on Software Reuse (ICSR). Springer. ; 2017: 184–191.

49. Oliveira J, Viggiato M, Figueiredo E. How well do you know this library? mining experts from source code analysis. In: Brazilian Symposium on Software Quality (SBQS). Association for Computing Machinery; 2019: 49–58.

50. Pham R, Singer L, Liskin O, Figueira Filho F, Schneider K. Creating a shared understanding of testing culture on a social coding site. In: International Conference on Software Engineering (ICSE). IEEE. ; 2013: 112–121.

51. Gousios G, Pinzger M, Deursen A. An exploratory study of the pull-based software development model. In: International Conference on Software Engineering (ICSE). ; 2014: 345–355.

52. Minto S, Murphy GC. Recommending emergent teams. In: International Conference on Mining Software Repositories (MSR). ; 2007: 5-13.

53. Jiang J, He J, Chen X. Coredevrec: automatic core member recommendation for contribution evaluation. *Journal of Computer Science and Technology* 2015.

54. Costa C, Figueirêdo J, Pimentel JF, Sarma A, Murta L. Recommending participants for collaborative merge sessions. *IEEE Transactions on Software Engineering* 2019.

55. Miller C, Widder DG, Kästner C, Vasilescu B. Why do people give up flossing? a study of contributor disengagement in open source. In: IFIP International Conference on Open Source Systems. Springer. ; 2019: 116–129.

56. Krüger J, Wiemann J, Fenske W, Saake G, Leich T. Do you remember this source code?. In: International Conference on Software Engineering (ICSE). Association for Computing Machinery; 2018: 764–775.

57. Steinmacher I, Pinto G, Wiese I, Gerosa M. Almost there: a study on quasi-contributors in open-source software projects. In: International Conference on Software Engineering (ICSE). IEEE. ; 2018: 256–266.

58. Von Krogh G, Haefliger S, Spaeth S, Wallin M. Carrots and rainbows: Motivation and social practice in open source software development. *MIS quarterly* 2012: 649–676.

59. Lee A, Carver JC, Bosu A. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In: International Conference on Software Engineering (ICSE). ; 2017: 187-197.

60. Lee A, Carver JC. 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). IEEE. ; 2017: 1–10.

61. Kalliamvakou E, Damian D, Blincoe K, Singer L, German D. Open source-style collaborative development practices in commercial projects using GitHub. In: International Conference on Software Engineering (ICSE). IEEE. ; 2015: 574–585.

62. Giuffrida R, Dittrich Y. A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams. *Information and Software Technology* 2015: 11–30.

63. Storey MA, Zagalsky A, Figueira Filho F, Singer L, German DM. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering* 2016: 185–204.

64. Bissyandé T, Lo D, Jiang L, Réveillere L, Klein J, Le Traon Y. Got issues? who cares about it? a large scale investigation of issue trackers from GitHub. In: International Symposium on Software Reliability Engineering (ISSRE). IEEE. ; 2013: 188–197.

65. Stol KJ, Avgeriou P, Ali Babar M. Identifying architectural patterns used in open source software: approaches and challenges. In: International Conference on Evaluation and Assessment in Software Engineering (EASE). BCS Learning & Development; 2010: 91–100.

66. Steinmacher I, Graciotto Silva MA, Gerosa MA, Redmiles DF. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 2015: 67 - 85.

67. S. Stănciulescu SS, Wąsowski A. Forked and integrated variants in an open-source firmware project. In: International Conference on Software Maintenance and Evolution (ICSME). IEEE. ; 2015: 151–160.

68. Farias V, Wiese I, Santos R. What characterizes an influencer in software ecosystems?. *IEEE Software* 2019: 42-47.

69. Blincoe K, Sheoran J, Goggins S, Petakovic E, Damian D. Understanding the popular users: following, affiliation influence and leadership on GitHub. *Information and Software Technology* 2016; 70: 30–39.

70. Cai Y, Zhu D. Reputation in an open source software community: antecedents and impacts. *Decision Support Systems* 2016: 103–112.

71. Zhou M, Mockus A. Does the initial environment impact the future of developers?. In: International Conference on Software Engineering (ICSE). ; 2011: 271–280.

72. Surian D, Liu N, Lo D, Tong H, Lim EP, Faloutsos C. Recommending people in developers' collaboration network. In: Working Conference on Reverse Engineering (WCRE). IEEE. ; 2011: 379–388.

73. Begel A, Simon B. Novice software developers, all over again. In: International Workshop on Computing Education Research (ICER). Association for Computing Machinery; 2008: 3–14.

74. Balali S, Annamalai U, Padala HS, et al. Recommending tasks to newcomers in OSS projects: how do mentors handle it?. In: International Symposium on Open Collaboration (OpenSym). ; 2020: 1–14.

75. Canfora G, Di Penta M, Oliveto R, Panichella S. Who is going to mentor newcomers in open source projects?. In: International Symposium on the Foundations of Software Engineering (FSE). Association for Computing Machinery; 2012: 1–11.

76. Steinmacher I, Conte T, Gerosa MA, Redmiles D. Social barriers faced by newcomers placing their first contribution in open source software projects. In: ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW). ; 2015: 1379–1392.

77. Jiang J, Lo D, He J, Xia X, Kochhar PS, Zhang L. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 2017: 547–578.

78. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in software engineering*. Springer . 2012.

79. Bernard HR. *Research methods in anthropology: qualitative and quantitative approaches*. Rowman & Littlefield . 2017.

80. Morse JM. Designing funded qualitative research.. 1994.

How to cite this article: K. Constantino, M. Souza, S. Zhou, E. Figueiredo, and C. Kästner (2021), Perceptions of Open-Source Software Developer on Collaborations: An Interview and Survey Study, *Journal of Software: Evolution and Process*, *2021;00:1–27*.