UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Kattiana Fernandes Constantino

Finding Collaborations based on Co-Changed Files

Belo Horizonte
2022

Kattiana Fernandes Constantino

Finding Collaborations based on Co-Changed Files

Final Version

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Eduardo Magno Lages Figueiredo

Belo Horizonte
2022

Constantino, Kattiana Fernandes.

C758f Finding collaborations based on co-changed files [recurso eletrônico] / Kattiana Fernandes Constantino – 2022.
1 recurso online (159 f. il, color.): pdf.

Orientador: Eduardo Magno Lages Figueiredo.

Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 123-138.

1. Computação – Teses. 2. Engenharia de software – Teses. 3. Sistemas abertos (Computadores) – Teses. 4. Software – Desenvolvimento – Sistemas colaborativos - Teses. I. Figueiredo, Eduardo Magno Lages. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*32(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6/1510 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Finding Collaborations based on Co-Changed Files

KATTIANA FERNANDES CONSTANTINO

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Eduardo Magno Lages Figueiredo - Orientador
Departamento de Ciência da Computação - UFMG

Profa. Raquel Oliveira Prates
Departamento de Ciência da Computação - UFMG

Prof. Marco Túlio de Oliveira Valente
Departamento de Ciência da Computação - UFMG

Prof. Igor Fabio Steinmacher
Campo Mourão - UTFPR

Prof. Marcelo de Almeida Maia
Faculdade de Computação - UFU

Belo Horizonte, 15 de Julho de 2022.

Este trabalho é fruto de inúmeros e árduos sacrifícios em tempos difíceis. Tempos em que a ciência e a pesquisa foram fortemente atacadas no Brasil. Portanto, dedico esse meu trabalho a todos os cientistas, pesquisadores e estudantes que bravamente resistiram e ainda, resistem. Por uma ciência de qualidade e mais igualitária.

Acknowledgments

Em primeiro lugar, quero agradecer a Deus por tudo. Sou profundamente grata a Ele, por me guiar e me dar forças para superar os obstáculos na minha vida e principalmente, nessa etapa do doutorado. Também, agradeço sinceramente a toda a minha enorme família. Em especial, agradeço aos meus pais Edgar e Cícera, minha irmã Kellen, meus irmãos Kássio e Kelbe, meu cunhado Moacir, minha cunhada Daliane, minhas sobrinhas Bruna e Fernanda e meu sobrinho Miguel. Agradeço pelo amor, orações, apoio e torcida. Agradeço por acreditarem em mim sempre. Por se fazerem presentes mesmo que distantes. Obrigada por tudo. Amo todos vocês. Além disso, agradeço aos amigos que a vida me deu, que foram e são essenciais na minha jornada, principalmente nos momentos mais adversos. Sou feliz por ter cada um de vocês perto de mim.

Também agradeço por todo apoio, orientação e incentivo que meu orientador Eduardo Figueiredo me deu durante o doutorado. Sua experiência e conhecimento me desafiaram a pensar criticamente e me levaram a atingir meu potencial máximo. Além disso, sua paciência e disposição para responder às minhas perguntas foram fundamentais para moldar minha pesquisa e melhorar minhas habilidades de escrita. Eu não poderia ter concluído esta jornada sem a sua ajuda. Também, quero expressar minha mais sincera gratidão a cada um dos meus amigos da DCC e do LabSoft que, ao longo de todos esses anos, caminharam juntos cada passo dessa longa, árdua e vitoriosa jornada. Sem vocês, meus queridos amigos, teria sido muito mais difícil. Segundo um provérbio africano, *"Se quer ir rápido, vá sozinho. Se você quer ir longe, vá em grupo."*

Agradeço também o inestimável apoio e trabalho dos professores do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais (DCC/UFMG). O compromisso do corpo docente em proporcionar uma educação de qualidade aos seus alunos tem sido notável, me sinto privilegiada! Também, estendo minha gratidão a todos os colaboradores que trabalham incansavelmente para garantir o bom funcionamento de todas as atividades desta excelente universidade. Além disso, gostaria de agradecer às agências de financiamentos por todas as bolsas recebidas ao longo desta jornada: (1) Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) bolsa 424340/2016-0; (2) Programa de Doutorado Sanduíche no Exterior (PDSE) da CAPES, bolsa 88881.189537/2018-01 - Sob a supervisão do professor Christian Kästner (agradeço pela orientação durante o intercâmbio); e, (3) Ministério Público do Estado de Minas Gerais (MPMG) pelo Programa de Capacidades Analíticas - sob a supervisão dos professores Marcos André Gonçalves e Alberto Laender (obrigada pelo apoio e a oportunidade).

"*Alone we can do so little, together we can do so much.*"
(Helen Keller)

Resumo

Desenvolvedores devem colaborar entre si em todos os estágios do ciclo de vida do software para criar sistemas bem-sucedidos. No entanto, em grandes projetos com centenas de desenvolvedores, como os projetos de código aberto, pode ser muito complexo encontrar desenvolvedores com a mesma afinidade e, assim, obter boas colaborações e novos *insights*. Além disso, nesse contexto de projetos, pode haver desperdício de recursos e esforços, o que desencoraja a permanência de muitos desenvolvedores. Portanto, esta tese de doutorado propõe uma investigação sobre o desenvolvimento colaborativo baseadas em interesses similares de código. Foram realizados cinco estudos empíricos, nos quais: (1) investigamos como as colaborações acontecem em desenvolvimento de projetos de código aberto. Algumas das principais conclusões do estudo da entrevista incluem que a colaboração transcende a codificação e inclui tarefas de documentação e de gerenciamento; (2) investigamos quão abertos os desenvolvedores estão para colaborarem uns para com os outros. Algumas análises do estudo de pesquisa revelaram que a maioria dos participantes (85%) prefere trabalhar em colaboração com a equipe principal e 30% prefere trabalhar em tarefas independentes; (3) propomos duas estratégias de recomendação de desenvolvedores e um protótipo para suportá-las; (4) avaliamos as duas estratégias de recomendação de desenvolvedores, sob o ponto de vista de quem recebe a recomendação; as taxas de aceitação para eles foram superiores a 65%. Quando juntamos as estratégias, a taxa de aceitação foi de 81%. Finalmente, (5) também avaliamos o suporte ferramental com usuários e não usuários do GitHub. Baseado nos resultados obtidos nesta tese de doutorado, é possível que os desenvolvedores e mantenedores possam adquirir o conhecimento para fomentar as colaborações no projeto e consequentemente, evitar o esvaziamento do mesmo.

Palavras-chave: projetos de software de código aberto; desenvolvimento colaborativo de software; colaboração distribuída; recomendação de desenvolvedor.

Abstract

Software developers must collaborate at all stages of the software life-cycle to create successful software systems. However, for large projects with hundreds of dynamic developers, such as several successful open source projects, it can be very complex to find developers with the same affinity and thus gain suitable collaborations and new insights. Besides, in this project context, resources and efforts may be wasted, discouraging many developers from staying. Therefore, this doctoral thesis proposes an investigation of collaborative development based on similar code interests and tool–supported strategies to help developers find suitable collaborators. We performed five empirical studies: (1) we investigated how collaborations happen in open–source software development projects through an interview study. Some main findings from the interview study include that collaboration transcends coding and includes documentation and management tasks; (2) we designed and performed a survey study to investigate how open developers are to collaborate with others. Some analysis from the survey study revealed that most participants (85%) prefer to work collaboratively with the core team members and 30% prefer to work in independent tasks; (3) we provided two strategies based on co–changed files and a prototype tool, named CoopFinder, that support them; (4) we evaluated these two strategies to motivate collaborations based on changes of similar code of point of view of who receives the developer recommendations. As a result, the acceptance rates for them were greater than 65%. The joint strategy presented the best acceptance rate (81%); and, (5) we also evaluated these strategies and their supporting tool with GitHub users and non–GitHub users. About 86% of the participants answered that they could use or recommend this tool. Based on the results obtained in this doctoral thesis, it is possible that developers and maintainers can acquire the knowledge to foster collaborations in the project and, consequently, avoid emptying it.

Keywords: open-source software projects; collaborative software development; distributed collaboration; developer recommendation.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

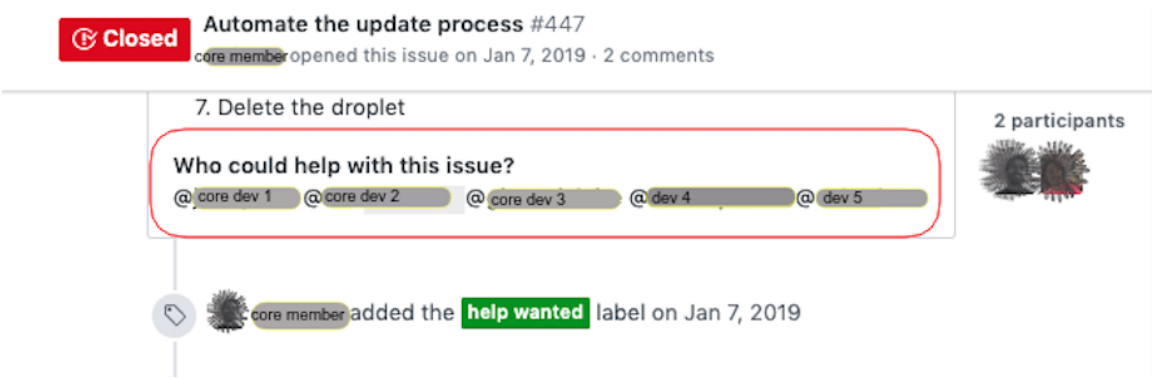
Consider two hypothetical scenarios. First, Mary is a core team member in an open-source software (OSS) project. She would like more contributors to develop new features, enhance, and maybe help manage the project. Mary also knows that many developers made the last contribution long ago or never contributed to the project again. Thus, she decided to promote an event to encourage the engagement of these temporarily inactive developers or attract new developers. Moreover, Mary realizes that it would be interesting for the project if active developers motivate others to contribute again or make their first contributions. Thus, the chances of engagement and assertive contributions would be more significant.

In the second hypothetical scenario, Joseph is a young developer and a volunteer in an OSS project hosted on `GitHub`. He has tried to make a few contributions to a specific project. For example, he was recently asked to design a new feature for this project. However, Joseph is not very familiar with this specific project. Thus, he needs some help. Perhaps, he could find another developer with whom he could discuss various design ideas to have new insights. Therefore, Mary and Joseph look for the solution to their problems. In other words, they want to find other developers with the same interests in the project. That is, developers prefer or are familiar with specific parts of the code, being able to make contributions regarding these parts. Consequently, they contribute to the engagement in the project as a whole and enhance the opportunities for collaborations.

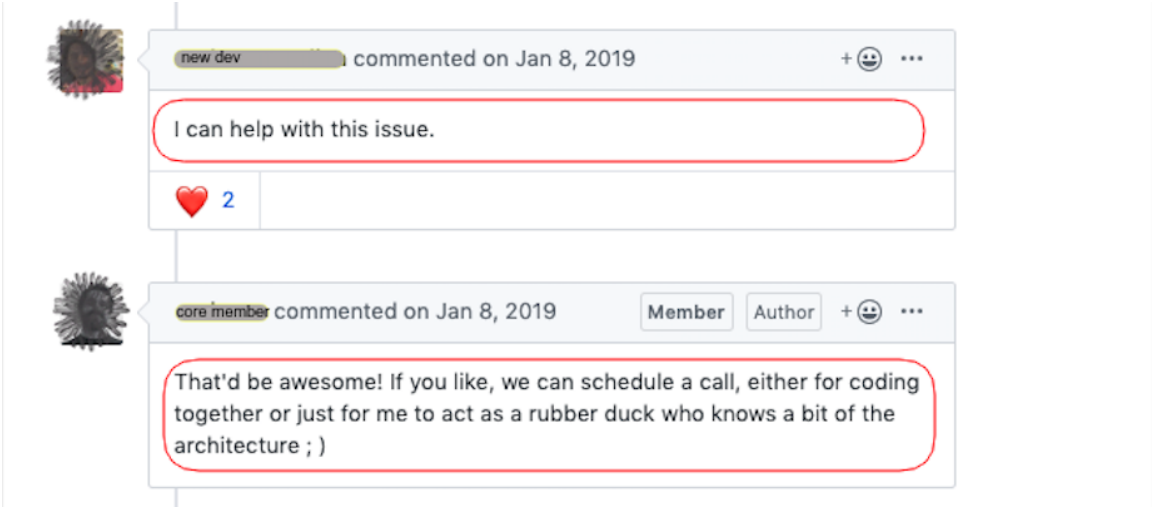
Although Mary and Joseph are hypothetical cases, Figure 1.1a shows a concrete example of a GitHub project in which a core member called five other developers (three core and two casual developers) to help him with an issue[1]. The post author probably thought that these developers' work would be relevant to this issue; thus, the author mentioned (@)<developer> to join in the discussion. However, for some reason, none of them answered the request. Hence, this real example leads us to think about one of our general questions: *although they are members of the project, would they be the most appropriate and interested developers to help the post author?*

Figure 1.1b presents a second part of the same example. After one day, another developer, different from the five called ones, offered to help. Afterward, the issue author

[1]https://github.com/okfn-brasil/serenata-de-amor/issues/447



(a) Core member called other core developers to help with this issue.



(b) A new developer offered help with this issue. And, the core member suggested that they work collaboratively.

Figure 1.1: Discussion on GitHub forum about an issue.

offered to code together or to help this developer as a mentor. By observing this second situation, we could wonder: *since the core members are overloaded, what other developers could be called upon to work together?* After a few days, that issue was closed. Despite the enthusiasm of the issue author to help in what the new developer needed, there is no evidence that the collaboration happened. There was no record of commits on the new developer fork. Moreover, there is no evidence that any other developers helped the core member solve this project's issue.

1.1 Problem Statement

Previous work showed that developers usually prefer to request collaboration from core team members, who are supposed to have sufficient motivation, knowledge, and

experience in the project [Minto and Murphy, 2007; Kononenko et al., 2016]. However, based on other prior studies, core team members may be overwhelmed and, as a result, they may not provide collaborative support promptly [Yu et al., 2015; Gousios et al., 2015; Steinmacher et al., 2018]. Moreover, other experienced developers, who are not part of the core team, could be better used by the project. In other words, all collaboration is essential for the sustainability of the project [Gamalielsson and Lundell, 2014]. Hence, all contributions should be valued and encouraged [Pham et al., 2013; Gousios et al., 2014; Pinto et al., 2016].

Previous work also mentioned that the lack of people performing some roles that compose the core team, such as maintainers, supporters, reviewers, and others, impacts the sustainability of the project [Jiang et al., 2015; Costa et al., 2021]. Another impact on the project is related to developer turnover. For instance, a small group of developers may be overloaded and centered on the project information and knowledge [Avelino et al., 2016; Ferreira et al., 2017]. Moreover, other developers may be underused, even scarce, or with restricted access to information due to limited knowledge-sharing opportunities (e.g., collaborations, discussions) [Tamburri et al., 2015]. Both situations can frustrate the developers, encouraging them to leave the project. All of the issues raised above are on how the community of developers relates to each other. Moreover, how these relationships positively or negatively impact the project. These situations also lead us to question: how to balance and optimize collaboration among project developers?

1.2 Research Goals

This doctoral thesis aims to support developers, maintainers and researchers with a better understanding of how to improve collaboration opportunities among developers in a specific project and, consequently, avoid project starvation. Thus, the general objective can be divided into the following specific goals (SGs) as follows.

- **SG1** Investigate the motivations, processes, interactions, and barriers involved in collaboration during open–source software development.
- **SG2** Investigate how open the developers are for collaboration with others.
- **SG3** Provide tool–supported strategies based on co–changed files to find suitable collaborators.
- **SG4** Evaluate developers recommendations based on co–change files from the point of view of who receives the recommendations.

- **SG5** Evaluate the effectiveness of developer recommendation tools in supporting developers and maintainers, considering both perspectives (GitHub user and non–user).

1.3 Method

We divided this doctoral thesis into five main steps described in Figure 1.2. Therefore, this research begins with an interview study (Step 1). Afterward, we designed and applied a survey study (Step 2) to investigate if developers are open to collaborations. Following this, we designed and implemented tool–supported strategies of developer recommendation based on similar interests (Step3). Next, we designed and applied a survey study (Step 4) to evaluate the developer recommendations. Finally, we performed a controlled experiment (Step 5) to complete the evaluation.

Step 1. As shown in Figure 1.2, 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. We focused on understanding (i) what motivates developers to collaborate, (ii) the collaboration process adopted, and (iii) challenges and barriers involved in collaboration (Chapter 3).

Step 2. We performed a survey study (Figure 1.2) to cross-validate the interview results (Chapter 3). We aimed to know how open developers work collaboratively based on their behaviors and to identify and check the main tasks to explore further collaboration opportunities (Chapter 4).

Step 3. As detailed in Figure 1.2, based on the lessons learned from the previous steps, we designed and proposed two strategies of developer recommendation based on co-changed files. To extract these files, we considered the number of commits for Strategy 1. For Strategy 2, we used the number of changed lines of code. Furthermore, we designed and implemented a visual tool to support these strategies (Chapter 5).

Step 4. We performed a survey study to evaluate two developer recommendation strategies based on co-change files from the point of view who receives the recommendations (Figure 1.2). These recommendation strategies were results of our interview study (Chapter 3) and the survey study (Chapter 4).

Step 5. We performed a controlled experimental study to evaluate two recommendation strategies and the proposed visual tool (Figure 1.2).

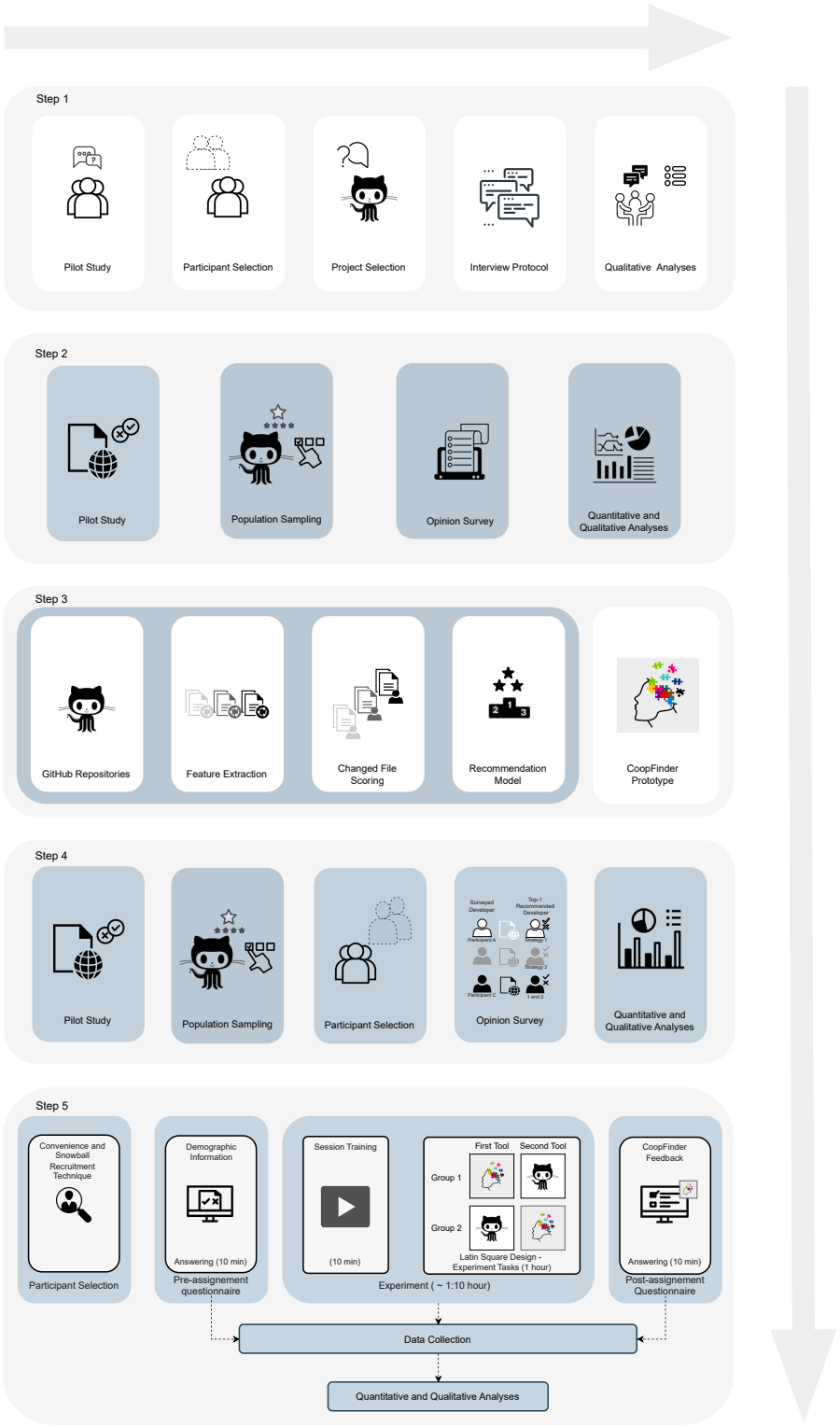


Figure 1.2: Study steps.

1.4 Contributions and Publications

One of the main expected contributions of this doctoral thesis is the lessons learned concerning collaboration in open–source software development. With our results, we believe that practitioners acquire some necessary knowledge to improve the collaborations among developers and to avoid starvation in the project. The second main expected contribution is the visual framework to help developers improve collaboration opportunities in a open–source software development project. The recommendations are extracted from the software development activities among developers of the same project.

Until the date of production of this document, the following publications were by products of this doctoral thesis, and contain parts of the thesis results.

1. *Understanding Collaborative Software Development: An Interview Study.* 15TH IEEE/ACM International Conference on Global Software Engineering (ICGSE), Seoul, South Korea, 2020. [Constantino et al., 2020].

2. *Perceptions of Open-Source Software Developers on Collaborations: An Interview and Survey Study.* 2021. Journal of Software: Evolution and Process (JSEP), page e2393. [Constantino et al., 2021].

3. *CoopFinder: Finding Collaborators Based on Co–Changed Files.* 2022. IEEE Symposium on Visual Languages and Human–Centric Computing (VL/HCC), Rome, Italy, 2022. [Constantino and Figueiredo, 2022].

4. *Dual Analysis for Helping Developers to Find Collaborators Based on Co–Changed Files: An Empirical Study.* 2023. Software: Practice and Experience (SPE). doi: 10.1002/spe.3194. [Constantino et al., 2023a].

5. *Recommending Collaborators Based on Co–Changed Files: A Controlled Experiment.* 2023. XVIII Simpósio Brasileiro de Sistemas Colaborativos (SBSC). [Constantino et al., 2023b].

6. *Finding Collaborations based on Co-Changed Files.* 2023. XVIII Simpósio Brasileiro de Sistemas Colaborativos (SBSC). [Constantino and Figueiredo, 2023].

Our work (1) has been recognized with an honorable mention at the prestigious ICGSE/2020 conference. This conference is renowned worldwide for its focus on software engineering processes and globally distributed software development. In recognition of the quality of our work (1), we were invited by ICGSE/2020 to contribute with work (2) to a special issue in the Journal of Software: Evolution and Process (*Impact factor (2021):1.864*). Furthermore, our work (3) has been accepted for presentation at the IEEE

Symposium on Visual Languages and Human-Centric Computing, widely recognized as the premier international forum for research on this topic. Another significant accomplishment is the publication of our work (4) in the journal of Software: Practice and Experience (*Impact factor (2021):3.200*), which is highly respected for its contributions to the practical application of software techniques and tools for both software systems and applications. We are thrilled to announce that our latest work (5) was not only presented at SBSC (2023) but also received the prestigious Best Paper Award, recognizing its exceptional contribution to the field. Additionally, this current work received an honorable mention at the SBSC Doctoral and Master Theses Competition, further acknowledging its merit.

Furthermore, our work provided us with the opportunity to visit the Institute of Software Research (ISR) at Carnegie Mellon University (CMU) in Pennsylvania, United States from October 2018 to March 2019. During this period, we had the privilege of being supervised by Professor Christian Kästner.

Moreover, we also contributed to the following work during this Ph.D. research:

1. Maurício R A Souza, Kattiana Constantino, Lucas Veado, Eduardo Figueiredo. Gamification in Software Engineering Education: An Empirical Study. 30th International Conference on Software Engineering Education and Training (CSEE&T). Savannah, GA, USA, 2017. DOI: 10.1109/CSEET.2017.51.

2. A. Silva, G. Carneiro, M. Monteiro, F. Abreu, K. Constantino, and E. Figueiredo. *On the Impact of Product Quality Attributes on open–source Project Evolution.* In Proceedings of the 14th International Conference on Information Technology: New Generations (ITNG), Las Vegas, USA, 2018.

3. A. Silva, K. Constantino, G. Carneiro, A. Paula, E. Figueiredo, M. Monteiro, and F. Abreu. *The Influence of Software Product Quality Attributes on open–source Projects: A Characterization Study.* In Proceedings of the International Conference on Enterprise Information Systems (ICEIS), Setúbal, Portugal, 2017.

4. Juliana Pereira, Kattiana Constantino, Eduardo Figueiredo, and Gunter Saake. Quantitative and Qualitative Empirical Analysis of Three Feature Modeling Tools. In Communications in Computer and Information Science (CCIS), 2017.

1.5 Results

This doctoral thesis provides the following main results.

Our main findings from the interview study (Chapter 3) 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 interacting with peripheral developers; and, (iii) knowledge management is a challenge in collaboration, and it is important to carefully define communication policies to mitigate and avoid problems related to knowledge retention and decentralization.

Our analysis from the survey study (Chapter 4) revealed that most participants (85%) prefer to work collaboratively with the core team, 30% prefer to work in independent tasks, and 22% value the collaborations with the other developers. Furthermore, the majority of participants selected the category related to software development (60%) as the main task 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.

In Chapter 5, we provided and discussed two strategies for recommending developers based on co–changed files. First, we recommend developers to engage in the project and enhance the opportunities for collaborations, not only core members, code-reviewers, or mentors but also all developers commonly interested. We also presented the CoopFinder tool to help developers find collaborators in a specific project. CoopFinder supports both strategies mentioned.

Our main findings from the survey study (Chapter 6) to evaluate two recommendation strategies showed that (i) developers have a similar interest in the co-change files for both strategies. These considerations are of relevance because many opportunities for contributions to the project are linked with coding. Thus, that may indicate one less barrier to improving collaboration among developers. (ii) The acceptance rates for the two recommendation strategies were greater than 65%. The joint strategies presented the best acceptance rate (81%), which raises evidence of the benefits of combining both Strategies 1 and 2.

Finally, as a result of the experimental study (Chapter 7), participants pointed out that CoopFinder is easy to use, intuitive, exciting, and supports project maintainer. Moreover, about 66% of the participants stated they would use or recommend this tool.

1.6 Doctoral Thesis Outline

In addition to this introductory chapter, the remainder of this thesis is organized as follows.

Chapter 2 provides the essential concepts to support this work. In addition, we present an overview of collaboration, processes, communication, and barriers and challenges faced by developers. This chapter also discusses related work.

Chapter 3 presents the interview study aiming to understand the motivations, processes, interactions, and barriers involved in collaboration during software development. We interviewed 12 experienced developers of open–source software projects hosted on GitHub. The results of this study contribute to the specific goal SG1 of this doctoral thesis.

Chapter 4 presents the survey study aiming to obtain a understanding of how collaboration happens in open–source software development, based on developers' behavior to work with others. To this end, we designed and performed an opinion survey answered by 121 developers. The results of this study contribute to the specific goal SG2 of this doctoral thesis.

Chapter 5 presents two recommendation strategies of collaborators based on coding activities, especially in co–changed files. Besides, we present the prototype tool, namely CoopFinder, to support the recommendation strategies. The results of this study contribute to the specific goal SG3 of this doctoral thesis.

Chapter 6 presents a survey study to evaluate two collaborator recommendation strategies based on co-change files from the perspective of who receives the recommendations. We evaluated these strategies based on an extensive survey with 102 real-world developers for this evaluation. The results of this study contribute to the specific goal SG4 of this doctoral thesis.

Chapter 7 presents a experimental study to evaluate the recommendation strategies and tool-prototype from the point of view of GitHub users and non-GitHub users. We present a quantitative and qualitative evaluation of it using the state of the practice as a baseline. The results of this study contribute to the specific goal SG5 of this doctoral thesis.

Chapter 8 shows a summary of this doctoral thesis.

1.7 Funding

This research was supported for the following scholarships:

- **CAPES Scholarship.** This research was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil* (CAPES) grant 424340/2016-0.
- **PDSE Scholarship.** The author's visit to Institute of Software Research (ISR) of Carnegie Mellon University (CMU), Pennsylvania/United State, from October/2018 to March/2019 under the supervision of professor Christian Kästner. This exchange was supported by the *Programa de Doutorado Sanduíche no Exterior* (PDSE) from CAPES grant 88881.189537/2018-01.
- **MPMG Scholarship.** This research was partially supported by the *Ministério Público do Estado de Minas Gerais (MPMG)* through the project Analytical Capabilities.

Chapter 2

Background and Related Work

Open–source software (OSS) is a notable model for open collaboration, which happens when people are trying to make something together to achieve the same goal [Laurent, 2004; Whitehead, 2007; Riehle et al., 2009]. 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 [Tymchuk et al., 2014; Bird, 2011]. Indeed, software developers 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 (e.g., source code, models, documentation, and test scenarios), managing and coordinating their development work or team, and communicating with each other [Whitehead, 2007]. This chapter presents a summary of background information and related work that is fundamental to the understanding of this doctoral thesis. The remainder of this chapter is organized as follows. Section 2.1 presents the main concepts on collaboration in open-source projects. Section 2.2 describes a overview of the traditional concepts of recommender systems. Section 2.3 presents the fundamental concepts of algorithms used in this doctoral thesis. Finally, we discuss the related work (Section 2.4) and conclude this chapter in Section 2.5.

2.1 Collaborative Software Development

Project Sustainability. Private and public companies want to guarantee stable and longevity software systems for their costumers. To this end, companies create a sustainable culture to support software development. Following work investigated the various entities that compose this sustainable culture. For instance, the license type can positively or negatively impact the project community [Engelfriet, 2009], the different governance approach applied in open–source software development [Eckert et al., 2019], the communication and coordination mechanisms that need constant improvement [Cataldo et al., 2006; Bird, 2011; Pham et al., 2013]. Moreover, the member diversity in the project [Lima et al., 2014; Vasilescu et al., 2015b; Qiu et al., 2019], and their sustained commitments [Dabbish et al., 2012a,b; Marlow et al., 2013] influence the project productivity. In this doctoral thesis context, we focus on project sustainability related to its developers, interaction, and individual responsibilities. Project sustainability depends on the diversity and engagement of its contributors. The more diverse a project is, the more chance the project is to succeed [Mockus et al., 2000; Qiu et al., 2019]. Furthermore, contributors who enjoy the work are more likely to remain than those driven by personal interest [Shah, 2006; Lee et al., 2017]. Indeed, contributors who identify with the community and feel like a part of it are more likely to be long-term contributors [Hippel and Krogh, 2003; Fang and Neufeld, 2009].

Developer Roles. Team can be constituted by the developers to achieve a common goal. Some teams have factors that may affect either performance or contributions to the project, as follows. Nakakoji et al. [2002] identified potential roles presented in open–source communities based on the level of involvement with the project (time and relevant contributions): project leader who is project owner, core members who are trusted members of community, active developers, peripheral developers who contribute casually, bug fixers, bug reporters, readers who discuss about code changes and leave comments, and passive users. Not all communities have all types or the same names of these roles [Nakakoji et al., 2002; Von Krogh et al., 2003]. For instance, Matragkas et al. [2014] investigated the diversity and structure of open–source software communities. They found three types of contributors: core developers, active users, and passive users. Yamashita et al. [2015] identified the core and non–core developers as the kind of contributors in open–source software development teams. Our work considers four type of contributors: newcomer, one-time contributor, peripheral contributor, and core contributor. The newcomer is a novice in the project without any contribution accepted yet [Steinmacher et al., 2014, 2018; Rehman et al., 2022]. The one-time contributor (OTC) who has one contribution accepted [Lee and Carver, 2017; Lee et al., 2017]. The peripheral contributor is a casual contributor [Lee and Carver, 2017; Barcomb et al., 2018], and finally, the core

contributor who is highly involved and usually contributes 80 percent of the source code [Barcomb et al., 2018].

Collaboration Challenges. Collaboration brings known challenges. For instance, the turnover of contributors is one of the factors that can impact the sustainability of the projects [Gamalielsson and Lundell, 2014; Lin et al., 2017] or their quality [Foucault et al., 2015]. Therefore, maintainers need to be aware of the need to create a receptive culture for everyone interested in joining the team and value their contributions [Gousios et al., 2015, 2016]. Furthermore, it is necessary to offer appropriate support for contributors to make quality contributions. This individual experience can be one of the keys to retain the contributor for the short or long-term in the project [Zhou and Mockus, 2012, 2014; Qiu et al., 2019; Barcomb et al., 2019]. Other challenges are the coordination and communication that often break down in large and distributed teams and result in longer resolution times and build failures [Herbsleb, 2007; Cataldo et al., 2008; Cataldo and Herbsleb, 2012; Vale et al., 2021]. Collaborative Development Environments integrate source code management tools and bug trackers with collaborative development features [Lanubile et al., 2010]. They can be a solution to the communication and coordination challenges of distributed software projects [Abbattista et al., 2008]. Besides, various social coding platforms allow developers to make contributions flexibly and efficiently, such as GitHub[1], GitLab[2], Bitbucket[3], SourceForge[4], and others. In our context, we focus on one social coding platform, GitHub, to improve open–source software development collaboration.

GitHub is an example of a collaborative development environment that provides an infrastructure for collaborative development. Besides, GitHub is a Web-based code-hosting service that uses the Git distributed version control system. GitHub is available for free. As of July 2022, GitHub reports having over 83 million users and more than 200 million repositories[5]. In fact, GitHub has become an essential tool in technology areas that demand collaboration, such as globally distributed software development [McDonald and Goggins, 2013; Storey et al., 2014]. GitHub also provides two collaborative development models: shared access and fork & pull model. In the shared repository model, team members have direct push access to the principal repository. This model is considered suited to small teams and organizations. The fork & pull model lower the "barrier for entry" for users interested in collaborating to an open–source software project [Gousios et al., 2015]. In this model, developers "fork" the repository to create their own copy of the source code and make changes without affecting the upstream development. Afterwards, developers can submit a pull request to inform the project maintainers to integrate

[1]https://github.com/
[2]https://about.gitlab.com/
[3]https://bitbucket.org/
[4]http://sourceforge.net
[5]https://github.com/

the changes into the main branch of the project. Pull requests are often used to initiate a code review or discussion around code changes. The fork & pull model is popular with OSS projects for reducing coordination requirements.

2.2 Recommender Systems

Recommender Systems (RecSys) usually make use of the approaches Content-Based Filtering (CB) (also known as the personality-based approach), Collaborative Filtering (CF), and Hybrid Filtering [Adomavicius and Tuzhilin, 2005; Bobadilla et al., 2013; Robillard et al., 2009; Beel et al., 2016] as well as other systems, such as knowledge-based systems, as shown by Figure 2.1. While content-based tries to recommend based on the characteristics of consumed items, collaborative tries to correlate users with shared preferences to generate the recommendations [Schafer et al., 2007]. In turn, hybrid methods combine both approaches to minimize the limitations and to improve recommendation performance [Jung et al., 2004; Su et al., 2007; De Campos et al., 2010]. We present more detail of each approach as follows.

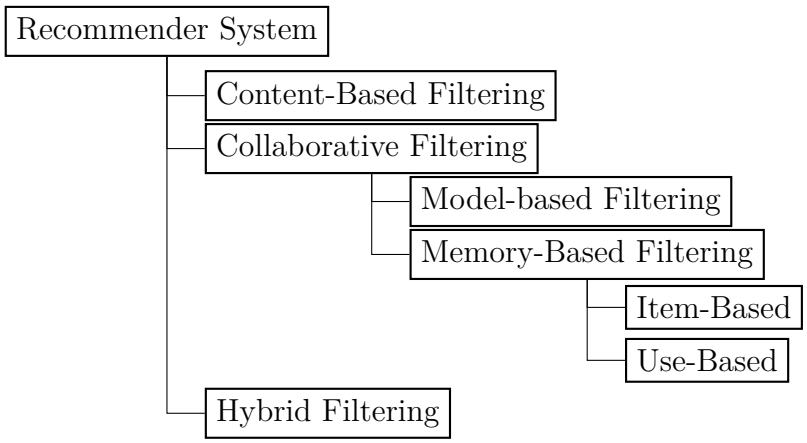


Figure 2.1: Summary of several existing recommendation strategies.

Content-Based Filtering has as basic idea using the properties of an item to recommend other similar items/people based on user preference in the past [Pazzani, 1999] (Figure 2.1). For instance, in a music context, one may use the singer's name, compositor, music genre, and keywords. These properties match the taste of a target user. In this context, only the kinds of music that have a high degree of similarity to whatever the user preferences are recommended. The content-based approach to recommendation has its origins in information retrieval research [Allenby and Rossi, 1998; Billsus et al., 1998] and information filtering [Ricci et al., 2011].

Collaborative Filtering [Breese et al., 1998] is the most popular recommendation approach. The key idea of collaborative filtering is to use the feedback from each individual user. In this approach, the user feedback may be distinguished between explicit feedback (e.g., the user assigns a rating to an item) and implicit feedback (e.g., the user purchases an item, watches a video, or contacts someone). Collaborative filtering has two groups [Yang et al., 2014], as presented by Figure 2.1: *Model-Based Filtering* applies RecSys information to create a model that generates recommendations; *Memory-Based Filtering* usually uses similarity metrics to obtain the distance between two users, or two items, based on each of their ratios. Furthermore, this group is categorized according to the item and user as follows.

Item-Based Method [Sarwar et al., 2001] is a basic collaborative filtering algorithm (Figure 2.1). The item-based method can make a recommendation by following the steps: (1) After using items (e.g., movies, music, project repositories, source code), users explicitly assign numeric ratings to the items; (2) A recommender system correlates the ratings in order to determine which item ratings are most similar to other items; (3) The system predicts ratings of new items for the target user based on the ratings of similar items already rated by the users, and (4) If these new items seem to be preferred, the system recommends them to the user. The famous sentence of Amazon.com recommendations, "Customers who bought this book also bought ...", represents the core idea of the item-based method.

User-Based Method [Resnick et al., 1994] is a basic collaborative filtering algorithm (Figure 2.1). The user-based method makes recommendations by executing (2') and (3') instead of (2) and (3) in the item-based method, respectively: (2') A recommender system correlates the ratings in order to determine which user's ratings are the most similar to other ones, and (3') The system predicts ratings of new items for the target user based on the ratings of similar users.

Hybrid Filtering [Burke, 2002; Porcel et al., 2012] seeks to combine the strengths of collaborative filtering and content-based filtering to create a system that can best meet users needs. There are different ways to combine collaborative and content-based methods. Based on the combination, this approach can be classified as follows [Adomavicius and Tuzhilin, 2005; Robillard et al., 2009]: (a) implementing collaborative and content-based methods separately and combining their predictions [Billsus and Pazzani, 2000; Kim et al., 2011], (b) incorporating some content-based characteristics into a collaborative approach [Melville et al., 2002; Li and Kim, 2003; Hu and Pu, 2010], (c) incorporating some collaborative characteristics into a content-based approach [Popescul et al., 2001; De Campos et al., 2010; Choi et al., 2012], and (d) constructing a general unifying model that incorporates both content-based and collaborative characteristics [Mooney and Roy, 2000].

2.3 Recommendation Algorithms

Information Retrieval [Salton and Buckley, 1988; Salton, 1989] and Information Filtering are the basis of the content-based approach to the recommendation. Because of the significant progress made by the information retrieval and filtering communities and because of the importance of several text-based applications, many current content-based systems focus on recommending items containing textual information, such as documents, Web sites (URLs), and news messages. The advancement over the traditional information retrieval approaches comes from the use of user profiles that contain information about users' preferences, needs, and tastes. The profiling information can be extracted from users directly or indirectly, e.g., through questionnaires or learned from their behavior over time, respectively.

Term Frequency - Inverse Document Frequency (TF–IDF) [Yu and Salton, 1976; Robertson and Jones, 1976; Salton and Buckley, 1988; Salton, 1989; Amati and Van Rijsbergen, 2002; Jones, 2004] is one of the most traditional measures for specifying term weights for information retrieval research. It is the product of two statistics: term frequency and inverse document frequency. The measure determines the importance of a term within a document that is defined as follows:

$$w_{t,d} = tf_{t,d} \cdot idf_t \tag{2.1}$$

Where, $tf_{t,d}$ is term frequency of term t in document d (a local parameter) and idf_t is inverse document frequency (a global parameter).

Term Frequency (TF) measures how frequently a term appears in a document. Since every document is different in length, a term can occur much more times in long documents than shorter ones. Thus, as a way of normalization, the term frequency is often divided by the document length:

$$tf_{t,d} = \frac{f_{t,d}}{\sum_k f_{t,d}} \tag{2.2}$$

Inverse Document Frequency (IDF) measures how important a term is. All terms are equally important while computing TF. However, specific terms, such as "is", "of", and "that", may appear many times but have little importance. Thus, we need to weigh down the many terms while scaling up the rare ones, by computing the following:

$$idf_t = \log \frac{N}{n_t} \tag{2.3}$$

Where, the log of the number (N) of sets of terms divided by the number of sets that contain the term t. Inverse document frequency $(1/n_t)$ determines the weight of rare term across all sets in the documents.

Cosine Similarity [Salton, 1971; Salton and Harman, 2003] is a measure of similarity between two non zero vectors of n dimensions by finding the cosine of the angle between them. Given two vectors of attributes, A and B, the cosine similarity, θ is represented using a cross product as:

$$\textit{cosine similarity}\ (A, B) = \frac{AB}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.4}$$

Where, in text matching, the attribute vectors A and B are usually the term frequency vectors of two text documents. The cosine similarity also is a method of normalizing document length during the comparison. The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning the same. The angle θ indicates independence, and in-between values indicate intermediate dissimilarity or similarity between vectors A and B. In information retrieval, the cosine similarity of two documents will range from 0 to 1 since the term frequencies (TF–IDF weights) cannot be negative. The angle between two term frequency vectors cannot be greater than 90°. When the angle is 90°, it means that the two term frequency vectors are opposite [Baeza-Yates and Ribeiro-Neto, 1999; Gunawardana and Shani, 2009].

2.4 Related Work

This section discusses previous work that have similar goals and how they differ from ours along two main dimensions.

Collaboration in Open–Source Software Development. Several studies have discussed diverse aspects of collaboration in open–source software development [Lakhani and Hippel, 2004; Gamalielsson and Lundell, 2014; Gousios et al., 2015, 2016; Linåker et al., 2018; Zhou et al., 2019]. However, each one focuses on distinct aspects and perspectives. For instance, two studies investigated the "fork & pull" development model from the integrators' [Gousios et al., 2015] and contributors' [Gousios et al., 2016] perspectives. Both studies investigated practitioners' working habits and challenges alike. Gousios et al. [2015] investigated which motivation keeps contributors working on the project and how not to discourage them. In a complementary study, Gousios et al. [2016] 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 works investigated different perspectives of "fork & pull" development,

they do not target collaboration as we do in this thesis. For instance, Chapter 3 presents an interview study with twelve experienced developers of open–source software projects hosted on GitHub. In such study, we aim to investigate how the community supports an increasing number of developers through spontaneous collaboration.

Zhou et al. [2019] 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 correctly using fork-based development and helping developers to detect redundant development. In contrast, we focus on a qualitative analysis based on interviews (Chapter 3) and a survey (Chapter 4) with experienced software developers to understand the barriers and motivations 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 the collaboration occurs, in the perspective of developers.

Social theories propose that the sustainability of the community depends on engaged and aware community members to support demands of the community [Gamalielsson and Lundell, 2014; Butler et al., 2002]. Some studies investigate the use of GitHub social features to understand developer behavior and skills [Oliveira et al., 2021; Oliveira, 2022], to evaluate projects' success, and to identify potential collaboration opportunities. For instance, Marlow et al. [2013] observed that developers use signs (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. [2014] found that both technical and social information influence the evaluation of contributions. McDonald and Goggins [2013] noted that one of the main reasons for the increasing number of contributions and developers to a project are features provided by GitHub. Unlike these papers, this doctoral thesis focuses on how the developers perceive the collaboration process in the project. We investigated the motivations and the strategies to find and, when possible, retain developers in a project. Additionally, our study focuses on identifying developers' technical and social aspects that can impact their project contributions (Chapter 3 and 4).

Developer Recommendations for Collaborative Interactions. The relationship between developers is collaborative when they interact with each other to achieve a common goal or do a task in an intellectual effort. Table 2.1 shows an overview of many works in the literature related to recommendations for collaborative interactions in software engineering. For instance, Minto and Murphy [2007] ranked a list of the likely emergent team members with whom to communicate based on a set of files of interest. They applied a simple frequency-based weighting to form these recommendations. Be-

sides, they extracted information available from the Subversion repositories. In our work, we also recommend developer to developer (user–user) of a project (Chapter 5). We applied the TF–IDF score as a "relevance file scoring" and a classifier based on cosine similarity measure to support recommendations using different code activity information (number of commits and quantity of changed LoC).

Surian et al. [2011] recommended a list of top developers that are most compatible based on their programming language skills, past projects, and project categories they have worked on before. In addition, they extracted information available from SourceForge.Net. In our context, we collected information based only on source code changes from the target project in which the developer collaborates with other developers or with the project. Besides, we collected data from GitHub. Canfora et al. [2012] identify and recommend mentors for newcomers in software projects by mining data from mailing lists and versioning systems. They used raw frequencies (TF) and asymmetric Dice coefficient as techniques to construct the recommendation list. They evaluated the approach on data from five open–source projects: Apache httpd, the FreeBSD kernel, PostgreSQL, Python, and Samba. In our context, we also recommend a list of developers with knowledge in specific parts of software projects to connect them with other developers of the project (Chapter 5). However, we extended our recommendations for all active collaborators of the project that need help or want to follow the work of another developer. As mentioned before, we used TF–IDF and cosine similarity measures.

Table 2.1: An overview of prior works that explored recommendation strategies.

Paper	Purpose	Feature Set	Techniques*	Repository
Minto and Murphy [2007]	user-user	source code data	IR	Subversion
Kagdi and Poshyvanyk [2009]	user-task	source code data	IR/SM	Subversion
Surian et al. [2011]	user-user	technical skills and project data	SM	SourceForge.Net
Canfora et al. [2012]	user-user	social interactions and project data	IR/SM	Git-based and community repository
Jiang et al. [2015]	user-task	social interactions and project data	ML	GitHub
Thongtanunam et al. [2015]	user-task	source code data	SM	Gerrit code review
Rahman et al. [2016]	user-task	technical skills and source code data	SM	GitHub
Costa et al. [2021]	user-task	project data	IR	GitHub
Our work	user-user	source code data	IR/SM	GitHub

**The acronyms used in the "Techniques" column stand for: Information Retrieval (IR) methods: Includes topic detection, term frequencies, relative category frequency, semantic filtering, ranking functions. Machine Learning (ML) methods: Includes support vector machines. Similarity Measure (SM): Includes cosine similarity, euclidean distance, asymmetric Dice coefficient, random walk restart compatibility metric.*

Moreover, other works focused on recommendations whose general purpose is to recommend developers for a specific task. For instance, Kagdi and Poshyvanyk [2009]

recommend a ranked list of developers to assist in performing software changes. They combined Latent Semantic Indexing (LSI) techniques with Mining Software Repositories (MSR) to recommend a ranked list of candidate developers for the source code change. They obtained data from the Subversion repository. Jiang et al. [2015] recommended core members for contribution evaluation. They used Support Vector Machines techniques (SVM) to analyze different kinds of features, including file paths of modified codes, relationships between contributors and core members, and the activeness of core members.

Thongtanunam et al. [2015] recommended reviewers based on past reviews of files with similar names and paths. First, it finds past reviews with files whose paths and names are similar to those in the patch under review by string comparison. Then, it assigns the same score for all the reviewers from each such past review. Finally, all reviewers are ranked based on the sum of their scores. Based on these results, they aims to help developers to identify appropriate code–reviewers. Rahman et al. [2016] identified an appropriate code reviewer for a pull request. In addition, they analyzed the past developer work experience with external software libraries and specialized technologies used by the pull request.

Zanjani et al. [2016] identified reviewers who have changed files with similar names. Kagdi et al. [2012] recommend a ranked list of expert developers to assist in the implementation of software change requests. They applied the Information Retrieval (IR) based concept location technique to locate source code entities relevant to a given textual description of a change request. Costa et al. [2021] recommended participants for collaborative merge sessions. They analyzed the project history and built a ranked list of most appropriate developers to integrate a pair of branches (Developer Ranking).

Other efforts [Terra et al., 2015; Xu et al., 2017; Ponzanelli et al., 2017; Jiang et al., 2017b; Rahman et al., 2018; Sajedi-Badashian and Stroulia, 2020; Gong et al., 2021; Dey et al., 2021] have been done in context of recommendation projects for developers and vice-versa. For instance, Liu et al. [2020] recommend software features from the users' perspective for designers of the project. Another example, Berkani [2020] recommends friends in social networks. They take into consideration the semantic and social information of users. However, this kind of recommendations is not the focus of our work.

In our context, we recommend user(s) to user based on co–changed files for together contributing to the engagement in the project and enhance the opportunities for collaborations, not only core members or code-reviewers but also any collaborator that has commonly interested. For instance, developers who implement similar features by changing the same files. We showed a list of possible opportunities for collaboration and the surveyed participants (Chapter 5) chose in which of them they could work together with the recommended collaborator, as a mentor, as a tester, as a developer, or as a reviewer. Besides, we evaluate these strategies of recommendations in the context of GitHub repositories using data collected from answers of 102 surveyed developers (Chapter 6). It

is important to say that previous work [Minto and Murphy, 2007; Kagdi and Poshyvanyk, 2009; Jiang et al., 2015; Thongtanunam et al., 2015; Rahman et al., 2016] relies on automatic evaluations based on very limited ground truths that do not necessarily capture the will to collaborate.

Furthermore, the software developer recommendation problem can be defined as producing, for a given developer d, a list of possible collaborators, sorted according to their relevance (interest in collaborating with) to d. Similarly to other recommendation problems (e.g., movie recommendation), our problem can be viewed in two scenarios: a *cold start* scenario and a *non cold start* scenario. The cold start problem refers to the absence of sufficient information regarding a given user, which makes it difficult to produce effective recommendations for that user [Hu et al., 2019]. Following prior studies [Minto and Murphy, 2007; Canfora et al., 2012], in this work, we focus on the *non cold start scenario*, leaving specific solutions to address cold start as future work.

2.5 Concluding Remarks

In this chapter, we introduced the main concepts on collaboration in open–source projects. In addition, we defined the traditional concepts of recommender systems. Furthermore, we presented the recommendation algorithms relevant for this thesis, namely, Term Frequency - Inverse Document Frequency (TF–IDF) and Cosine Similarity. Finally, we discussed the works related to this doctoral thesis. The next chapter presents an interview study aiming to understand the motivations, processes, interactions, and barriers involved in collaboration during software development.

Chapter 3

Software Developer Perceptions on Collaborations

This chapter presents the interview study to understand the motivations, processes, interactions, and barriers involved in collaboration during software development. This knowledge could help us retain developers in the projects and, consequently, guarantee their sustainability. To achieve this aim, we designed a semi-structured interview protocol and interviewed twelve experienced and active software developers of from large collaborative projects from GitHub. 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. The remainder of this chapter is organized as follows. Section 3.1 describes the interview study design. Section 3.2 reports the results of the study regarding the research questions defined for this study. Section 3.3 discusses the limitations of the study. Finally, we end this chapter with some final remarks (Section 3.4).

3.1 Interview Study Design

This section describes the goals, research questions, and method for the interview study. In this study, each interview was transcribed verbatim and applied standard coding techniques for qualitative research.

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 [Qiu et al., 2019; Pinto et al., 2016]. Developers may have expectations concerning the kinds of contributions they want and concerning the roles and responsibilities of each party [Onoue et al., 2013]. 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 aim 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 our results could help project maintainers to optimize collaboration opportunities and attract more community members. To obtain a understanding on how collaboration happens in software development projects, we interviewed twelve experienced developers in open–source software projects in the context of the social coding site GitHub. Furthermore, we set the goals of our interviews using the Goal/Question/Metric template (GQM) [Basili and Weiss, 1984]. Following such a goal definition template, the scope of our interview study is summarized as outlined below.

Analyze individual motivation, collaboration process and challenge and barriers
for the purpose of identifying opportunities for collaborations
with respect to collaboration process
from the point of view of developers and maintainers
in the context of the social coding site GitHub.

To achieve this goal, we consider the following research questions:

RQ_1 **- What are the motivations to work collaboratively?** With RQ_1, we want to investigate the individual motivation of developers. We also discuss the reasons for working independently mentioned by participants.

RQ_2 **- How does the collaboration process occur?** To make a better analysis, we refine the RQ_2 in the following sub-questions.

$RQ_{2.1}$ – *What are the collaborative contributions?* With $RQ_{2.1}$, 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.

$RQ_{2.2}$ – *What are the roles involved?* With $RQ_{2.2}$, we want to comprehend how the developers organize themselves, in particular, considering their roles and interest in different types of collaborative contributions.

$RQ_{2.3}$ – *What communication channel do developers use to support collaboration among them, and how?* With $RQ_{2.3}$, 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.

RQ_3 **- What are the challenges and barriers in working collaboratively?** With RQ_3, 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.

Research Method. To answer the research questions, we performed an interview study, as summarized in Figure 3.1. In this study, we invited GitHub developers and performed individual interviews that were conducted face to face or through Skype. Each interview was transcribed verbatim. For the data analysis, we applied standard coding techniques for qualitative research [Corbin and Strauss, 2014; Creswell and Creswell, 2017]. The sections below explain the details of our method while Figure 3.1 gives an overview of it.

Pilot Study. We met with all researchers involved in this study to discuss an interview protocol. As shown in Figure 3.1, we first decided to run a pilot study by selecting three developers of the open–source projects. 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, while also software developers and technical contributors to GitHub projects. Among others, we learned that time for answering was too short, and we decided to give more time to explore the topic better. Therefore, we reformulated the interview script and excluded the pilot interview data from our analysis.

Participant Selection. As presented in Figure 3.1, to select participants, we first mined Portuguese speakers (the language of the author) and frequent GitHub developers with more than 500 commits in the last three years using GitHub's REST API v3[1], in line with prior studies on GitHub [Viggiato et al., 2019b,a; Oliveira et al., 2017, 2019].

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

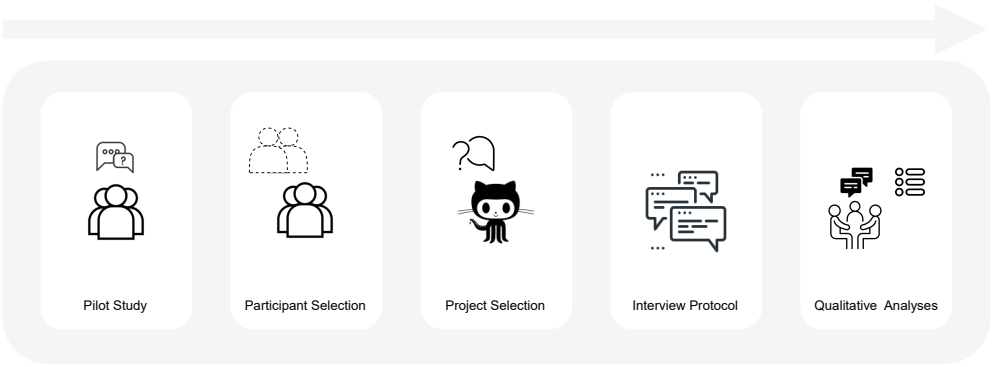


Figure 3.1: Interview study overview

With this last selection criterion, we tried to guarantee that the developer had experience in open–source development and was currently involved in at least one project. Therefore, we could 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 (Figure 3.1). 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 were over 18 years old (condition to be interviewed). In Table 3.1, we summarize demographic information of our interviewees. Participants' demographic information concern gender, last education status (Edu.), years of OSS development experience (OSS Exp.), number of contributions on GitHub for the past three years (GH Contr.), previous or current roles in the project, and their project domains. 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 (Figure 3.1). Moreover, the project could help them remember or focus answers on their collaborative experience a project. When the participant was active in multiple projects, we picked the most popular one in terms of stars and forks. Each participant (P#) answered the interview questions focusing on a project to which they collaborated.

Interview Protocol. We conducted individual interviews that were conducted face to face or through Skype (Figure 3.1). Each interview lasted between 30 to 60 minutes. We recorded and transcribed interviews in order to code them. The interviews were semi-structured based on five guiding questions, as shown in Table 3.2. Since our interviews were semi-structured, the interviewer asked follow-up questions that were not in

Table 3.1: Participants Demographics.

ID	Gender	Edu.	OSS Exp.	GH Contr.*	Roles**	Project Domains
P01	M	B.S.	4	>900	TC/U	Compilers/E-business
P02	M	B.S.	2	>1,900	TC	E-commerce
P03	F	B.S.	4	>2,800	TC/SEO	Artificial Intelligence/Education
P04	M	M.S.	5	>600	TC/ME	E-business
P05	M	B.S.	9	>4,600	TC	E-business/Infrastructure
P06	M	B.S.	11	>3,900	TC	E-Collaboration Tools
P07	F	M.S.	7	>2,800	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	>4,000	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	>600	PM/TC/M/U	Artificial Intelligence

**We recorded the values of the "#" of GH Contributions column in March 2021 (in the last 3 years). **The acronyms used in the "Contribution Type" column stand for: Community Manager (CM), Technical Contributor (TC), Former Project Manager (FPM), Social Event Organizer (SEO), Maintainer (M), Mentor (ME), and User (U).*

the interview script in order to further explore potentially interesting points that participants said. Moreover, we talked for a short time informally with the participants before the interviews to facilitate a friendly and relaxed atmosphere. During the interviews, we encouraged them to talk freely. In some cases, the participants referred to GitHub for clarifications or explanations. All these strategies supported the exploratory nature of our study and allowed us to examine unexpected insights.

Table 3.2: 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 to you?

Qualitative Data Analysis. As detailed in Figure 3.1, we qualitatively analyzed the interview transcripts using standard coding techniques for qualitative research [Corbin and Strauss, 2014; Creswell and Creswell, 2017]. Two researchers analyzed the responses individually and marked relevant segments with codes (tagging with keywords). Later,

the researchers compared their codes to reach consensus and tried to group these codes into relevant categories. With the support of the other three researchers, the codes and categories were discussed to extract key findings and theories.

3.2 Results of the Interview Study

This section presents the results from the perceptions of developers on collaborations according to each research question proposed in Section 3.1.

RQ_1 - What are the motivations to work collaboratively?

The first research question aims to investigate the individual motivation of open–source software developers to work individually or in collaboration. 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 3.3, 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 their opinion on the developed code is essential to make developers more confident,"* 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 3.3, 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:

Table 3.3: Reasons for working collaboratively and independently.

Category	Codes	P(#)	F(#)
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

The acronyms used in the P(#) and F(#) columns stand for: Number of participants that cited the codes(P), frequency of the code (F).

"I prefer to work alone, in my own time, for fun, without any pressure." Our results also 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 [Pinto et al., 2016; Steinmacher et al., 2018; Von Krogh et al., 2012; Pham et al., 2013]. 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.

RQ_1 **Summary:** The motivations of developers for working collaboratively are focused on providing positive results to the project. They highlight the benefits of knowledge sharing, strengthened synergy in teamwork, and increased productivity and code quality. Regarding working independently, 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.

RQ_2 - 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

Table 3.4: Categories and codes for the types of collaborative contributions in the projects.

Category	Codes	P(#)	F(#)
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

The acronyms used in the P(#) and F(#) columns stand for: Number of participants that cited the codes(P), frequency of the code (F).

process.

Collaborative Contributions. In Table 3.4, 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 [Lee et al., 2017] 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 3.4. The pull-based development model helps the control of the contributed code quality by selecting new contributions and incremental code reviews [Tsay et al., 2014; Gousios et al., 2014]. 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 3.4, `Software Development Tasks` and `Issues Management Tasks` are the most prominent categories regarding what participants understand as collaborative contributions into projects.

Table 3.5: Categories and codes for the people roles involved in collaborative software development projects.

Category	Codes	P(#)	F(#)
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

The acronyms used in the P(#) and F(#) columns stand for: Number of participants that cited the codes(P), frequency of the code (F).

Roles Involved. In Table 3.5, we show the roles of developers involved in collaborations. The first column of Table 3.5 presents the two categories we identity: `Software Developer Project Roles` and `Committer Type`. The analysis of Table 3.5 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 3.5. Interestingly, these types of committers match the terminology used in recent research paper [Lee and Carver, 2017]. 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 [Qiu et al., 2019; Miller et al., 2019; Lee et al., 2017]. 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."*

Collaboration Means. In Table 3.6, we present the codes related to the communication channels that developers used to support collaboration among them in open–

source development projects. GitHub issues was the most cited, following by Slack, GitHub Pull Requests, e-mail, Telegram, and Gitter as communication channels for collaboration. Prior works reported similar findings for open–source software projects [Dabbish et al., 2012b; Qiu et al., 2019] and commercial projects [Kalliamvakou et al., 2015].

Table 3.6: Categories and codes for the collaboration channels.

Category	Codes	P(#)	F(#)
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

The acronyms used in the P(#) and F(#) columns stand for: Number of participants that cited the codes(P), frequency of the code (F).

They pointed out that developers usually communicate through text messages. Developers stated that each project is free to adopt its own communication policies, and in some cases, they do not have 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 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 [Giuffrida and Dittrich, 2015; Storey et al., 2016].

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 perception of participants that issue solving contributes significantly to collaborative software development. Bissyandé et al. [2013] 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."* Following a different strategy, P9 shared: *"I sent an e-mail asking whether they could accept being members of the repository's maintainer group."*

RQ_2 **Summary:** Software Development 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. Furthermore, 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. Regarding the communication means, GitHub issues (forum) was the most cited communication tool by developers. Moreover, they usually communicate with text messages. Furthermore, participants pointed out that they use communication tools to ask for help with a new feature, fix an issue, or manage a repository.

RQ_3 - What are the challenges and barriers in working collaboratively?

In order to answer RQ_3, we identify the challenges and barriers that would impact on decisions into collaborative software development. Table 3.7 presents the challenge and barrier categories as follow: `Knowledge and Time`, `Documentation`, `Collaborators`, and `Community Issues`. For each category, we outline the main codes based on the frequency of code and the number of participants citing them.

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 [Minto and Murphy, 2007]. P01 agrees with P07, but looks for knowledge among senior developers of the project (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 extensive 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

Table 3.7: Challenges and barriers faced by participants.

Category	Codes	P(#)	F(#)
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
Doc. 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

The acronyms used in the P(#) and F(#) columns stand for: Number of participants that cited the codes(P), frequency of the code (F).

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 prevalent barriers. 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 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 enough technical knowledge 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 [Steinmacher et al., 2018]. The lack of time is one of the reasons for developers to stop contributing to GitHub projects [Qiu et al., 2019]. Lack of time was also identified as the most common barrier to participation faced by peripheral developers [Pinto et al., 2016] and "no time" was one of the reasons for disengagement in open–source software projects [Miller et al., 2019].

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 [Pham et al., 2013; Stol et al., 2010; Steinmacher et al., 2015b] 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 making 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 [Rastogi and Nagappan, 2016]. Hence, to avoid losing developers or collaborations, P07 alerts: *"If volunteer developers are making several contributions and the core team stops giving feedback to them, certainly, they will abandon that work."*

Other participants faced difficulties when the project decisions were too centralize only on one member of the core team. Many decisions to take, issues to solve, pull requests to review were cumulated or frozen waiting for maintainer decision. P02 reported: *"First, maintaining a project is a too 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"*. Indeed, forgetting is an important factor that can impact software development tasks [Nembhard and Osothsilp, 2001; Krüger et al., 2018].

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 may 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 [Zhou et al., 2019; Stănciulescu et al., 2015]. Consequently, it makes a great set of specialized forks. For projects with a large number of forks, it becomes impracticable [Stănciulescu et al., 2015].

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 [Pham et al., 2013; Pinto et al., 2016]. 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 and integrators are aware of these restrictions [Greiler et al., 2012; Pham et al., 2013; Gousios et al., 2015]. 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 and experienced developers [Qureshi and Fang, 2011; Schilling et al., 2012; Rastogi and Nagappan, 2016; Morrison et al., 2016].

RQ_3 **Summary:** 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.

3.3 Limitations and Threats to Validity

In this section, we clarify potential threats to the study's credibility, discuss some bias that may have affected the study results, and explain our actions to mitigate them. The main threats and our respective actions to mitigate them are discussed below based on the proposed categories of Wohlin et al. [2012].

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 [Wohlin et al., 2012]. 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.

Internal Validity. The internal validity is related to uncontrolled aspects that may affect the study results [Wohlin et al., 2012]. 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) [Bernard, 2017] and phenomenological studies (minimum 6 interviewees) [Morse, 1994]. 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.

External Validity. External validity concerns the ability to generalize the results to other environments, such as to industry practices [Wohlin et al., 2012]. 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 [Wohlin et al., 2012]. The results presented in the study are first and foremost observations and interpretations of the researchers from the interview studies. These results reflect our individual perceptions of practitioners, and our interpretations of their responses. All researchers participated in the data analysis process and discussions on the main findings to mitigate the bias of relying on the interpretations of a single person. Nonetheless, there may be several other important issues in the data collected, not yet discovered or reported by us.

3.4 Concluding Remarks

In this chapter, we reported and analyzed data collected through interviews conducted with twelve developers from different open–source software communities to know how collaborations happen, the process, 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. Next chapter, we present a survey study to cross-validate this interview study and better understand how open the developers are to collaborate with others. The opinion survey was answered by 121 developers (response rate of 12%) who contribute to open–source projects hosted on GitHub.

Chapter 4

Openness for Collaborations

In this chapter, we cross-validate our interview results (Chapter 3) to obtain a deep understanding of how collaboration happens in open–source software development, based on developers' behavior to work with others. Moreover, we aim to identify and check witch the main tasks can increase the opportunities for collaboration. To achieve these aims, we designed and performed an opinion survey answered by 121 developers (response rate of 12%) who contribute to open–source projects hosted on GitHub. All participants are developers familiar with their projects and that made their last commit within the last year. The remainder of this chapter is organized as follows. Section 4.1 describes the survey study settings. Section 4.3 analyzes and reports the results of this empirical study. Section 4.4 discusses some threats to the study validity. Finally, we end this chapter with some concluding remarks (Section 4.5).

4.1 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 (Chapter 3). We describe below the survey goal, research questions, and the steps of the research method.

Goal and Research Questions. In Chapter 3.2, some interviewees (e.g., 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 non–core team developers. However, all collaborations are essential for the sustainability of the project [Gamalielsson and Lundell, 2014]. Hence, all contributions should be valuable and encouraged [Pinto et al., 2016; Pham et al., 2013; Gousios et al., 2014]. 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, we aim to know how open they are to working collaboratively with others and which main tasks are to further opportunities for collaboration in the open-source software development project. Furthermore, we set the goals of the survey using the Goal/Question/Metric template (GQM) [Basili and Weiss, 1984]. Following such a goal definition template, the scope of the survey study is outlined below.

Analyze developers' behavior and preferences
for the purpose of knowing the openness for collaborations
with respect to increase the opportunities for collaboration
from the point of view of developers and maintainers
in the context of open-source software development project.

To achieve this goal, we consider the following survey research questions (RQs):

RQ_1 **- How open are developers to work collaboratively?** With RQ_1, we want to know how developers prefer to work and how they actually work, with respect to: working alone, working with the core team, or working with other developers.

RQ_2 **- What types of activities do developers prefer to collaborate?** With RQ_2, we aim to investigate the types of activities that are more likely to improve

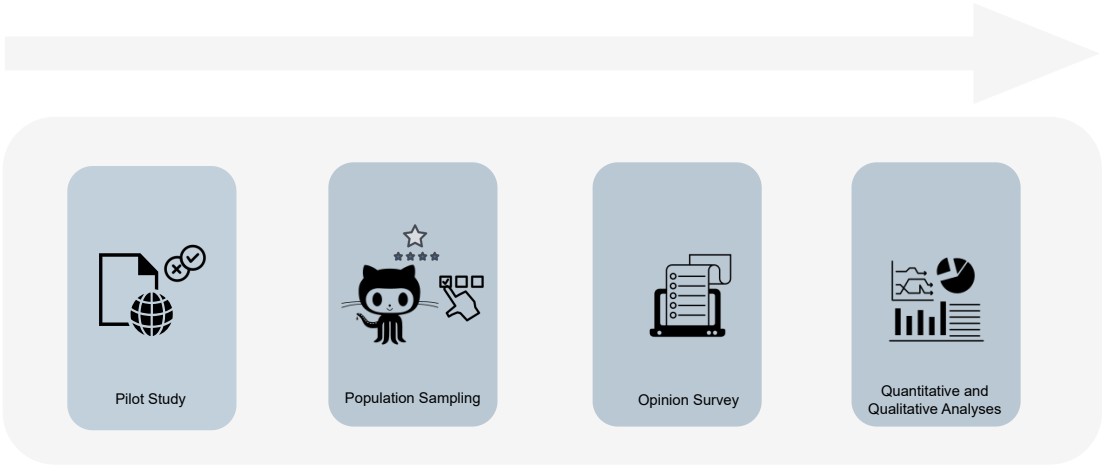


Figure 4.1: Research method overview

collaboration among developers.

RQ_3 **- What other perceptions do developers provide about collaborations?** With RQ_3, we aim to know which are the observations or developers' perceptions provided in the open questions of the opinion survey.

Research Method. To answer the research questions, we performed a survey study. According to Easterbrook et al. [2008], survey studies, usually associated with the application of questionnaires, are used to identify characteristics of a great population. Surveys are meant to collect data to describe, compare, or explain knowledge, behaviors, and attitudes [Pfleeger and Kitchenham, 2001]. Figure 4.1 presents our research method overview. In summary, we select projects hosted on GitHub. We extract data from the original repository and its contributing forks. Additionally, based on some collaborative strategies, we choose the surveyed participants and their similar developers based on code changes. Each participant answers an online questionnaire. The survey study was executed in accordance to the following steps.

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 4.1). 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 [Minto and Murphy, 2007; Jiang et al., 2015; Costa et al., 2021] 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 on contributing forks of open–source projects hosted on GitHub, as presented by Figure 4.1. 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[1] (Version 4.4, 19K stars, 7K forks and C++ as predominate language), Eclipse Deeplearning4J[2] (Version 1.0.0, 12K stars, 5K forks and Java as predominate language), Pandas[3] (Version 1.1.3, 27K stars, 11K forks and Python as predominate language), and Vue[4] (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 [Miller et al., 2019; Krüger et al., 2018].

Opinion Survey. We created a questionnaire[5] on Google Forms composed of questions about the project developers' perception of their collaborative works with other project community members (Figure 4.1). Table 4.1 shows the final version of list of questions (after the pilot study) for this 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.

[1] https://github.com/mongodb/mongo
[2] https://github.com/eclipse/deeplearning4j
[3] https://github.com/pandas-dev/pandas
[4] https://github.com/vuejs/vue-next
[5] The complete questionnaire is presented on Appendix B. Besides, this complete questionnaire was used partially in two study (Chapters 4 and 6).

Table 4.1: List of Questions answered by participants for this study.

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)

Quantitative and Qualitative Analyses. As indicated in Figure 4.1, we collected quantitative and qualitative data from the opinion survey (concerning developer opinions and preferences) and objective data such as demographic information. Section 4.2 presents this demographic information and Section 4.3 presents quantitative and qualitative analysis to answer the research questions of this work.

4.2 Surveyed Participant Overview

Through the GitHub REST API[6], we collected the public and available information of the survey participants in their GitHub profiles as follows. Table 4.2 summarizes statistics on the background of participants such as the number of followers and following as indicators of social interaction and popularity of the participants. They have a median

[6]https://docs.github.com/en/rest

Table 4.2: Background of participants.

	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.*

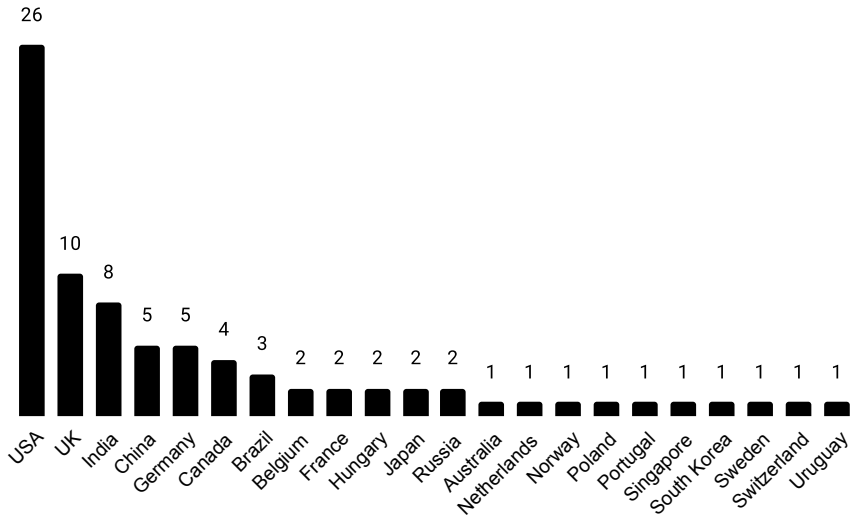


Figure 4.2: Location of the survey participants for of this study.

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 4.2 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.

4.3 Survey Results

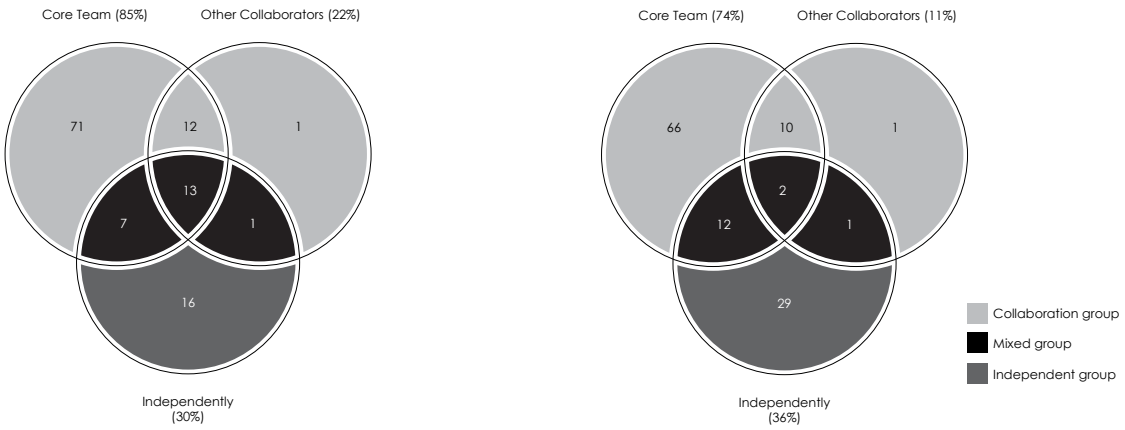
This section describes the results of the survey study regarding the three research questions introduced in Section 4.1. We analyzed 121 responses from 130 participants. We decided to exclude 9 participants that left some questions without answers, i.e., incomplete questionnaires. We also do not include any responses from our pilot study.

RQ_1 - How open are developers to work collaboratively?

To answer the RQ_1, 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?*) (Table 4.1). 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 4.3a and 4.3b 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 4.3a 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 4.3b 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 4.3a and 4.3b 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 4.4 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. [2019] 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. [2016] 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 [2016] argued that



(a) Developer expectations (preference). (b) Developer reality.

Figure 4.3: Collaborative and independent work expectations vs. realities (what developers prefer vs. how it is).

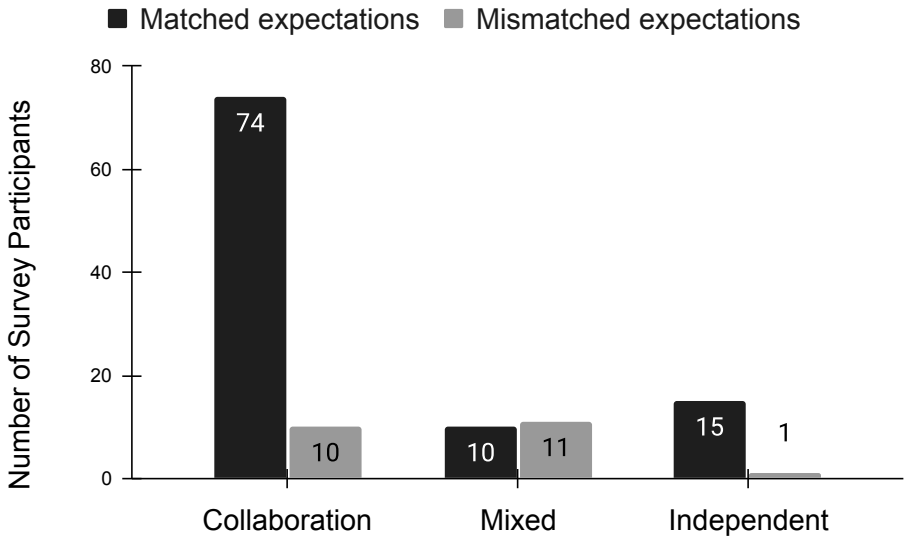


Figure 4.4: Participants clustered in the three groups: Collaboration, Mixed, and Independent.

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 [Blincoe et al., 2016; Farias et al., 2019].

Simultaneously, the `Mixed` Group has the highest number of developers with mis-

matched 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 with project members [Bird, 2011; Zhou and Mockus, 2011], lack of knowledge codebase [Gousios et al., 2016], and others [Steinmacher et al., 2015b], not everyone works well with everyone, conflicting their preferences and styles [Surian et al., 2011]. Although developers are open to work collaboratively, this situation is not always easy when faced with technical or social barriers.

RQ_1 **Summary:** 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.

RQ_2 - What types of activities do developers prefer to collaborate?

To answer the RQ_2, 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 4.5 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.

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

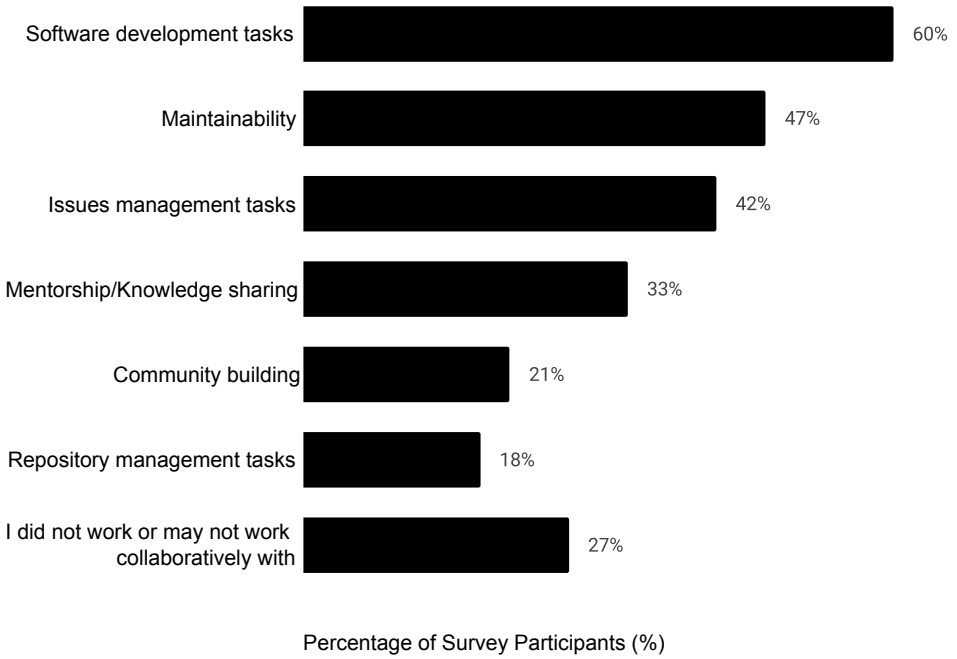


Figure 4.5: Survey results on the developers' prominent task categories to work collaboratively with others in the project.

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 (Section 3.2), 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 4.3a). Figure 4.6 shows the detailed analysis for each type of task, considering two groups from the results of RQ_1: (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. Furthermore, 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

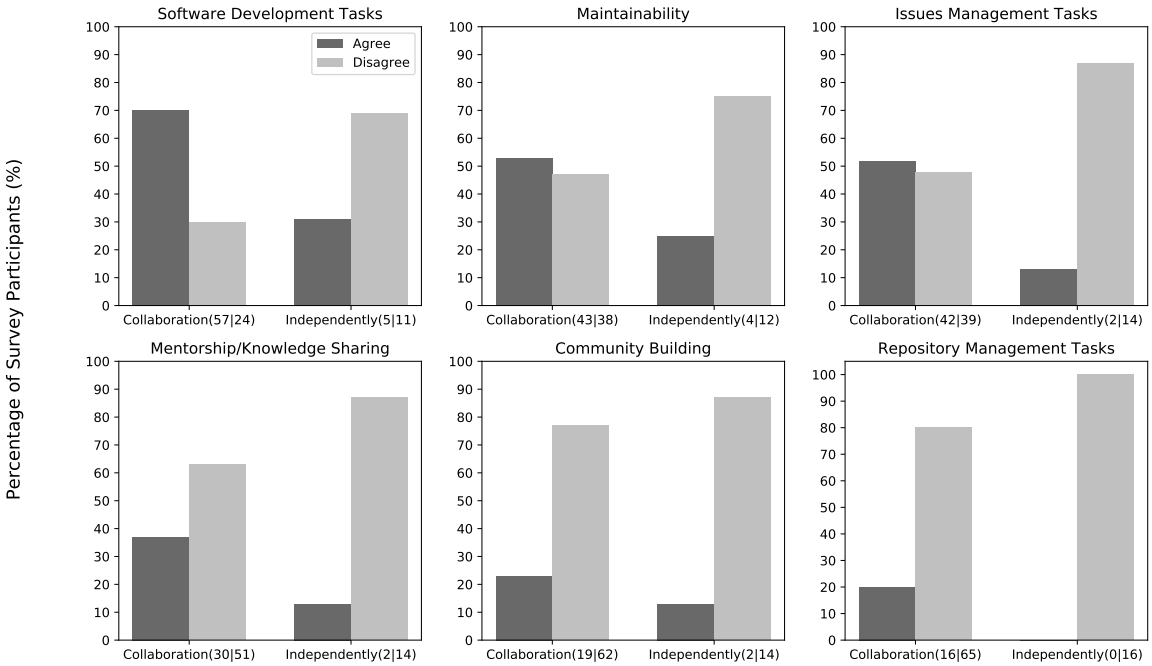


Figure 4.6: Each task category has two groups: collaboration and independent group. The number of participants for each group is in parentheses.

who disagree or did not choose the task category (Disagree).

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 [Tsay et al., 2014; Gousios et al., 2014; Steinmacher et al., 2018] 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 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 [Begel and Simon, 2008; Balali et al., 2020] encourage mentoring to support the integration of the newcomers into projects, and others [Canfora et al., 2012; Steinmacher et al., 2015a] present the difficulty faced by developers to obtain effective mentoring. Cross-validating these results, in our interview study (Section 3.2), 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 (Section 3.2), developers pointed out the motivations/reasons to work independently. However, they are open to working collaboratively if they need some help.

RQ_2 **Summary:** 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.

RQ_3 - What other perceptions do developers provide about collaborations?

To answer the RQ_3, we address the results for the third research question related to other perceptions of surveyed participants. 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 perspective of participant 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 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 [Jiang et al., 2017a]. 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 3.2).

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 3.2), 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."*

RQ_3 **Summary:** Participants stated that 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 the development tasks of the projects.

4.4 Threat to Validity

In this section, we clarify potential threats to validity which may affect the survey study results. We also explain our actions to mitigate them based on the proposed categories of Wohlin et al. [2012].

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 [Wohlin et al., 2012]. We were aware that the results could be affected by the quality of the questions. Thus, we were careful about this threat by preparing a pilot questionnaire before making them available to the participants. Furthermore, 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 [Wohlin et al., 2012]. 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 industry practices [Wohlin et al., 2012]. Indeed, our results are restricted to the OSS projects. However, we tried to variety the domains of the projects. Besides, we filter the participants from different popular and public projects hosted on GitHub to reduce this risk. This way, we believe these participants from different projects can represent a reasonable option to answer the survey reflecting the best samples of the recurrent practices.

Conclusion Validity. The conclusion validity concerns with issues that affect the ability to draw the correct conclusions from the study [Wohlin et al., 2012]. This survey study involved five researchers from three different universities, including an international university. The results presented in this survey are first and foremost observations and interpretations from survey studies. These results reflect individual perceptions of practitioners, and our interpretations of their responses. Hence, 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.

4.5 Concluding Remarks

In this chapter, we cross-validated the interview results presented in Chapter 3 to understand better how collaboration happens in software development projects based on developers' behavior. In particular, we focus on 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. 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 the 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. Hence, in the next chapter, we present

two strategies of developer recommendations with the factors that were identified in our interview study (Chapter 3), for instance, development tasks. We use the co-changed files to recommend developers of the same project. Besides, we present the prototype-tool, namely CoopFinder, to support these developer recommending algorithms.

Chapter 5

Tool-Supported Strategies to Find Collaborators

Although finding the suitable developers to form a team, a partner or a mentor is not a new problem, this subject still find a lot of space with the growth of social coding platforms, such as GitHub, and their impact on open–source software projects. This subject call also the attention of several researchers interested in verifying or identifying what factors may impact collaborations, knowledge sharing or strengthening the bonds between collaborators that retain them in the project [Surian et al., 2011]. Thus, an increasing number of studies seek to provide a direction or address this problem [Surian et al., 2011; Jiang et al., 2015; Balali et al., 2020].

This chapter presents two recommendation strategies of collaborators based on coding activities, especially in co–changed files. Besides, we present the prototype-tool, namely CoopFinder, to support the recommendation strategies. To extract these co–changed files, for Strategy 1, we considered the number of commits. For Strategy 2, we used the number of changed lines of code. To create this context, we considered our interview study results (Chapter 3) that were cross–validated by our survey study (Chapter 4). In our previous studies, participants deemed that software development tasks are the most prominent category regarding what participants understand as collaborative contributions to projects. Thus, we considered the co-change files to strengthen the ties among developers. Other previous works [Minto and Murphy, 2007; Jiang et al., 2015; Costa et al., 2021] also explore co-changed files to recommend developers for different propose in software development projects. The remainder of this chapter is organized as follows. Section 5.1 and 5.2 present the design of two recommendation strategies and an overview of the prototype-tool. Finally, we end this chapter with some concluding remarks (Section 5.3).

5.1 Strategies of Recommending Collaborators

This section presents the details of two recommendation strategies for helping developers to find other collaborators with similar interests and familiarity based on their co–changed files. In our context, we consider a co–changed file as the history of modifications made by two developers on the same file.

Even though developers face some barriers, they are generally still willing to collaborate with the project, mainly whether they are familiar or interested in a specific part of code or the whole project. Thus, we are using the criterion of their interest or familiarity with co–changed files to strengthen or create new ties among developers to improve the opportunities for collaboration and engagement in the project. For example, a developer trying to fix an issue on network security may be interested in other developers familiar with similar problems. Alternatively, a developer interested in data science may pay attention to developers who are experts in building models for statistical analysis in the project. Another example is when developers fork the project for their specific interest. Since they became familiar with the project, when customizing it, they could be potential new collaborators, although their initial intention was not to participate in the project. They could change their mind with some interaction with the other members. Thus, the activities of developers around certain files may reflect their interests and expertise in the project. As a result, if a set of files are relevant in satisfying the same interest and familiarity, it is most likely that similar groups of developers may work together on related or other tasks. Thus, we can improve the interaction between such developers via their code activities.

These strategies are inspired by two previous works [Minto and Murphy, 2007; Canfora et al., 2012]. We adapted the matrix-based computation of the former to support recommendations using different code activity information, specifically co–changed files. We extended the latter to recommend not only mentors, but also all active collaborators of the project that need some help. Figure 5.1 presents an overview of the steps needed to recommend developer to developer in the development project. The strategies to connect them are based on co–changed files relevant to developers. That represents the part of the project that the developers are interested in and familiar with, as presented in Figure 5.1. We exploit Information Retrieval (IR) techniques [Baeza-Yates and Ribeiro-Neto, 1999] to depict developers based on the files they edit. Each step of the general recommendation strategy presented in Figure 5.1 can be described as follows.

Feature Extraction. GitHub is a social coding platform built on top of the GIT version control system and supports the fork & pull model. To define this model, developers make a copy of the original repository and change the project in their copies. When these changes are ready, they can (or not) submit them back into the original

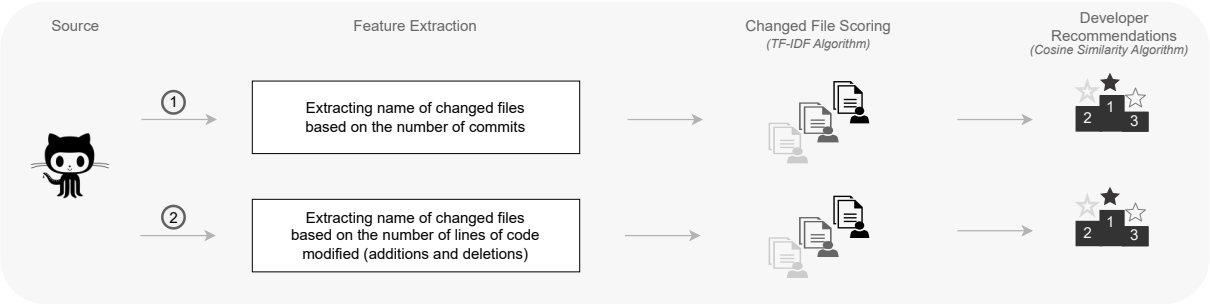


Figure 5.1: Overview of recommendation strategies.

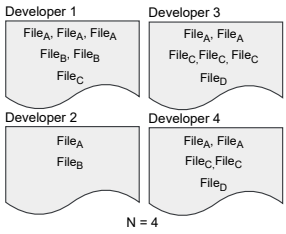
repository by means of a pull request. We assume that two developers are likely changing the same set of files if they have similar interests in the software development project (i.e., they are interested in and familiar with the same part of the project). Thus, we want to determine the similarity of interest among developers of a project. For instance, two developers (Developer 1 and Developer 2) modified the File$_A$, we assumed that they have interests in File$_A$. If only Developer 3 modified the File$_F$, it means that only Developer 3 is interested in File$_F$. We can also have the case that all developers are interested in File$_C$, because all of them modified this file.

To extract these changes, we used the following metrics: *Number of commits*, used in Strategy 1, calculates the number of times a developer has modified a file. *Number of changed lines of code (LoC)*, used in Strategy 2, calculates the sum of the number of code lines added and removed in a specific file that the developer works on. Both aforementioned metrics are calculated considering the whole life time of the project.

Strategies 1 and 2 may provide different (and thus complementary) results, as we will see in Section 6.3. This motivates us to evaluate the joint strategy that aggregates the results of the other two strategies. For both strategies, we extracted these changes from the original repositories and their contributing forks. That is, we collected information from both merged and non–merged changed files.

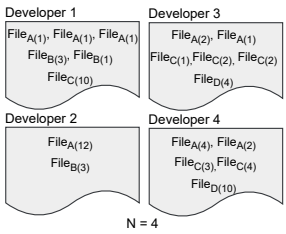
Changed File Scoring. Once we know which set of files developers are interested in or familiar with, we need to know how important each file is for each developer, as presented in Figure 5.1. To this end, we use an algorithm called Term Frequency - Inverse Document Frequency (TF–IDF) [Salton, 1989] to get the rank of relevant files by developers. The definition of the algorithm is adapted for our context. Figures 5.2 and 5.3 present how this algorithm works as a "relevance scoring" for Strategies 1 and 2, respectively.

Term Frequency (TF) assigns a higher relevance score to the most frequent terms (words) in a text document [Yu and Salton, 1976]. In our context, we consider a term as a source code file in the project. Thus, for Strategy 1, TF means the number of times the developer changed a file in the project (*number of commits*). For Strategy 2, TF is defined as the total number of code lines added and removed in the file that the developer



Term	TF_{Dev1}	TF_{Dev2}	TF_{Dev3}	TF_{Dev4}	n_i	IDF_i	$TF\text{-}IDF_{Dev1}$	$TF\text{-}IDF_{Dev2}$	$TF\text{-}IDF_{Dev3}$	$TF\text{-}IDF_{Dev4}$
$File_A$	3	1	2	2	4	1	0.677174	0.551939	0.450066	0.571034
$File_B$	2	1	0	0	2	1.51082562	0.682061	0.833884	0	0
$File_C$	1	0	3	2	3	1.22314355	0.276094	0	0.825743	0.698457
$File_D$	0	0	1	1	2	1.51082562	0	0	0.339985	0.431367

Figure 5.2: TF–IDF results for STRATEGY 1. Terms means the set of files of the project. N means the total number of developers. IDF determines the weight of rare file across all sets in the project.



Term	TF_{Dev1}	TF_{Dev2}	TF_{Dev3}	TF_{Dev4}	n_i	IDF_i	$TF\text{-}IDF_{Dev1}$	$TF\text{-}IDF_{Dev2}$	$TF\text{-}IDF_{Dev3}$	$TF\text{-}IDF_{Dev4}$
$File_A$	3	12	3	6	4	1	0.214763	0.935494	0.329444	0.326566
$File_B$	4	3	0	0	2	1.51082562	0.432626	0.353342	0	0
$File_C$	10	0	5	7	3	1.22314355	0.875620	0	0.671596	0.466010
$File_D$	0	0	4	10	2	1.51082562	0	0	0.663644	0.822307

Figure 5.3: TF–IDF results for STRATEGY 2. For each changed file, the value of the number of changed LoC is in parentheses.

modified (*changed LoC*). Before generating TFs, we perform a pre-processing step aiming at eliminating "stop words". In our context, it means that we eliminate all executable files (e.g., .exe) or compressed files (e.g., .zip and .rar) when these kinds of files were in the project. Every file has its TF calculated according to the strategy adopted. For instance, Developer 1 has modified $File_A$ in three commits (Figure 5.2). For each commit, the developer made one change that can be one addition or one removal of lines of code. Thus, the total changed lines of code count are 3 (Fig. 5.3). In this example related to Developer 1, the TF_{Dev1} results for $File_A$ is the same for both strategies. In the second example, Developer 2 has changed $File_B$ in only one commit (Fig. 5.2). In this commit, Developer 2 made three modifications that can be two additions and one deletion of lines of code. (Fig. 5.3). Thus, the TF results for $File_B$ related to Developer 2 are $TF_{Dev2} = 1$ (Fig. 5.2) and $TF_{Dev2} = 3$ (Fig. 5.3), for STRATEGIES 1 and 2, respectively.

Inverse Document Frequency (IDF) measures how relevant is a file in a project. While computing TF, all files ("terms") are considered equally important. However, we know that some specific files may appear many times but they have little importance in the project to differentiate developer interests. Thus, we need to weigh down the frequent files while scaling up the rare ones. Towards this goal, IDF is calculated as log of the total number (N) of developers divided by the number of developers that modified the file. $IDF = \log\left((N+1)/(n_i+1)\right) + 1$[1]. For instance, all developers modified $File_A$, thus $IDF_{FileA} = 1$ (Figs 5.2 and 5.3). Differently, only two developers have modified $File_B$, thus $IDF_{FileB} = 1.51082562$. Finally, we combine both metrics described above by

[1]This formula adds "1" to the numerator and denominator to prevents zero divisions.

calculating their product (TF*IDF) for each file and each developer in the project. The resulting TF–IDF vectors are then normalized by the Euclidean norm. The final result is a rank of files by developer for each strategy, and consequently among developers. For instance, considering STRATEGY 1 for Developer 2, File$_B$ is more relevant than File$_A$ (Fig. 5.2). However, in STRATEGY 2, the rank of files changed, then File$_A$ became more relevant then File$_B$. Consequently, it changed the relevance of files for the developers. For example, in STRATEGY 1, File$_B$ was relatively more relevant for Developer 2 than for Developer 1 (Fig. 5.2), while STRATEGY 2 provides the opposite result (Fig. 5.3). Another example, for STRATEGY 1, the rank of files to Developer 4 was File$_C$, File$_A$, and File$_D$ (Fig. 5.2). In STRATEGY 2, this rank changed to File$_D$, File$_C$, and File$_A$. When comparing with other developers, this result also changed the level of relevance of File$_A$ for Developers 3 and 4.

COLLABORATOR RECOMMENDER MODEL. After representing developers in a rank of relevant files (the vector space model), we can measure their similarity using the cosine metric that are widely used [Rahman et al., 2016; Franco et al., 2019; Berkani, 2020] because of its potential to quantify the similarity of two objects [Ricci et al., 2011]. Therefore, given two vectors A and B representing two developers, the cosine similarity is calculated using a cross product of these two vectors. The idea of this measure is that the similarity between developers A and B is higher the more files they have in common, and with similar weights. This measure is used in both strategies.

$$cosine\ similarity\ (A, B) = \frac{AB}{\sqrt{A^2}\sqrt{B^2}} \tag{5.1}$$

To summarize, the focus of these developer recommendation strategies is on the contribution based on co–changed files. For STRATEGY 1, we used the number of commits in a code file. However, this metric may suffer from the following drawback. A developer who makes frequent small commits in a code file is considered "more engaged" with the given file than another developer who makes infrequent large commits. To minimize this issue, STRATEGY 2 uses the "LOC metric" by accounting for added or deleted code lines in a project file. By this metric, we may capture the volume of changes reflecting the level of engagement and interest in the file. For both strategies, we do not address the quality of contributions, i.e., we do not distinguish whether some contributions are more or less relevant for the project. Table 5.1 summarizes the input (in) and output (out) of each strategy.

Table 5.1: Summarizing the collaborator recommendation strategies.

	Strategy 1	Strategy 2
in	number of commits	number of changed LoC
out	ranked developers	ranked developer

5.2 CoopFinder Overview

This section presents the overview of CoopFinder[2], a collaborator recommendation prototype tool based on co–changed files for improving the opportunities of collaborations in a specific project.

Implementation Technologies. CoopFinder is implemented as a Web application. We designed our Web tool using client-server architecture and visualization techniques. In addition, we used Python 3 language[3] and a free machine learning library for Python, called scikit-learn libraries[4] for the server-side. To implement views in CoopFinder, we used the analytical data visualization components called HighCharts[5], a JavaScript library for manipulating documents based on data. Finally, we built the tool using components provided by the Bootstrap Framework[6], which includes several stylesheets and jQuery plugins for establishing interactive Web sites or application user interfaces. All these technologies were employed for realizing a dynamic exploration and visualization experience.

How to use CoopFinder. Figure 5.4 shows the screenshots of CoopFinder related to the list of contributors of a selected project. This list corresponds to all contributors that modified any files in their copies. Frame (A) shows the project information to which the collaborators belong to, such as repository name, number of stars, number of forks, and number of open issues. Frame (B) shows the table of all collaborators of the focused project. For each collaborator, this table presents the developer information, such as their avatar, name, fork, number of their followers, following, merged commits, non–merged commits, and the last commit date.

Frame (C) shows the code activities related to the number of commits in upstream, number of the non–merged commits, and the last commit date. With this information aggregated, the user is able to know the status of the collaborator in the project. For instance, users can investigate whether collaborators are still active in the project based on the number of merged commits and the last commit date. On the other hand, the non–merged commits associated with the last commit date indicate that the collaborator may need some help. In this scenario, the maintainer can overview all collaborators interested in the project or create a team based on co–changed files.

Figure 5.5 illustrates the screenshot of CoopFinder with the list of recommended collaborators for the target developer. This list may vary depending on the strategy selected. It also depends on the rank of relevant files for each project developer (Section

[2]https://homepages.dcc.ufmg.br/ kattiana/coopfinder/welcome.html
[3]https://www.python.org/
[4]https://scikit-learn.org/stable/index.html
[5]https://www.highcharts.com/
[6]https://getbootstrap.com/

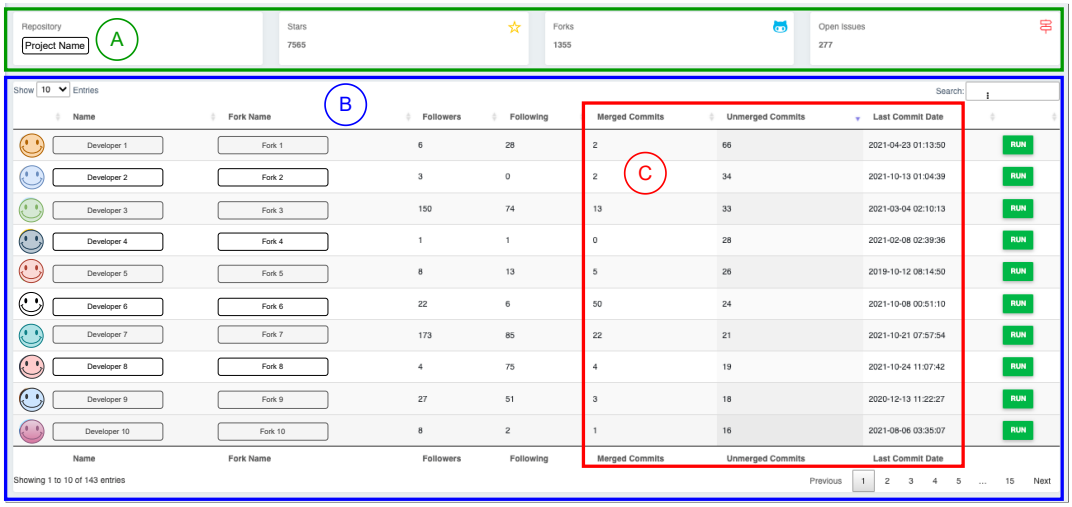


Figure 5.4: Contributors information.

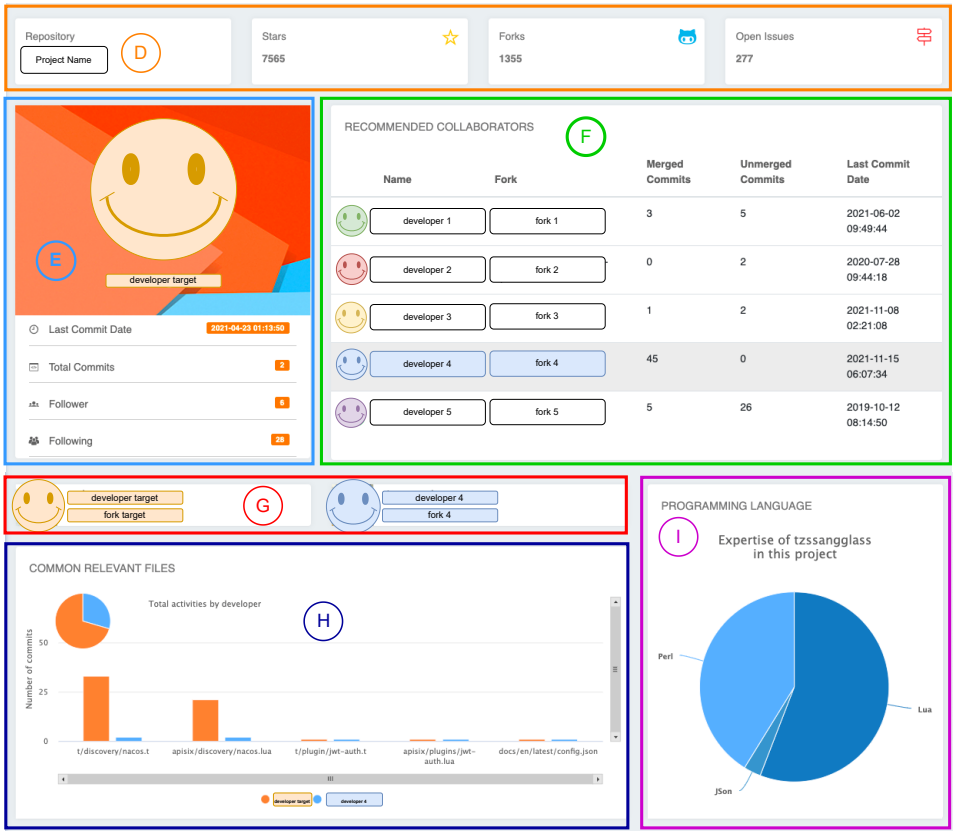


Figure 5.5: Exploring collaborator recommendations.

5.1). Frame (D) shows the project information to which the collaborators belong to, such as repository name, number of stars, number of forks, and number of open issues. Frame (E) shows the target developer and their information, for example, name, avatar, last commit date, the number of total commits, followers, and followings. Frame (F) shows a list of recommended developers with similar interests based on co–changed files. Information of developers is their name, fork, number of the merged commits, number of the non–merged commits, and last commit date.

Users can select one of the recommended collaborators on frame (F) to compare with the target developer on frame (E). Then, frame (G) presents these two collaborators selected, their names, and their fork linked with their GitHub profile. After that, the users can analyze the common files of the two developers, as shown in frame (H). Finally, frame (I) presents the expertise of recommended developer (programming language) related to the focused project. The developer expertise refers to the percentage of files changed in each programming language.

5.3 Concluding Remarks

In this chapter, we presented and discussed two strategies for recommending developers based on co–changed files. We recommend collaborators to engage in the project and enhance the opportunities for collaborations, not only core members, code-reviewers, or mentors but also any collaborator that is commonly interested. We also presented the CoopFinder tool to help developers find collaborators in a specific project. CoopFinder supports both strategies presented in this chapter. When the developer is looking for collaboration among developers of the same project, our tool analyzes their histories based on records of changed files. As a result, CoopFinder suggests a ranked list of suitable collaborators of the project for connecting them. In the next chapter, we discuss a survey study to evaluate these two recommendation strategies based on perspective of who receive the recommendations (project developers).

Chapter 6

Evaluating Recommendations from the Developer Perspective

Software developers must collaborate at all stages of the software life-cycle to create successful complex software systems. However, for large projects with hundreds of involved developers, such as several successful open–source projects, it can be complicated to find developers with the same interest and familiarity with a part of the software and, thus, achieve suitable collaborations and new insights. Resources and efforts may be wasted in this context, discouraging many developers from contributing. Moreover, it can be costly to manage many contributions, which is another challenge for the maintainer who wants to take advantage of this small, timid, but valuable contribution made by a volunteer developer in a short time.

This chapter presents a survey study to evaluate two collaborator recommendation strategies based on co-change files (Section 5) from the perspective of who receives the recommendations. We evaluated these strategies based on an extensive survey with 102 real–world developers. The remainder of this chapter is organized as follows. Section 6.1 describes the survey study settings. Section 6.2 analyzes and reports the perceptions of the developers of this study. Section 6.3 presents the analysis of the joint strategy results. Section 6.4 reports a qualitative analysis. In Section 6.5, some threats to the study validity are discussed. Finally, we end this chapter with some concluding remarks (Section 6.6).

6.1 Study Design

This section presents the design of the survey study to evaluate the recommendation strategies based on co-changed files (STRATEGY 1 and STRATEGY 2) with perspective of who receives the recommendations. We describe bellow our goal, research questions, formulated hypotheses, and the research method.

Goal and Research Questions. We set the goal of our study using the Goal/Question/Metric template (GQM) [Basili and Weiss, 1984]. Following such a goal definition template, the scope of this study is outlined below.

Analyze STRATEGY 1 and STRATEGY 2
for the purpose of evaluation
with respect to precision of recommendations
from the point of view of who receives the recommendations (developers)
in the context of developer recommendations based on co-changed files in the open–source environment.

To achieve our goal, we based our evaluation method on the following research questions.

RQ_1 - Are developers interested in code changed by recommended developers? First, we aim to know the interests of surveyed developers on files changed by other developers in the same project. The lack of interest in the code of the project may be one of the reasons why developers do not participate or leave the project, impacting directly on collaborative work among developers.

RQ_2 - Are developers familiar with code changed by recommended developers? With RQ_2, we aim to know the familiarity of surveyed developers with files changed by other developers in the same project. Developers who know about the specific parts of the project can provide the necessary technical expertise that a project needs and encourage other developers to contribute to the project.

RQ_3 - How precise are the recommendation strategies based on co–changed files? With RQ_3, we aim to know the precision of each recommendation strategy (STRATEGY 1 and STRATEGY 2). We answered this RQ_3 with the surveyed developers that declared the preference to work strictly in collaboration with other developers.

RQ_4 - Do the recommendations contribute to the discovery of potentially new collaborators? In recommendation systems, it is consensus among researchers that the correctness of recommendations is not enough to guarantee the effectiveness and utility of recommendations [Ge et al., 2010; Belém et al., 2016]. One important aspect that must be considered is the recommendation novelty, usually defined as the ability to recommend

items that are different from what is known. Therefore, with RQ_4, we aim at verifying if the recommendations constitute new possibilities for collaborations in the project. That is, collaborators that the target developer is not likely to be aware of.

We defined below hypotheses for RQ_1 aiming to verify which strategy (Strategy 1 or Strategy 2) is the most effective to recommend, to a given target developer d, other developers with similar interests in co-changed files. To answer RQ_1 we evaluated the effectiveness of the strategies in terms of the agreement measure. That is, the percentage of the agreement of the GitHub developers with recommendations. Thus, RQ_1 was turned into the null and alternative hypotheses as follows.

H_0: There is no statistically significant difference in agreement related to interests in co–changed files among developers identified by strategies 1 and 2.
H_a: There is statistically significant difference in agreement related to interests in co–changed files among developers identified by strategies 1 and 2.

We coined the following two hypotheses for RQ_2 to investigate which strategy (Strategy 1 or Strategy 2) is the most effective to recommend, to a given target developer d, other developers with similar familiarity with co-changed files?

To answer RQ_2 we also evaluated the effectiveness of the strategies in terms of the agreement measure.

H_0: There is no difference in agreement related to familiarity with co–changed files among developers identified by strategies 1 and 2.
H_a: There is difference in agreement related to familiarity with co–changed files among developers identified by strategies 1 and 2.

Finally, we also designed hypotheses for RQ_3: which strategy (Strategy 1 or Strategy 2) is more effective to recommend collaborators. As mentioned, to answer RQ_3, we evaluated the effectiveness of the strategies in terms of the precision measure; that is, the percentage of the acceptance answers according to surveyed GitHub developers. Thus, the null and alternative hypotheses are:

H_0: There is no statistically significant difference in precision for developer recommendations among strategies 1 and 2.
H_a: There is statistically significant difference in precision for developer recommendations among strategies 1 and 2.

Let μ be the average amount of agreement (RQ_1 and RQ_2) or precision measure (RQ_3). Thus, $\mu 1$ and $\mu 2$ denote the average amount of agreement or accepted answers by surveyed developers using exclusively either Strategy 1 or Strategy 2, respectively.

Then, the aforementioned set of hypotheses can be formally stated as:

H_0: $\mu 1$ and $\mu 2$ are statistically tied.
H_a: $\mu 1$ and $\mu 2$ are not statistically tied.

To test all aforementioned hypotheses, we considered 95% confidence levels ($\rho = 0.05$).

Precision Measure. Precision measures the correctness of the recommended strategies, i.e., the ratio of correctly recommended developers by the total developers in the recommendation list. To compute precision, we need to know the values of true positives (TP) and false positives (FP). In our context, TP and FP quantify the number of correctly and wrongly recommended developers by the strategy evaluated by the surveyed developers (the oracle), respectively. The computation of precision is: *Precision* = TP / (TP + FP). Precision varies from 0 to 1, and higher values are related to better correctness. In this work, given the time consuming task of manually verifying the relevance of recommendations, volunteer participants were able to evaluate only one recommendation. Thus, for individual recommendations, the obtained values for precision are either 0 or 1. However, we evaluate the recommendation effectiveness in a collective perspective, in which the average precision over all recommendations gives us the expected proportion of top-1 recommendations that are considered relevant by the users[1].

6.1.1 Research Method

To answer the research questions, we planned and performed a survey study, as shown in Figure 6.1. In summary, we select public projects hosted on GitHub. We used GitHub REST API version 3 to retrieve data from the original repositories and their contributing forks. Additionally, based on the two recommendation strategies, we chose the surveyed participants and their similar developers based on co–changed files; we used both merged and non–merged commits. Each participant analyzed only one recommended developer (the recommendation in the first position of the rank) of one (randomly chosen) strategy. Furthermore, we evaluated both strategies in a single inspection, when they

[1]Another measure to evaluate the effectiveness of recommendations is the *Recall*, which measures the completeness of the recommendations, defined by the ratio of correctly recommended developers over the total number of developers which are relevant for the target user (*Recall* = TP / (TP + FN), where FN is the amount of false negative recommendations). However, the already mentioned costs in the evaluation constitute a very limiting factor to compute recall. Thus, we focus our evaluation on the precision measure, leaving the recall analysis as future work.

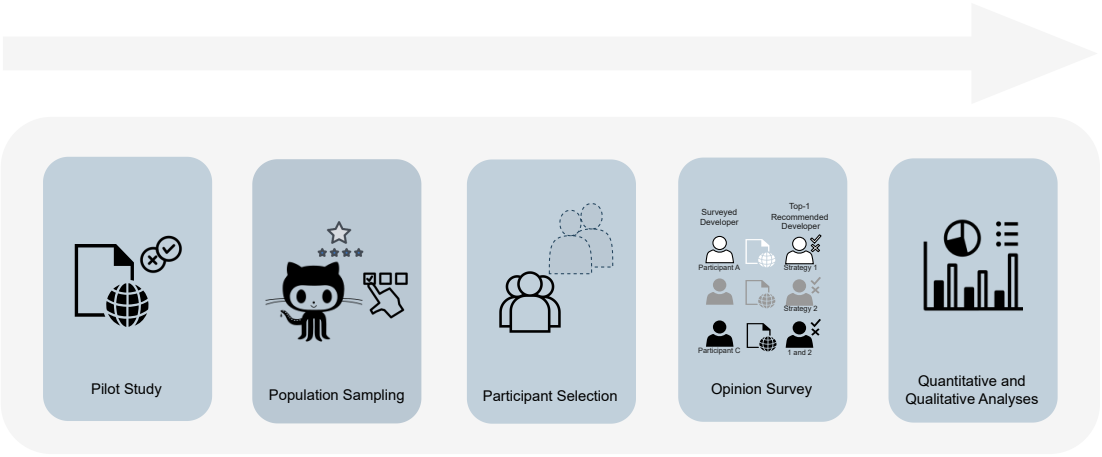


Figure 6.1: Research method overview.

produced the same top-1 recommendation. The survey study was executed under the following steps (see Figure 6.1).

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 one question in the questionnaire. Initially, we asked participants about their current collaboration practices (questionnaire item SQ1, in Table 6.3). However, during the pilot study, we noticed that the current work practice may not reflect the participant preferences. Therefore, we added another question to the questionnaire (SQ4). Another change is related to the number of recommended developers presented for each participant. First, 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 (e.g., low response rate). Thereby, we decided to reduce for one recommended developer per participant. For the same reason, we decided to present only one strategy (either based on the number of commits or based on the number of changed LoC) for each participant, randomly. Finally, for comparison purposes, we also surveyed when these two recommenders provided the same top1-developer recommendation. In Section 6.3, we present this result that could help us to find some insights from participants to merge or improve the developer recommendation strategies.

Population Sampling. The target population sample is composed of developers working in contributing public forks of open–source projects hosted on GitHub, as presented in Figure 6.1. 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 be active with ongoing development. For survey purposes, we considered projects with at least 50 contributors and avoided large projects, for instance, projects with more than 30K forks or 10K commits. This yielded 3,464 repositories, from which we randomly selected 50. The selected repositories include many famous and popular

projects, such as Eclipse Deeplearning4J [2] (Version 1.0.0, 12K stars, 5K forks and Java as predominate language), Pandas [3] (Version 1.1.3, 31K stars, 13K forks and Python as predominate language), and Vue [4] (Version 3.0, 26K stars, 4K forks and TypeScript as predominate language). Afterward, through the GitHub REST API, for both strategies, we collected all merged and non–merged commits of the whole life of the project and its forks. Besides, we automatically collected the name and public 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 [Krüger et al., 2018; Miller et al., 2019]. This filtering of the participants was a matter of evaluation methodology, to have enough data to evaluate the strategies.

Participants Selection. We had tree groups of participants, for Strategy 1, Strategy 2, and Joint Strategy. We sent an equal number of inviting emails for GitHub developers, totalizing 1,113 e-mails. Besides, we invited developers that had common recommendations in both strategies. We had an overall response rate of around 9%. In total, 130 participants completed the survey. We analyzed their responses, and filtered incomplete responses, excluding 12 participants. For this analysis, we also excluded the participants that claimed the preference to work independently because no recommendation strategy would work for them (16 participants). Table 6.1 summarizes the number of participants assigned in each strategy completely randomized. The first line of the table corresponds to all participants who completed the survey. The second line of the table corresponds to the participants eligible for the analysis of this work. All subjects signed a consent form before participating in the study. All subjects already had prior experience with open–source software development. Each subject evaluated only one strategy. We nicknamed the participants as follows: (i) *S1.1* to *S1.32* worked with Strategy 1, (ii) *S2.1* to *S2.23* worked with Strategy 2, and (iii) *S3.1* to *S3.47* worked with recommendations common to both strategies (joint strategy), which means these two recommenders provided the same top1-developer recommendation. Our goal is to use these nicknames while keeping the anonymity of the participants, separating them by the strategy since we did not repeat participants in the study.

Participants Selection. We had tree groups of participants, for Strategy 1, Strategy 2, and Joint Strategy. We sent an equal number of inviting emails for GitHub developers, totalizing 1,113 e-mails. Besides, we invited developers that had common recommendations in both strategies. We had an overall response rate of around 9%. In total, 130 participants completed the survey. We analyzed their responses, and filtered incomplete responses, excluding 12 participants. For this analysis, we also ex-

[2] https://github.com/eclipse/deeplearning4j
[3] https://github.com/pandas-dev/pandas
[4] https://github.com/vuejs/vue-next

Table 6.1: Completely randomized design.

	Strategy 1	Strategy 2	Joint Strategy	Total
Participants that answered the survey (#)	37	31	62	130
Participants eligible for this analysis (#)	32	23	47	102

cluded the participants that claimed the preference to work independently because no recommendation strategy would work for them (16 participants). Table 6.1 summarizes the number of participants assigned in each strategy completely randomized. The first line of the table corresponds to all participants who completed the survey. The second line of the table corresponds to the participants eligible for the analysis of this work. All subjects signed a consent form before participating in the study. All subjects already had prior experience with open–source software development. Each subject evaluated only one strategy. We nicknamed the participants as follows: (i) *S1.1* to *S1.32* worked with STRATEGY 1, (ii) *S2.1* to *S2.23* worked with STRATEGY 2, and (iii) *S3.1* to *S3.47* worked with recommendations common to both strategies (joint strategy), which means these two recommenders provided the same top1–developer recommendation. Our goal is to use these nicknames while keeping the anonymity of the participants, separating them by the strategy since we did not repeat participants in the study.

Survey Demographic Information. Again, through the GitHub REST API, we collected the public and available demographic information of the survey participants in their GitHub profiles. Table 6.2 shows demographic information, such as the number of public repositories they have been interested in following or participated in and the number of contributions in the last three years. Likewise, they have a median of 29 public repositories and 804 commits. Moreover, we mined the public and available roles and technologies information self–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, B, Node.js, Plugins, and React/Redux. We also collected 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. Finally, Figure 6.2 presents the location of the survey participants. Many of them are from the USA, UK, and India, but we observe a wide distribution among other unspecified countries, because this information was not always available (total of 33).

Opinion Survey. We created a questionnaire on Google Forms composed of questions about the project developer perceptions of their similar interests on co-changed files and collaborative works with other developers of the project (Figure 6.1). Appendix

Table 6.2: Participants demographics.

	Mean	St. Deviation	Min	Median	Max
Public Repos	48	51	2	29	244
Contributions	1,716	2,290	38	804	11,093
Followers	80	209	0	20	1,505
Following	11	16	0	4	104

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

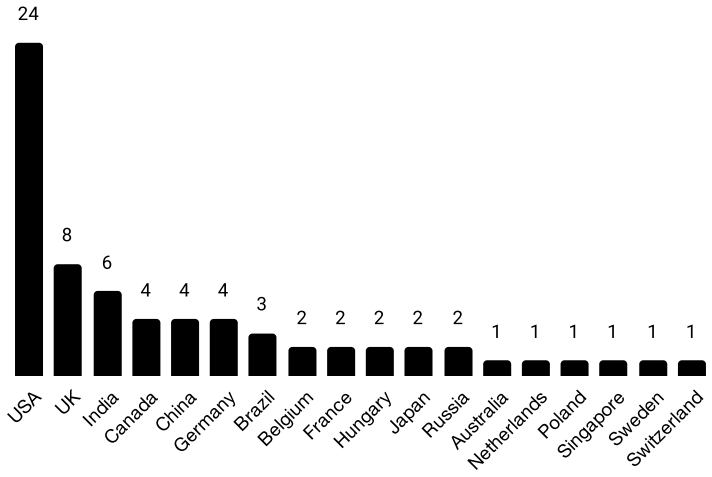


Figure 6.2: Location of the survey participants for analysing the recommendations of this study.

B shows the list of questions for the final version (after the pilot study) related to evaluate the recommendation strategies. This questionnaire is the same presented in Chapter 4, but with another focus. The SQ1 item is to filter out the developers that are not open to work collaboratively with other developers. Developers who prefer to work strictly independent task may not be open for any recommendation. The items SQ2 and SQ3 of the questionnaire are related to relevant co-changed files for both participant and the recommended developer for identifying similar interests.

In the SQ4 item of the questionnaire, we presented for participants a list of possible opportunities for collaborative works with the recommended developer that whether participant chose at least one of the options presented, it means they accepted the recommended developer. Besides, the participant still could add a new task that was not on the list, or even any other comment about other possible collaborative works. On the other hand, whether the participate choose the option *"I did not work or may not work in partnership with the owner of fork"*, it means that participant rejected the recommendation. The SQ5 item allows the participant to identify whether there is any other suitable developer. It is essential information for us to check or to improve the strategies. Finally, SQ6 item is an open space for comments and observations that the participants considered

Table 6.3: Survey questions.

ID	Questions
SQ1	How do you prefer to work on the project? [ ] In collaboration with the core team [ ] In collaboration with owners of other forks [ ] Independently
SQ2	I am interested in some of these changed files in this fork: [ ] Strongly Disagree, [ ] Disagree, [ ] Neither Agree or Disagree, [ ] Agree, and [ ] Strongly Agree
SQ3	I am familiar with some fork changes in these files: [ ] Strongly Disagree, [ ] Disagree, [ ] Neither Agree or Disagree, [ ] Agree, and [ ] Strongly Agree
SQ4	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)
SQ5	Are there other active forks in this project that you know about that you would consider to be of your interest? Which are they? (open question)
SQ6	Other important observations or suggestions (open question)

relevant. These additional questions aim to catch any important issues to participants not asked in the questionnaire.

Quantitative and Qualitative Analyses. First, we collect quantitative and qualitative data from the opinion survey and mined data, such as demographic information. Section 6.2 presents the descriptive analysis of these data and Chi–Squared (χ^2) test [Rayner and Best, 1990; Mendenhall et al., 2012]. We applied the Chi-Squared test to analyze categorical grouped responses to Likert scale questions and to test the hypotheses of no association between the two groups (i.e., to check independence between two variables) [Wohlin et al., 2012]. Furthermore, to apply the Chi-Squared test, we should fulfill three prerequisites: (1) Random data from a population; (2) The expected value of any cell should not be less than five; (3) if the value in any cell is less than five, it should not occupy more than 20% of cells, i.e., in two by two table, no cell should contain an

expected value less than five. Violation of this assumption needs to be corrected by Yate's correction or Fisher's Exact test [Miller and Siegmund, 1982]. To show the effect size the difference between the groups, we compute the *phi coefficient* for a 2×2 contingency table. A value of 0.1 is considered a small effect, 0.3 a medium effect, and 0.5 a large effect. For non 2×2 contingency tables, we use the Cramer's V which is an extension of the phi effect size. Following the guideline (Cohen) [Cohen, 1988], the interpretation depends on the degrees of freedom (df). For example, for the $df^* = 2$, $V = 0.07$ represents a small effect, $V = 0.21$ represents a medium effect and $V = 0.35$ represents a large effect.

We used the R language, RStudio[5], and some statistical R packages. Finally, we qualitatively analyzed the open question in items SQ4 and SQ6 of the questionnaire. In Section 6.4, we applied the open-coding techniques for qualitative research [Corbin and Strauss, 2014]. Afterward, we analyzed the responses and marked relevant segments with codes. Later, we grouped these codes into relevant categories to extract key findings. Conflicts in labelling is resolved by joint discussion among the researchers involved in this study.

6.2 Results

This section presents the results according to each research questions of this study. These results provide insights into the perspectives of developers.

Overview of Participants. This section summarizes the participant responses for the SQ1 item of the questionnaire (*How do you prefer to work in the project?*) (Appendix B). They had the following options to answer (more than one option is allowed): (1) *In collaboration with the core team members*, (2) *In collaboration with the other developers* (owners of forks), and (3) *Independently*. Table 6.4 shows all combinations of these groups based on developer preferences. As detailed in Section 6.1, we focus on analyzing the answers from developers open to work collaboratively (102 developers). It means that we excluded all participants that preferred to work strictly in independent tasks (16 developers). Thus, most participants (68%) declared that they prefer to work collaboratively only with members of the core team. The other participants were open to collaborating with other developers or independent tasks, depending on the project demands.

[5]https://www.rstudio.com/

Table 6.4: Developer expectations (preference).

Group of Preferences	#
core team members	70
core team members, other developers, and independently	12
core team members and other developers	11
core team members and independently	7
other developers and independently	1
other developers	1
Total	102

RQ_1 - Are developers interested in code changed by recommended developers?

To answer RQ_1, we used the participant responses for the SQ2 item of the questionnaire (*I am interested in some of these changed files in this fork*) (Appendix B). For each participant, we presented another developer of the same project and a set of co-changed files. This developer is one of the top-3 recommended developers identified by one of the two scenarios we exploit.

Table 6.5 summarizes of data collected with Likert rating scales [Robbins et al., 2011] for the survey question (SQ2). Participants answered the question with the following options for each statement: (1) "*Strongly Disagree*", (2) "*Disagree*", (3) "*Neither Agree or Disagree*", (4) "*Agree*", and (5) "*Strongly Agree*". For this RQ, we make two analyses. First, we analyzed data about precision of similar interests from the sum of levels 3, 4, and 5 of the Likert scale. Afterward, we followed a more conservative criterion with the levels 4 and 5 of the Likert scale, as presented in the last row of Table 6.5. Thereby, considering the first analysis (3-4-5 of the Likert scale), we obtained the level of 88% and 48% of precision related to the interests in the works of recommended developers from Strategy 1 and Strategy 2, respectively. To check the independence of the strategies, we applied the Chi-Squared (χ^2) test with Yates' continuity correction [Wohlin et al., 2012]. According to Chi-Squared test, the p–value is 0.003, which allows us to conclude that the precision is statistically different for strategies ($\chi^2 = 8.37$, $\alpha = 0.05$). We used the Phi Coefficient to calculate the effect size, which was medium (ϕ=0.43).

Besides, by analyzing the more conservative criterion (4-5 of the Likert scale), we also classified the interest of the developer that informed a higher (≥ 4) concordance with interest in co-changed files (Table 6.5). We obtain 64% and 48% of precision for Strategy 1 and Strategy 2, respectively. The comparison of precision with Fisher's Exact test showed that there were statistically significant differences between strategies (p–$value = 0.001$, $\alpha = 0.05$). In this case, we utilized Cramer's V as a measure of effect size. The calculated value of V was 0.48, indicating a large effect size. Therefore, being interested in code changes of other developers may indicate an openness to creating ties

Table 6.5: Percentage of the surveyed participants related to interest in co-changed files.

Interest in co-changed files		
Likert Scale*	Strategy 1	Strategy 2
1	9%	35%
2	3%	17%
3	22%	0%
4	44%	31%
5	22%	17%
3-4-5	88%	48%
4-5	64%	48%

**Using Likert type rating scale of (1) Strongly Disagree, (2) Disagree, (3) Neither Agree or Disagree, (4) Agree, and (5) Strongly Agree.*

of collaboration. From that perspective, we conclude that most surveyed developers from Strategy 1 are interested in the works of recommended developers.

> *RQ_1 summary:* We observe that developers have interests with other developers based on their co–changed files, especially for Strategy 1. This consideration is relevant because coding tasks may indicate opportunities for contributions to the project. Thus, the interest of the developer in a specific code may motivate them to become a long-term developer in the project or be willing to engage in collaborations.

RQ_2 - Are developers familiar with code changed by recommended developers?

To answer the RQ_2, we used the participant responses for the SQ3 item (*I am familiar with some fork changes in these files*) (Appendix B). We followed the same procedures as explained before for RQ_1 (Section 6.2). That is, we also presented another recommended developer of the same project and their set of co-changed files. Table 6.6 summarizes of data collected with Likert rating scales for the SQ3 question.

We expect that developers familiar with the code may be open to collaborate. They may help each other, give tips for code improvement, code together, or other possibilities. Table 6.6 also presents the percentages of participant answers related to the familiarity with works of recommended developers by each strategy. Again, if we consider the sum of levels 3, 4, and 5 of the Likert–type scale, as presented in Table 6.6, the results are 75% and 52% of precision for Strategy 1 and Strategy 2, respectively. We highlight that Strategy 1 presents a slightly better result. Chi–Squared test with Yates' continuity correction shown that no significant statistical difference for the two samples with small effect size ($\chi^2 = 2.15$, $p\text{–}value = 0.14$, $\alpha = 0.05$, $\phi = 0.23$).

Table 6.6: Percentage of the surveyed participants related to familiarity with co-changed files.

Familiarity with co-changed files		
Likert Scale*	Strategy 1	Strategy 2
1	12.5%	35%
2	12.5%	13%
3	3%	4%
4	25%	26%
5	47%	22%
3-4-5	75%	52%
4-5	72%	48%

**Using Likert type rating scale of (1) Strongly Disagree, (2) Disagree, (3) Neither Agree or Disagree, (4) Agree, and (5) Strongly Agree.*

For the levels 4 and 5 of the Likert–type scale, i.e., a conservative analysis, the results are 72% and 48% of precision for STRATEGY 1 and STRATEGY 2, respectively. In this case, the Fisher's Exact test also shown that no significant statistical difference for the two samples ($p\text{-}value = 0.135$, $\alpha = 0.05$). Cramer's V was 0.24, indicating a medium effect size. In that way, the results were different, but the strategies maintained the same order of precision based on familiarity with co–changed files.

Moreover, the changed files may highlight the opportunities of collaborations among them as participant S2.07 declared "*These are not long-term forks. My fork, as well as the one linked to, are places where we, as core contributors of the project, prepare work before sending it for review. For all of these examples (co–changed files), in fact, we made the changes together.*"

Overall, about half of all surveyed developers from STRATEGY 2 are familiar with other developers based on co–changed files. One of the reasons for these results is that STRATEGY 2 promotes the files based on the number of lines of code added or removed (sum). Some large and unusual commits based on this strategy are considered important for developers [Goyal et al., 2018]. However, in both contexts of analysis, we observed that the developers from STRATEGY 2 were a little less confident in the co–changed files than the other strategy.

RQ_2 summary: Concerning the level of familiarity, we observe that developers identified by STRATEGY 1 are more familiar with co–changed files than developers from STRATEGY 2. Familiarity with code from other developers may indicate one less barrier for improving collaborations and providing code quality.

Table 6.7: Survey results on the task categories of developers to work collaboratively with others in the project.

Task Category	#	%
software development tasks	66	69
maintenance tasks	52	54
issues management tasks	48	50
mentoring tasks	37	39
community building tasks	22	23
repository management tasks	22	23
no tasks	22	23

RQ_3 - How precise are the recommendation strategies based on co–changed files?

The purpose of RQ_3 is to evaluate if the participants accept or reject the recommended developer of the same project after the analysis of interest and familiarity with the relevant co-changed files. To this end, we used the choice of the participants 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...*) (Appendix B). Besides, software development tasks are the most prominent category regarding what participants understand as collaborative contributions to projects, as observed in our prior works [Constantino et al., 2020, 2021]. Each participant analyzed the contributions of another developer of the same project, as presented in Figure 6.1. These contributions were the result of one of the two strategies. As detailed above, each strategy is based on co-changed files across participants and recommended developers. Thus, we asked participants which types of tasks they could work collaboratively with the recommended developer. Note that, we excluded the answers of 6 participants who gave inconsistent responses (they would not work collaboratively in any task). Table 6.7 summarizes the responses of 96 participants. The task categories are: software development tasks (e.g., feature or test suites developing, or code review), maintenance tasks (e.g., improving code/project quality), issues management tasks (e.g., reporting, triaging, or solving issues), and mentoring tasks (e.g., giving/asking help to develop a new feature or fix an issue, and sharing knowledge with the team), community building tasks (e.g., motivating/recruiting developers, or promoting/directing the project), and repository management tasks.

Most developers collaborate in software development tasks (69%). Maintenance tasks (54%), issues management tasks (50%), and mentoring tasks (39%) are also tasks to work collaboratively with other recommended developers. Community building and repository management are the least selected tasks for collaborative work. Participants had the opportunity to further provide an optional comment within the survey. In this optional answer, they commented that Documentation is another category for collaborative

tasks. On the other hand, 23% of them claimed that they would not work collaboratively on any task in the presented recommended developer ("no task"). It means that the developers did not accept the recommendations in this case. Table 6.8 summarizes the responses of participants for non–acceptance or acceptance of the developer recommendation in each strategy. Our results show that the recommendation precision were 80% and 65% for STRATEGY 1 and STRATEGY 2, respectively. Chi-Squared test with Yates' continuity correction shown no significant statistical difference for the two samples with a small effect size ($\chi^2 = 0.73$, *p-value* $= 0.392$, $\alpha = 0.05$, $\phi = 0.16$). As results show that when applied in isolation, STRATEGY 1 performs better than STRATEGY 2.

Table 6.8: Percentage of non–acceptance or acceptance the recommended developers in each strategy.

	Strategy 1	Strategy 2
Non-Acceptance	20%	35%
Acceptance	80%	65%

For SQ5 item of the questionnaire (*Are there other active forks in this project that you know about that you would consider to be of your interest? Which are they?*) (Appendix B), it was impossible to make a quantitative analysis because most surveyed developers did not answer this open question. Few surveyed developers stated general answers, such as, *"yes, the member of core team"* or *"yes, many other developers"*. Besides, other developers claimed on three specific developers (active forks), but one was private, another was the own repository (origin), and the last one was another branch owned by the same developer. Thus, we could not compare these answers with the recommend developer lists. As mentioned before, we received some responses for the open question SQ8, commenting on the co-changed files or recommended developer presented to participants. For instance, some participants emphasized the collaborative relationship among them interacting with each other. Participant S2.08 confirmed this case: "*I and all of the other core contributors are employed to work on this project, we report to the same management chain and communicate frequently outside of GitHub about the project.*"

RQ_3 Summary: We observe that STRATEGY 1 performs better than STRATEGY 2. The precision of acceptance of the former was 80%, while the latter was 65%.

RQ_4 - Do the recommendations contribute to the discovery of potentially new collaborators?

Towards answering RQ_4, we manually categorized each surveyed participant and recommended developers by the type of committer (*core developer*, *casual developer*, and *newcomer*). To categorize as a core developer, we first considered that the core contributor is highly involved and usually contributes to 80 percent of the source code [Yamashita et al., 2015; Barcomb et al., 2018]. Hence, we analyzed the ranking of commits made by GitHub based on the total number of commits and their recent activities. Besides, we also considered the public information (e.g., maintainer of <name of project>) on their GitHub profile and some self–declaration as claimed by S2.08 "*I and all of the other core contributors are employed to work on this project...*" (from the open question in items SQ4 or SQ6, see Table 6.3). To categorize developers as newcomers, we considered the developers with no commits accepted in the project and a total of non–merged commits greater than zero [Begel and Simon, 2008; Yamashita et al., 2015; Steinmacher et al., 2018]. The others were categorized as casual developers, as defined in previous studies [Yamashita et al., 2015; Lee and Carver, 2017; Lee et al., 2017; Barcomb et al., 2018].

To verify the novelty of the developer recommendations, we assume that, for a given project, the core developers already know each other and work collaboratively together, while most casual developers and newcomers are not well known by other developers. Thus, for this analysis (RQ_4), we are interested in investigating if the strategies can present new developers to each other. That means, either recommending casual or newcomers for any developer or recommending core developers for the other groups. Although some of these novel recommendations were not accepted by the surveyed participants (in particular, casual developers, since there were only two cases of a core developer rejecting a recommended casual or newcomer developer), we verified that they still have potential relevance, given that the survey answers indicate that they do not know well the recommended developers.

Tables 6.9 and 6.10 show the recommendations categorized by groups for Strategy 1 and Strategy 2, respectively. The columns are the recommended developers categorized by the *core developers*, *casual developers*, and *newcomer* groups. Besides, these rows in these tables indicate the surveyed participants categorized by the *core developer* and *casual developer* groups. As mentioned before (Section 6.1), we consider developers with the last commit accepted in the last year and with at least four commits accepted to ensure knowledge of the project. Therefore, no newcomer participated in the survey.

Table 6.9 shows the percentage of the recommended developers accepted by each surveyed group from Strategy 1. We defined each recommendation type as <group1>-to-<group2>. It means that recommendations consist of a developer in group1 being

Table 6.9: Percentage of the recommended developers accepted by surveyed groups of the Strategy 1.

	Recommended Developers		
Strategy 1	Core Dev.	Casual Dev.	Newcomer
Core Dev.	13%	7%	0%
Casual Dev.	23%	54%	3%

recommended to a developer in group2. For instance, a recommendation of type *core-to-casual* consists of a core developer being recommended to a casual developer. Our results show that 23% of the recommendations are core-to-casual. Other 54% of recommendations are casual-to-casual. Exploring the novelty of the recommendations, the prior knowledge of developers of the project is not an indicator that they already work together. Thus, it is still possible to explore these opportunities of collaboration in the same or different components of the project. Participant S1.32 (casual developer) detailed this situation: "*I don't work with the specified developer <fork name/project name>. I (independent) work on one component of the concerned module and they (who is also a member of the core team) work on another. Some files are common for both of us.*" Furthermore, participant S1.09 (casual developer) stated that they might work in collaboration with a recommended developer (casual developer). We can understand that this recommendation may be novel for this participant when the participant S1.09 explains their routine of interaction with the other developers (core team members) of the project: "*I am not familiar with other forks (developers) of the <project name>. Usually, I discuss my ideas and thoughts in a designated GH Issue/PR with the core team...*"

Table 6.10: Percentage of the recommended developers accepted by surveyed groups of the Strategy 2.

	Recommended Developers		
Strategy 2	Core Dev.	Casual Dev.	Newcomer
Core Dev.	5%	15%	0%
Casual Dev.	5%	55%	20%

Table 6.10 shows the percentage of the recommended developers accepted by each surveyed group from Strategy 2. For this strategy, 55% of the developer recommendations were casual-to-casual and 20% are newcomer-to-casual. Another evidence from developer perspective about the novelty is when participant S2.16 (casual developer) tries to explain the criteria used to find the developer recommended (casual developer) for them. The participant S2.16 accepted the developer recommendation and explained: "*I do not know how these relevant commits were picked for me, but the only connection is that both of us made independent contributions that affect the same set of files (which is not surprising since they are closely linked).*"

RQ_4 summary: The two recommendation strategies shown favorable results related to novelty. That is, they did not overload the group of core developers. The group of casual developers evaluated developers from all groups, mainly from casual developers and newcomers groups. We emphasize that developers should pay attention to new recommendations (novelty).

6.3 Joint Strategy

As mentioned in Section 6.1, we also decided to analyze the joint strategy for helping us to find some insights from participants to merge or improve the developer recommendation strategies. Table 6.11 presents the percentages of 47 participant answers (Table 6.1) related to interest (RQ_1) and familiarity (RQ_2) with works of recommended developers for joint strategy. Again, if we consider the sum of levels 3, 4, and 5 of the Likert-type scale, the result was 75% for both perspectives. For the levels 4 and 5, i.e., a conservative analysis, the result are 58% and 62% for interest and familiarity, respectively.

Table 6.11: Percentage of the surveyed participants related to interest and familiarity with co–changed files.

Joint Strategy		
Likert Scale*	Interest	Familiarity
1	10%	12%
2	15%	13%
3	17%	13%
4	32%	34%
5	26%	28%
3-4-5	75%	75%
4-5	58%	62%

**Using Likert type rating scale of (1) Strongly Disagree, (2) Disagree, (3) Neither Agree or Disagree, (4) Agree, and (5) Strongly Agree.*

The precision related to the joint strategy was 81% (RQ_3). This result was better than for the other strategies (1 and 2) that may indicate potential for improvements when combining both aforementioned strategies. Although the joint strategy is the best in terms of precision, it does not provide recommendations when Strategy 1 and Strategy 2 do not present recommendations in common, which could affect recommendation completeness as accepted by the recall measure.

Table 6.12: Percentage of the recommended developers accepted by surveyed groups of the joint strategy.

	Recommended Developers		
	Core Dev.	Casual Dev.	Newcomer
Core Dev.	2%	2%	2%
Casual Dev.	11%	70%	13%

Table 6.12 shows the percentage of the recommended developers accepted by each surveyed group from joint strategy (RQ_4). Most of the developer recommendations were casual-to-casual (70%), followed by newcomer-to-casual (13%). Participant S3.13 (casual developer), who accepted the recommendation for working collaboratively with a newcomer, explained that not all changes get his attention in projects with too many forks. Generally, they are most interested in changes that can impact the architectural design of the project. However, participant S3.13 is open for collaborating with any other developer interested in the project, as declared: "*I am open to working together with anyone who has the capability to bring open–source projects further.*"

As mentioned before, we analyzed the joint strategy for comparison purposes. It could help us finding insights from participants to merge or improve the developer recommendation strategies. For example, the number of respondents was higher than the other strategies (even with the same number of invitations as the other strategies), and the acceptance percentage was slightly higher for the joint strategy than for Strategy 1. These examples are positive indicators for combining the strategies. However, the joint strategy could not always provide a recommendation, thus, we need to provide an heuristic, as future work, to decide when using the joint strategy.

Summary: First, in comparison with the other surveyed groups, we observed that participants of this group answered the survey more. Concerning the level of familiarity, we observe that at least 48% are interested in and at least 62% are familiar with co–changed files by the recommended developer. Besides, this joint strategy presented the best precision (81%), which raises evidence of the benefits of combining both Strategies 1 and 2.

6.4 Qualitative Analysis

In this section, we analyze the answers from the open questions of the survey to extract new insights, suggestions, or criticism.

For the data analysis, we employed an approach inspired by the open and axial coding phases of ground theory [Corbin and Strauss, 2014]. The open coding examines the raw textual data line by line to identify discrete events, incidents, ideas, actions, perceptions, and interactions of relevance that are coded as concepts [Corbin and Strauss, 2014]. To do so, one researcher analyzed the responses individually and marked relevant segments with "codes" (i.e., tagging with keywords) and organized them into concepts grouped into more abstract categories. Afterward, a second researcher reviewed and verified the categories created (the conflicts in labelling were resolved by researchers).

Consequently, it is possible to count the number of codes, corresponding percentages, and items in each category to understand frequent feedback from participants. In total, 42% (43/102) of participants filled this field in the questionnaire. Table 6.13 lists the categories and codes from participants. As we can see, the categories are Opportunity of Collaboration, Type of Contributions, Set of Features, Collaborative Workflow, and Characteristic of Changes.

Opportunity of Collaboration. For 21% of participants mentioned their experience in collaboration with the recommended developer or other developers of the project. For instance, participant S2.20 "*I became a maintainer for <project name> in 2017 which was after the specific change on <recommended developer name> was made. It is interesting to see it. In the past 1.5 years, I have become aware of this developer work and may collaborate more with him in the future*". Besides, participant S2.17 mentioned related to openness for collaboration (12%) "*I am on the core team, and we review PRs regularly, so really any active fork will be of interest.*" Finally, participant S2.16 had a positive view related to this study, declaring "*So, what you are really discovering here is that there is a cool way to detect which groups of files are correlated*".

Type of Contributions. For this category, the top-4 type of contributions are code review (19%), fixing bugs (14%), managing pull requests (14%), and discuss ideas (9%). First, participants declared which are their main contributions to the project. They also stated how collaboration happens. For instance, S3.33 explained "*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*".

Set of Features. We did not present for participants which strategy we applied to recommend a developer to them. Hence, we can identify some features that participants mentioned. Following, we can explore them in future works to improve our strategies. For

Table 6.13: Categories and codes for the feedback of (43) participants.

Category	Codes	#	%
Opportunity of Collaboration	collaborative experience	9	21
	openness for collaboration	5	12
	positive view	3	7
Type of Contributions	code review	8	19
	fixing bugs	6	14
	pull requests	6	14
	discuss ideas	4	9
	developing feature	2	5
	merging changes	2	5
Set of Features	committer type	7	16
	functionality	5	12
	organization (company)	5	12
Collaborative Workflow	"fork & pull" development model	7	16
	GitHub is practical environment	2	5
Characteristic of Changes	changes out of date	6	14
	relevance of changes	6	14
	size of changes	3	7

instance, 16% of participants mentioned the type of committer, i.e., either participant or the recommended developer was a member of the core team or a casual contributor. Furthermore, 12% of participants mentioned that they worked in the same functionality (or feature) of the project. Besides, some participants (12%) suggested that they are from the same company, organization, or team. For instance, participant S3.07 explained "*the owner of this fork and I work on the same team at <company name>.*".

Collaborative Workflow. For 16% of the participants explained that they use the "fork & pull" development model in their project. In this model, the developers can make a copy of the original project and made any changes without any permission request. Next, when the changes are ready, they can submit them to review and after to be integrated or not to the original repository. Furthermore, of participants declared that the GitHub repository is a practical environment to develop projects. For instance, participant S3.35 declared "*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*".

Characteristic of Changes. As mentioned earlier, we presented a set of relevant files for participants to analyze the recommendation further. Even though these files were in a recent time window (Section 6.1), about 14% of participants claimed that the changes were outdated and could impact the collaboration. For instance, participant S2.22 declared "*this fork did not have any recent changes, so potential collaboration seems less likely*". Besides, 14% of participants reported the relevance of commits. For some

participants, fixing typos and minor changes are not relevant. On the other hand, participant S3.31 pointed out "*It is rare to work directly on a fork together, unless we are prototyping a big feature*".

6.5 Threats to Validity

We designed and conducted carefully the opinion survey described in this study. For instance, we delimited our scope prior to its execution, defined our hypotheses, and how to assess them. However, some threats to validity may affect our research findings. In this section, we discuss these threats with respective treatments based on the proposed categories of Wohlin et al. [2012].

Construct Validity. The construct validity regards the relationship between theory and observation [Wohlin et al., 2012]. We run our script to filter the GitHub repositories and include project with different rising curve of collaborations among their developers, domains, number of contributors, etc; we defined, as inclusion criteria, the number of stars > 1,000. Another threat is that many casual developers copy the project, contribute back for some time, lose interest and abandon the project. Because they are casual developers and their copies are still active, we cannot guarantee that they still are interested in the project. Interest and familiarity with the source code of the project can impact our study results. Therefore, we try to minimize this threat by inviting only developers who have contributed to the project in the past year.

Internal Validity. The internal validity is related to uncontrolled aspects that may affect the study results [Wohlin et al., 2012]. Our findings may be affected by the unbalance among participant groups or the low number of participants in each strategy. Besides, only certain types of contributors of GitHub may volunteer for a survey (e.g., contributors that modified any files or contributors that their forks are public). However, we carefully mined, using code script, for popular projects (>1,000 stars) to obtain distributed samples of developers of different domain and origin projects for all strategies. Another threat is the use of statistical tools. We paid particular attention to the suitable use of statistical tests (i.e., Chi-Squared test) when reporting our results. This decreases the possibility that our findings are due to random events.

External Validity. The external validity concerns the ability to generalize the results to other environments [Wohlin et al., 2012]. A primary external validity can be the selected projects and participants. We analyzed public and different open–source projects hosted on GitHub, different community sizes, and programming languages, among many available ones. We cannot guarantee that our observations can be generalized to other

projects. For example, participants may not reflect the state of the practice developers. Furthermore, our results could also be different if we were analyzed projects hosted on other repositories, such as private or industrial projects.

Conclusion Validity. The conclusion validity concerns issues that affect the ability to draw the correct conclusions from the study [Wohlin et al., 2012]. The approach used to analyze our survey results represents the main threat to the conclusions we can draw from our study. Thus, we discussed our results by presenting descriptive statistics and statistical hypothesis tests. Besides, 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 collected data, not yet discovered or reported by us.

6.6 Concluding Remarks

In this chapter, we presented and discussed the survey study to evaluate two strategies based on co-changed files in the perspective of who receives the recommendations. We collected the data from an opinion survey answered by 102 GitHub developers. One of our results showed that the acceptance rates were 80% and 65% for Strategy 1 and Strategy 2, respectively. The joint strategy presented the best acceptance rate (81%), which rises evidence of the benefits of combining both Strategies 1 and 2. Besides, the two recommendation strategies showed favorable results related to novelty. In the next chapter, we conducted a controlled experiment to evaluate the developer recommendation strategies. This user evaluation concerned usability and user satisfaction with the proposed strategies and their supporting tool (CoopFinder).

Chapter 7

User Evaluation

Successful software projects demand active collaborators interacting with each other across the complete development life-cycle. Unfortunately, in social coding platforms, such as GitHub, it is still challenging for developers to identify potential collaborators with whom they could engage to create new/stronger ties and enhance the quality of contributions. In Chapter 5, we detailed developer recommendation strategies to help project contributors improve their collaborations. Besides, we described the prototype toll that implements them. Furthermore, in Chapter 6, we evaluated the recommendation strategies from the point of view of who receives the recommendations. Once the strategies and tool are presented, in this chapter, we describe a quantitative and qualitative evaluation of strategies and tool using the state of the practice as a baseline.

To this end, we conducted a controlled experiment to evaluate the developer recommendation strategies and tool. This user evaluation concerned usability and user satisfaction involving 35 participants. We asked participants to perform the experiment tasks to find collaborators with similar interests using a prototype recommendation tool and GitHub. Each participant completed the following tasks: fill a background questionnaire before the experiment, execute two set of tasks, and fill a post–assignment questionnaire about their opinion on the developer recommendations. The remainder of this paper is organized as follows. Section 7.1 describes the user study design. Furthermore, we analyze and report the results of this study (Section 7.2). Section 7.3 discusses some threats to the study's validity. Finally, we end this paper with some concluding remarks (Section 7.4).

7.1 Study Design

This section presents the design of an experiment study to evaluate the developer recommendations based on co–changed files supported by a prototype tool, namely CoopFinder. Due to the Covid-19 pandemic, we performed the experiment remotely. However, all required instructions were available for the participants. Besides, we were

available while participants perform the experiment tasks to clarify any doubts. To collect the data, we adopted questionnaires specially designed for this evaluation by using the Google Forms[1] service. Finally, we describe our goal, research questions, formulated hypotheses, and the research method, as follows.

Goal and Research Questions. We set the goal of our study using the Goal/Question/Metric (GQM) template [Basili and Weiss, 1984], as outlined below.

Analyze a tool–supported recommendation strategy
for the purpose of evaluation
with respect to ease of use and user satisfaction
from the point of view of developers
in the context of developer recommendations based on co–changed files in the open–source environment.

To achieve this goal, we consider the following research questions.

RQ_1 – **How easy is it to find collaborators using CoopFinder?** We compared CoopFinder with GitHub (state-of-the-practice) related to ease of use to find collaborators. Davis [1989] defined ease of use as the degree to which a user believes that using a specific system would be effort free.

RQ_2 – **Does the expertise with GitHub impact on the effectiveness of finding collaborators?** With RQ_2, we relate the background of participants with their experience with GitHub when using the analyzed tools.

RQ_3 – **How fast is it to find a collaborator using CoopFinder?** In this RQ, we also compared CoopFinder with GitHub (state-of-the-practice) in regard to the time required to perform all tasks for finding collaborators.

RQ_4 – **How do participants perceive CoopFinder?** In this RQ, we report the perceptions of the participants about the CoopFinder tool, as commented by them in the post–assignment questionnaire of the experiment.

RQ_5 – **How could the developer recommendations be improved?** In this last RQ, we report the suggestions of the participants related to developer recommendations features to improve the developer recommendations.

Hypotheses Formulation. We defined hypotheses for RQ_1, RQ_2, and RQ_3, as follows. However, for RQ_4 and RQ_5, we did not define hypotheses. We analyzed and answered them qualitatively. Thus, for RQ_1, we defined: Which tool (CoopFinder or GitHub) would it be easier for finding collaborators with similar interests. To answer

[1]https://docs.google.com/forms, accessed in April 2022

RQ_1, we evaluated the ease of use of the tools in terms of the scale: 1 (very easy), 2 (easy), 3 (hard), and 4 (very hard). Thus, RQ_1 was turned into the null and alternative hypotheses as follows.

> **H_0**: There is no significant difference related to ease of use between CoopFinder or GitHub.
> **H_1**: There is significant difference related to ease of use between CoopFinder or GitHub.

We defined hypotheses for RQ_2: which group (GitHub user or non–GitHub user) would impact the use of CoopFinder or GitHub. To answer RQ_2, we evaluated the answers (correct, incorrect and the blank) that participants provide for each task proposed. Thus, the null and alternative hypotheses are:

> **H_0**: There is no significant difference in the hit rate between the GitHub users and non–GitHub users.
> **H_1**: There is significant difference in the hit rate between the GitHub users and non–GitHub users.

Finally, we designed hypotheses for RQ_3: which tool (CoopFinder or GitHub) requires more time for finding collaborators with similar interests in co–changed files among developers. As mentioned, to answer RQ_2, we evaluated the duration of the tasks in terms of the time required to perform all tasks. Thus, the null and alternative hypotheses are:

> **H_0**: There is no significant difference related to time (in minutes) to perform all tasks using CoopFinder or GitHub.
> **H_1**: There is significant difference related to time (in minutes) to perform all tasks using CoopFinder or GitHub.

Research Method. To answer the research questions, we planned and performed an experiment study, as shown in Figure 7.1.

Participant selection. We selected the participants by convenience and using the snowball recruitment technique (i.e., one participant indicates another one, and so on) [Flick, 2018]. To be eligible to participate in this study, they must be collaborators of software development projects (developers or maintainers), especially collaborators who work on open–source projects in GitHub. Section 7.2 presents the overview of the selected participants. We received responses from 43 participants. Eight participants did not complete all questionnaires; thus, they were excluded. Therefore, we have valid answers from 35 participants.

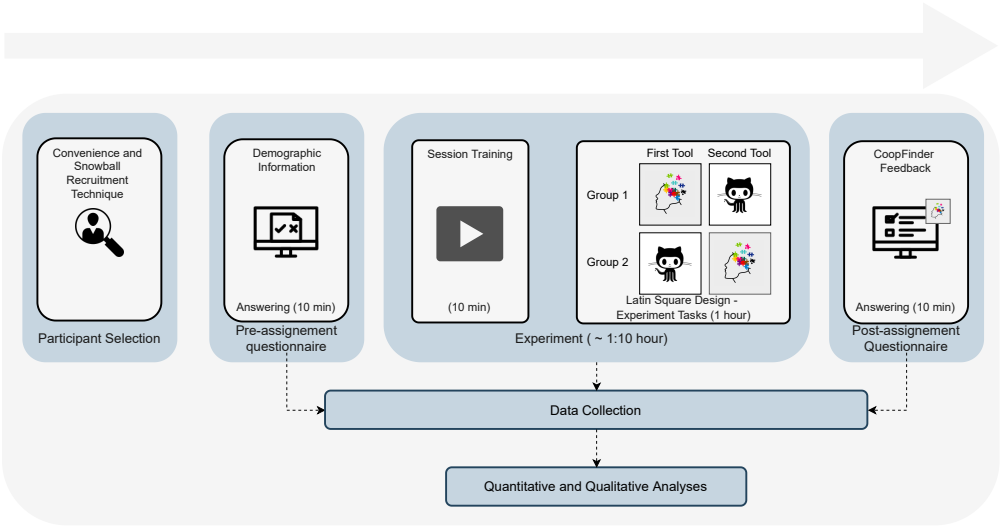


Figure 7.1: Research method overview.

Experiment design. First, we asked participants to complete a demographic and background questionnaire (10 minutes). After, we provided a training and explanation session about the experiment related to CoopFinder and GitHub (10 minutes) (Figure 7.1). After the training session, we asked participants to perform a set of seven tasks for each tool - CoopFinder and GitHub (1 hour). We instructed the participants to perform the tasks using both tools. To reduce the learning effect on the assessment results, we used the Latin square [Fisher, 1992] to distribute the tasks and tools between two groups of participants, as presented in Figure 7.1. Each treatment appears only once in each row (group of participants) and only once in each column (tools), allowing a broader evaluation concerning the tool and the group of participants. Finally, we presented a post–assignment questionnaire with open-ended questions, allowing participants to give feedback about the strategies and tool.

Experiment tasks. We defined and adapted the tasks for each tool to have the same goal (Table 7.1) and difficulty level. Moreover, we presented a brief scenario for each task to direct the activity of the participant to achieve the task goal. For each task, participants should provide an answer for the activity proposed and indicate their perception on how easy it was to perform the task. All questionnaires and tasks are available on Appendix C.2 to C.5.

Post–assignment questionnaire. After the experiment, we sent a short questionnaire to the participants regarding their perceptions about CoopFinder. In this questionnaire, we asked the following questions; and we received responses from all 35 participants.

- What did you think about the CoopFinder tool?
- What are the strengths of the CoopFinder tool?

Table 7.1: List of tasks to be performed by participants.

Task ID	Goal
Task 1	Exploring project information
Task 2	Exploring collaborators of a specific project
Task 3	Exploring (non–merged and merged) commits of the collaborators
Task 4	Exploring similar interests among collaborators
Task 5	Exploring contributions to identify relevant files for the collaborators
Task 6	Exploring developer recommendations
Task 7	Exploring expertises of a specific collaborator

- What are the points to improve this tool?
- What other technical or social information do you think could be explored to improve developer recommendations?
- Would you use and/or recommend this tool? Why?

Data collection. We collected data from the demographic and background questionnaire, the questionnaires of experimental tasks for both tools (CoopFinder and GitHub) and the post–experiment questionnaire related to the feedback of the participants for the CoopFinder tool (Figure 7.1). All data were analyzed, interpreted and reported in the results.

Quantitative and qualitative analyses. First, we collected quantitative and qualitative data from the opinion survey and mined data about the participants in social platforms, such as GitHub. Section 7.2 presents the descriptive analysis of these data and Wilcoxon (W) test [Wilcoxon, 1992]. We applied the Wilcoxon test for testing the statistical significance. This test is non–parametric; it makes no assumptions about the data distribution. Thus, we can use this test when comparing two groups by continuous or ordinal non–normally distributed dependent variables [Wohlin et al., 2012].

We applied the Chi-Squared test to analyze categorical grouped responses to Likert scale questions and to test the hypotheses of no association between the two groups (i.e., to check independence between two variables). Furthermore, to apply the Chi-Squared test, we should fulfill three requirements: (1) random data from a population; (2) the expected value of any cell should not be less than five; (3) if the value in any cell is less than five, it should not occupy more than 20% of cells, i.e., in two by two table, no cell should contain an expected value less than five. Violation of this assumption needs to be corrected by Yate's correction or Fisher's Exact test [Miller and Siegmund, 1982]. All three assumptions were met in our case.

We used the R language, RStudio[2], and some statistical R packages, such as "ggplot2", "scales", and "rstatix". Finally, we qualitatively analyzed the open questions

[2]https://www.rstudio.com/

from the post–assignment questionnaire using open and axial coding techniques [Corbin and Strauss, 2014]. In Section 7.2, we applied open coding techniques for qualitative research [Corbin and Strauss, 2014]. To do so, we analyzed the responses and marked relevant segments with codes (tagging with keywords). Later, we grouped these codes into relevant categories to extract key findings. Conflicts in labelling were resolved by joint discussion among the researchers involved in this study [**?**].

Ethical considerations. This study involves experiments with human subjects. All participants gave the consent for their answers to be used in this research (Appendix C.1). Regarding participant data, all sensitive information (i.e., names or GitHub profile) has been previously anonymized to ensure the privacy of participants. Last, we submitted this research for the Ethical Committee of our institution before performing this study (CAAE: 55476922.0.0000.5149).

7.2 Study Results

This section presents the results regarding each research question of this study. These results provide insights into the perspective of the participants.

Participant Overview. A user study was conducted with 35 participant to evaluate the usefulness and satisfaction of users with the CoopFinder tool. Table 7.2 shows some profiling information of the participants related to gender (26 males and 9 females participants), the time of experience in software development contributions. Finally, if they are or not a GitHub contributor. For 51% of the participants who are not GitHub contributors declared that they already have tried to make contributions to a GitHub project. We also asked them which kind of actions they have taken on GitHub. Participants P02, P20, and P035 noted that they only opened issues for a project. On the other hand, participant P03 faced some difficulties and declared "*I found exciting projects, but due to entry barriers (understanding of the code, time of dedication) I ended up postponing my work.*" This kind of declaration is in accordance with the literature on barriers faced by developers when trying to collaborate in a project [Steinmacher et al., 2015b; Gousios et al., 2016].

Furthermore, participant P21 also declared "*I had difficulty in understanding the code or the lack of help from the leading developers of the project so that I could make the contributions.*" This finding is consistent with literature [Bird, 2011; Zhou and Mockus, 2011; Gousios et al., 2016] related to the barriers of collaboration, such as lack of knowledge about the code–base and lack of interaction with project members. Besides, this result also reinforces the importance of providing support for developers to find appropriate

Table 7.2: Profiling information of the participants.

		#	%
Gender	Female	9	26
	Male	26	74
Software	None	8	23
Development	Less than 1 year	9	25
Contributing	1 year to 3 years	11	31
	More than 3 years	7	20
GitHub	Yes	18	51
Contributor	No	17	49

developers to help them and strengthen the ties among them for improving collaborations in the project.

RQ_1 - How easy is it to find collaborators using CoopFinder?

In this section, we present the results related to the ease of use of each tool (CoopFinder and GitHub), i.e., the degree of effort demanded by participants. We applied the same set of tasks with little adaptations for each tool. The tasks are related to exploring information on the project, collaborators, and their contributions and interests. Each task has a specific goal, as detailed in Table 7.1. However, the general goal of this set of tasks is to make it easier to find a suitable collaborator with similar interests based on co–changed files. Table 7.3 shows the statistical descriptive (Median (Med), Minimum (Min), Maximum (Max), Distribution (D)), and Wilcoxon (W) test result for each task performed by participants using both tools (CoopFinder and GitHub). After participants performed each task, they could express their experience related to ease of use with a scale ranging from 1 (very easy), 2 (easy), 3 (hard), and 4 (very hard).

We applied the Wilcoxon test to compare how easy the tasks were for participants when using CoopFinder and GitHub. According to the Wilcoxon test, the *p–value* for Task 1 is 0.03, and for the others, the *p–value* is less than 0.001, which allows us to conclude that the ease of use is statistically different for CoopFinder and GitHub (Table 7.3). Indeed, CoopFinder is a visual and interactive tool for finding suitable collaborators to improve collaborations into projects. Moreover, the tool provides metadata and links to different attributes that could not be analyzed efficiently using the GitHub interface. For example, this information is related to the source code activities of the collaborators of a specific project. Furthermore, this information can help finding collaborators based on similar interests in files that they have modified.

Table 7.3: Statistic Table.

Tasks	CoopFinder				GitHub				W
	Med	Min	Max	D* 1 2 3 4	Med	Min	Max	D* 1 2 3 4	p**
Task 1	1	1	1		1	2	2		0.037
Task 2	1	1	2		4	1	4		<0.001
Task 3	1	1	2		4	1	4		<0.001
Task 4	1	1	3		4	1	4		<0.001
Task 5	1	1	4		4	1	4		<0.001
Task 6	1	1	3		4	1	4		<0.001
Task 7	1	1	2		1	1	4		<0.001

The acronyms used in the columns stand for: Median (Med), Minimum (Min), Maximum (Max), Distribution (D), and Wilcoxon test (W).
** Note: The scale ranges from 1 (very easy) to 4 (very hard), on experience of participants for each task. ** p–value < 0.05.*

RQ_1 Summary: We observed that participants were able to perform tasks more easily using CoopFinder than GitHub. Wilcoxon test showed that there is statistical difference related to ease of use between CoopFinder or GitHub.

RQ_2 - Does the expertise with GitHub impact on the effectiveness of finding collaborators

In this section, we analyze whether the background related to GitHub expertise of participants can impact the use of the analyzed tools. To this end, we separated the participants into two independent groups (GitHub User group and non–GitHub User group). The former group is for participants who are developers or maintainers of, at least, one open–source project hosted on GitHub. The latter group is for participants who do not have experience with GitHub. Tables 7.4a and 7.4b present the results about the correct (C), incorrect (I) and the blank (B) answers that participants should provide for the activity proposed.

For each independent group, the first and second columns show the number of correct (C) and incorrect (I) answers for each task, respectively. Finally, the *"blank"* (B) column indicates when participants could not answer correctly and left them blank. For this analysis, we applied the Fisher's exact test to compare the hit rate between groups that are GitHub users and non–GitHub users (independent variable) and the answers ("correct", "incorrect", and "blank"), both are qualitative nominal variables.

Table 7.4a shows the predominance of correct answers when participants performed the tasks using the CoopFinder tool. On the other hand, Table 7.4b shows the answers

Table 7.4: Results of tasks performed by GitHub users and non–GitHub users.

(a) CoopFinder

	User (#)			Non-User (#)			
Tasks	C	I	B	C	I	B	p*
Task 1	16	2	0	15	2	0	1.00
Task 2	18	0	0	17	0	0	**
Task 3	16	2	0	17	0	0	0.48
Task 4	15	3	0	16	1	0	0.60
Task 5	17	1	0	12	3	2	0.15
Task 6	11	7	0	12	5	0	0.72
Task 7	18	0	0	17	0	0	**

(b) GitHub

	User (#)			Non-User (#)			
Tasks	C	I	B	C	I	B	p*
Task 1	18	0	0	17	0	0	**
Task 2	7	0	11	9	4	4	0.02
Task 3	2	2	14	2	3	12	0.86
Task 4	2	5	11	2	5	10	1.00
Task 5	3	2	13	4	3	10	0.68
Task 6	2	0	16	3	0	14	0.65
Task 7	15	1	2	14	0	3	1.00

The acronyms used in the columns stand for: correct answers (C), incorrect answers (I), and in blank (B).
** Fisher's exact test (p–value < 0.05).*
*** Test was not applied because the task contains fewer than 2 levels.*

were more distributed when participants used GitHub. The *"blank"* column draws attention to the fact that, except for Task 1, in all other questions, at least half of the participants left the answer blank when they performed the tasks using GitHub. Comments such as *"I didn't find this information"* or *"I don't know"* were common during the execution of the tasks. Participant P22 (GitHub user) explored GitHub to try to answer the tasks correctly. However, P22 stated *"I found it very difficult to find the necessary information on GitHub to do the analyses"*. It reinforces that CoopFinder provides metadata and links to different attributes that could not be explored efficiently using the GitHub interface. Fisher's exact test showed there was no significant difference in the hit rate between the users and non–users groups for almost all tasks (p–$value > 0.05$). When participants used GitHub to perform the task, exploring collaborators of a specific project (Table 7.3), the Fisher's exact test showed a significant statistical difference for the two samples (p–$value = 0.02$).

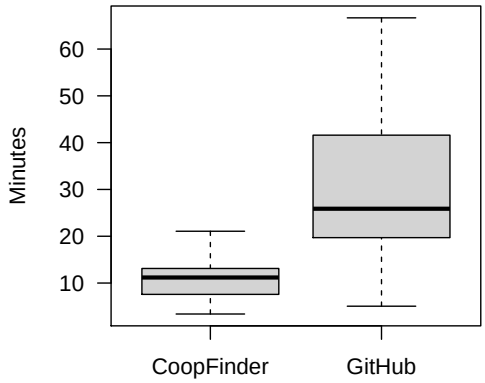


Figure 7.2: Distribution of time (in minutes) of the tasks performed by participants when they used CoopFinder and GitHub.

RQ_2 Summary: We observed the predominance of correct answers when participants used CoopFinder. On the other hand, we also observed the predominance of blank answers when using GitHub indicating that participants either did not know or did not find the correct answers. In general, Fisher's exact test showed no significant difference in the hit rate between the users and non–users groups for all tasks.

RQ_3 - How fast is it to find a collaborator using CoopFinder?

In this section, we analyzed the amount of time it took participants to perform tasks using CoopFinder and GitHub. This amount of time could be taken as an indicator of each tool's ease of use. Figure 7.2 shows the amount of time spent performing the set of tasks using CoopFinder and GitHub. The boxplot represents the median as the horizontal line within the box. Besides, the 25th and 75th percentiles are the lower and upper sides of the distribution box, respectively. Visually, we can notice that performing tasks using GitHub took more time than when using CoopFinder. Table 7.5 presents the descriptive statistic for both tools. For CoopFinder, the median of time spent performing all tasks was 11.2 minutes, and the 25th and 75th percentiles were 7.58 and 13.1 minutes, respectively. On the other hand, the median of minutes spent on GitHub was 25.9. The percentiles in minutes were 19.7 and 38.8 for the 25th, and 75th percentiles, respectively.

We use the Shapiro-Wilk test to verify if the data followed a normal distribuition. Shapiro-Wilk result is 0.659 and $p–value < 0.001$. This $p–value$ suggests a violation of

Table 7.5: Descriptive statistic. Minutes spent performing the tasks using both tools.

	Mean	Med	SD	Min	Max
CoopFinder	11.4	11.2	5.4	3.3	26.7
GitHub	36.1	25.9	33.6	5	159

The acronyms used in the columns stand for: Median (Med), Minimum (Min), Maximum (Max).

the assumption of normality. Afterward, the non–parametric Wilcoxon test showed that there is a difference related to time (in minutes) to perform all tasks using CoopFinder or GitHub (W=9 and $p\text{–}value < 0.001$). It shows that the time required for performing all tasks using CoopFinder and GitHub was significantly different. Combined with Figure 7.2, we observed that participants spent less time using CoopFinder, than using GitHub to perform the tasks.

RQ_3 Summary: We observed that participants spent less time using CoopFinder than GitHub to perform the tasks. This result could also indicate that CoopFinder is easier to use.

RQ_4 - How do participants perceive CoopFinder?

In this section, we report the results of the Post–assignment questionnaire of the experiment. We received responses from 35 participants. For the data analysis, we employed an approach inspired by the open and axial coding phases of ground theory [Corbin and Strauss, 2014]. The open coding examines the raw textual data line by line to identify discrete events, incidents, ideas, actions, perceptions, and interactions of relevance that are coded as concepts [Corbin and Strauss, 2014]. To do so, one researcher analyzed the responses individually and marked relevant segments with "codes" (i.e., tagging with keywords) and organized them into concepts grouped into more abstract categories. Afterward, a second researcher reviewed and verified the categories created (the conflicts in labelling were resolved by researchers).

Perceptions of the participants. In general, the participants commented positive impressions related to CoopFinder. That is, about 49% of the participants pointed out that CoopFinder is exciting and supports project maintainers. For instance, participant P14 remarked *"CoopFinder shows exciting information about developers and projects they are involved."* Furthermore, for other 37% of participants, the tool is easy to use (intuitive or simple). For instance, participant P03 stated, *"Much more practical than GitHub. I could not find any of the requested information in git. The tool clearly shows what I need to do and is much more intuitive"*. Besides, other participants (34%)

considered this tool helpful in finding new developers to collaborate with and manage a possible project. For instance, participant P10 noted *"It is useful both for finding new people to collaborate with and managing a potential project."* Finally, three participants pointed out that the tool needs some improvements.

Strengths. About 43% of participants indicated the easy and intuitive interface as strengths of the tool. For example, P01 pointed out that *"the information about developers and projects would not be easy to retrieve using more popular tools."* Other 40% of participants mentioned that CoopFinder readily provides aggregated and organized information on GitHub projects and their developers, representing an improvement related to finding information or collaborators on CoopFinder. For instance, P02 noted, *"We can quickly locate information about contributors. Besides, we carried out the tasks quickly. I also consider the column with the contributor's fork name very useful. Unfortunately, this information is unclear on the GitHub interface."* Moreover, about 31% of participants voted as a strength the purpose of connecting developers to improve collaborations to project. Furthermore, they mentioned the collaborator rankings, the recommendation based on similar interests, and the general management of collaborators. For instance, P17 commented, *"the strength point of this tool is the comparison of the skills and parts of the project that collaborators have the most in common. Another one is collaborators management."* Finally, 11% of participants mentioned the use of data visualization techniques, participant P09 said *"CoopFinder is a visualization tool for collaboration with a clean and well-organized interface and no visual clutter."*

Weaknesses. We received 32 responses pointing out limitations in CoopFinder. For example, 60% of the participants gave some suggestions to improve the interface. For instance, participants suggested improvements to the design of the buttons to click. Besides, they asked for an interface in *"dark mode"*. About 20% of participants indicated some new functionalities to the tool, such as opening the repository link or direct the user to GitHub, adding textual search, adding some similarity metrics between developer profiles. Besides, the participants also suggested adding new features to improve the way to group collaborators and adding the possibility to analyze other projects.

Recommending the tool. We asked if participants would use or recommend CoopFinder to others. About 66% of the participants answered that they would use or recommend this tool. They explained that CoopFinder may help to better understand the progress of the project concerning the collaborators and who can help whom. For instance, P01 commented, *"Yes. CoopFinder helps a lot in managing collaborators on a project. Besides, you can allocate people with the same interests/skills to work together and other features that GitHub does not have."* On the other hand, 14% of the participants answered negatively and justified that the tool was inappropriate for their work context. For example, participant P28 remarked, *"I would not use it because I do not have or maintain a project with many users where it is needed."* Other participants

(20%) conditioned the use or recommendation of the tool. For example, participant P01 mentioned *"I do not see much use in my daily life, as I work with smaller projects. However, putting myself in the position of the maintainer of large projects, I believe the tool should be handy. If I knew a developer with the mentioned profile, I would recommend it."*

RQ_4 Summary: Participants mentioned that CoopFinder is exciting and supports project maintainers. As for the strengths of the tool, they pointed out its easy and intuitive interface. Besides, about 66% of the participants answered that they would use or recommend this tool. However, other participants (20%) conditioned the use or recommendation of the tool.

RQ_5 - How could the developer recommendations be improved?

In this research question, we asked for participants which social or technical features we could explore to improve the developer recommendation. Table 7.6 summarizes the responses of participants. "Programming language" is the most common suggestion to improve the developer recommendation algorithms (97%); followed by "communication in the project forums" and "professional experience level", with 66% and 63% (Table 7.6).

Table 7.6: Other features to improve the recommendations.

	GitHub			
	User	non–User	Total	
Tasks	#	#	#	%
Programming language	18	16	34	97
Communication in the project forums	13	10	23	66
Professional experience level	12	10	22	63
Language	11	10	21	60
Source code (libraries, APIs, features)	15	5	20	57
Location	3	10	13	37
Followers and following	6	5	11	31
Gender	1	4	5	14

Furthermore, participants also mentioned "language" and "source code (libraries, API, feature)". Several works [Oliveira et al., 2019, 2020] identified developers with expertise in specific libraries from GitHub. Moreover, about 31% of participants indicated the followers and following (Table 7.6). Previous works [Wu et al., 2014; Blincoe et al., 2016] used it as an awareness mechanism to discover new projects and trends. Certainly, these features can be interesting in improving the developer recommendations.

"Gender" is the least common suggestion, with just 5%. It was mentioned mainly for non–GitHub users which may reflect the barriers faced by newcomers collaborators.

For instance, participant P02 noted *"Considering gender issues can be interesting. For example, women will be able to look for other women to collaborate with them. As a result, they feel more comfortable with people of the same gender. That is, they would be in a safe environment."* This result coincides with literature, for instance, Vasilescu et al. [2015a,b] argue that there is discrimination in online software engineering communities, and women are known to face more significant barriers than men. As gender diversity increases, team productivity increases.

Finally, participants cited freely other features, such as participation in issues, previous communication, and openness to answer issues/doubts. Besides, they suggested the developers who participated in new projects and complementary technologies. Finally, they suggested exploring personal profiles, soft skills, and collaboration on similar projects, checking programming language skills based on personal repositories.

RQ_5 Summary: Participants suggested mainly features to improve the developer recommendation system, such as programming language, communications, and professional experience level. They also suggested gender issues, soft skills, and collaboration in similar projects.

7.3 Threats to Validity

Even with careful planning, this research can be affected by different factors which might threaten our findings. We discuss these factors and decisions to mitigate their impact on our study divided into categories of threats to validity proposed by Wohlin et al. [2012].

Construct Validity. This validity is related to whether measurements in the study reflect real-world situations [Wohlin et al., 2012]. This kind of threat can occur in formulating the questionnaire in our experiment (quantitative and qualitative analysis). We designed the questionnaire with open questions as a qualitative study to list users' satisfaction provided by the CoopFinder tool. To minimize this threat, we cross-discuss all the experimental procedures. Basili et al. [1999] and Kitchenham et al. [2002] argue that qualitative studies play an essential role in experimentation in software engineering.

Internal Validity. The validity is related to uncontrolled aspects that may affect the strategy results [Wohlin et al., 2012]. Since we employed a snowballing approach to sampling our participants, we acknowledge that sampling bias affects the selection of the participants, namely self-selection and social desirability biases. However, we counteracted

this effect by inviting people with different profiles, from various projects, and with diverse backgrounds, seeking out different perspectives. Another threat is the use of statistical tools. We paid particular attention to the suitable use of statistical tests (i.e., Wilcoxon test) when reporting our results. This decreases the possibility that our findings are due to random events.

External Validity. The external validity concerns the ability to generalize the results to other environments [Wohlin et al., 2012]. There are three major threats to the external validity of our study, such as baseline tool, the selected projects and participants. First, we chose GitHub as baseline of the experiment, and we cannot guarantee that our observations can be generalized to other tools. Second, we analyzed public and different open–source projects hosted on GitHub, different community sizes, and programming languages, among many available ones. Moreover, we cannot guarantee that our observations can be generalized to other projects. Finally, participants may not reflect the state of the practice developers. Furthermore, our results could also be different if we had analyzed another software development network or projects hosted on other repositories, such as private or industrial projects.

Conclusion Validity. The conclusion validity concerns issues that affect the ability to draw the correct conclusions from the study [Wohlin et al., 2012]. The approach used to analyze our experiment results represents the main threat to the conclusions we can draw from our study. Thus, we discussed our results by presenting descriptive statistics and statistical hypothesis tests. Besides, 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, several other important issues in the collected data may not yet be discovered or reported by us.

7.4 Concluding Remarks

This chapter described a controlled experimental study to investigate the developers' perceptions of using CoopFinder, a prototype tool to support two strategies for recommending collaborations. These developer recommendation strategies aim to connect developers of a specific project based on their similar interests. The study involved 35 participants, 18 of which were GitHub users, and 17 were non–GitHub users. As results, participants pointed out that CoopFinder is easy to use, intuitive, exciting, and supports project maintainer. Besides, we observed that participants were able to perform tasks more easily using CoopFinder than GitHub. For instance, they spent less time using CoopFinder. While GitHub required more time to perform the tasks. It may indicate the ease of use of the CoopFinder tool. Moreover, about 66% of the participants answered that they would use or recommend this tool. The next chapter concludes this thesis by summarizing the main findings of this work.

Chapter 8

Conclusion

Software developers must collaborate at all stages of the software life-cycle to create quality software systems. However, for large projects with hundreds of dynamic developers, such as several successful open–source projects, it can be very complex to find developers with the same interests and, thus, gain suitable collaborations and new insights. Resources and efforts may be wasted in the project context, discouraging many developers from staying. It can be costly to manage so many contributions, which is another question for the maintainer who wants to take advantage of this small, modest, but useful contribution made by a volunteer developer in the shortest possible time. Therefore, this doctoral thesis proposes an investigation of collaborative development based on co-change files to improve the collaborations of a specific project.

This chapter summarizes the results of this doctoral thesis, regarding its goals, and future work. Section 8.1 summarizes the key findings of this thesis. Finally, Section 8.2 outlines possible ideas for future work.

8.1 Summary of the Work and Contributions

This doctoral thesis proposed to investigate how to find collaborations based on co-changed files. To achieve this goal, we defined five specific goals as follows.

- **SG1** Investigate the motivations, processes, interactions, and barriers involved in collaboration during open–source software development.
- **SG2** Investigate how open the developers are for collaboration with others.
- **SG3** Provide tool–supported strategies based on co-changed files to find suitable collaborators.
- **SG4** Evaluate developers recommendations based on co-change files from the point of view of who receives the recommendations.

- **SG5** Evaluate the effectiveness of developer recommendation tools in supporting developers and maintainers, considering both perspectives (GitHub user and non–user).

For SG1, we analyzed data collected through interviews conducted with developers from different open–source software communities to know how collaborations happen, the process, the barriers, and challenges developers face (Chapter 3). Some interesting findings from SG1 are:

- Collaboration transcends coding, and includes documentation and management tasks.
- 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 management skills impact it in defining, categorizing, and sizing tasks accordingly, in such way that the community (including newcomers) can collaborate independently.
- Knowledge management is a challenge in collaboration, and it is important to carefully define communication policies to mitigate and avoid problems related to knowledge retention and decentralization.

For SG2, we designed and performed a survey study to understand better how collaboration happens in software development projects based on developers' behavior. In particular, we focus on how open developers are to work collaboratively with others and the main tasks that increase collaboration opportunities (Chapter 4). Some interesting findings from SG2 are:

- Most participants (86%) prefer to work collaboratively with the core team, 29% prefer to work in independent tasks.
- When exposed to the project's collaborative scenario, the majority of participants selected the category related to software development (65%), maintenance (50%), issues management (45%), and mentorship/knowledge sharing (35%) as the main tasks to work collaboratively with other developers.
- Despite personal preferences to work independently, some developers still consider collaborating with others in some scenarios, especially in development tasks.

The findings from SG1 and SG2 are inputs for the next step related to the SG3, which proposes tool-supported strategies to help developers find collaborators in the open-source projects. Thus, we propose two developer recommendation strategies based on

coding activities, especially in co–changed files, that is, modifications made by developers on the same file (Chapter 5). This set of files can indicate that developers have interests and familiarity with a specific part of the project, impacting directly on collaborative work among developers. To extract these changes, we used the number of commits for Strategy 1. For Strategy 2, we used the number of lines of changed code (code churns). Furthermore, we proposed CoopFinder, a visual and interactive tool that implements the two strategies (Strategy 1 and 2) to connect collaborators based on a set of files of their interest.

For SG4, we evaluated two developer For SG4, we evaluated two developer recommendation strategies based on coding activities from the point of view of who receives the recommendations. Besides, we analyzed the joint of these two strategies and the novelty of their recommendations, i.e., how recommended developers are different from what is known. Thus, we mined data from GitHub public repositories and surveyed 102 developers from these repositories. Besides, we collected the data from an opinion survey answered by 102 GitHub developers of popular projects (Chapter 6). Some exciting findings from SG4 are:

- Concerning the level of interest in and familiarity with co-changed files, we can conclude that developers have a similar interest in the co-change files for two strategies, especially for Strategy 1. These considerations are of relevance because many opportunities for contributions to the project are linked with coding. Thus, this result may indicate one less barrier to improving developers' collaboration.
- The acceptance rates were 80% and 65% for Strategy 1 and Strategy 2, respectively.
- The joint strategies presented the best precision (81%), which raises evidence of the benefits of combining both Strategies 1 and 2.
- The two recommendation strategies have shown favorable results related to novelty. That is, they did not overload the group of core developers. The casual developers evaluated developers from all groups, mainly casual developers and newcomers. We highlight that developers should pay attention to new recommendations (novelty). Many developers are expecting an opportunity to make pertinent contributions to the project.

Finally, for SG5, we conducted a controlled experiment to evaluate the developer recommendation strategies and the CoopFinder. This user evaluation concerned usability and user satisfaction involving 35 participants, of which 18 were GitHub users, and 17 were non-users. All of them are maintainers and/or developers of software projects. As

required, the study was submitted and approved to the Brazilian Committee for Ethics in Research[1]. Some interesting findings from SG5 are:

- We observed that participants could perform tasks more easily using CoopFinder than GitHub. For instance, they spent less time using CoopFinder. While GitHub required more time to perform the tasks. It may indicate the ease of use of the CoopFinder tool.
- Participants mentioned that CoopFinder is exciting and helps project maintainers. They also said, as a strength of the tool, that it is easy and has an intuitive interface. Besides, about 66% of the participants confirmed they would use or recommend this tool. On the other hand, some participants did not see the benefits of using the tool in smaller teams, where collaborators are known. However, other participants (20%) conditioned the use or recommendation of the tool.
- Participants mainly suggested features to improve the developer recommendations, such as programming language, communications, and professional experience. They also suggested gender issues, soft skills, and collaboration in similar projects.

8.2 Future Work

During this research, we perceived many future directions to this doctoral thesis.

Cross-platform solution. The approaches proposed in this thesis to strengthen the collaboration ties among developers can be served in future work as a base to provide a solution for cross-platform, such as Stack Overflow, Linkedin, or other git-based repositories.

Other text mining techniques. We applied classical algorithms such as TF–IDF and Cosine Similarity to rank the files of developers and connect them, respectively. Thus, future work could also explore other algorithms, such as deep learning, active learning, or BERT [Kenton and Toutanova, 2019], which have been the state-art in numerous text mining tasks.

Developer interaction by comments/discussions. In this doctoral thesis, we used the number of commits and the number of lines of code changed (change volume) as metrics to connect project developers. In future work, we could complement the developer recommendations, considering the communication among them, for example, with whom and the frequency of interaction in the question & answer forum. Exploring the potential

[1]Protocol CAAE:55476922.0.0000.5149

of these communication metrics or joining them with metrics related to code activities can bring interesting results.

Developer recommendations based on social network features. In the same sense, exploring the social relationships between developers can be a good line. For example, some developers want to contribute to a project due to the project's popularity. As well as they want to contribute to a project of a specific "influencer". Therefore, identifying which social network feature most attracts developers to a particular project or to work on a particular issue can further strengthen the ties between them and the project.

Developer recommendation based on non-coding activities. Considering that our attempt to recommend developers based on co-changed files was positive, these results are only preliminary indications of the relevance of this work for improving collaborations in the project. Therefore, we also could expand this work to other activities, such as documentation, issue tracking, and repository management activities, that is, the other activities mentioned by the interview and survey study participants (Chapter 3 and 4).

Supporting newcomers. A promising line of research could be to give specific support to newcomers. In this thesis, we consider all developers who took the initiative to implement some coding activity, independently this coding was updated or not to the main project. We highlight these developers that tried. The core team knew some of them, but others were not. Thus, we could pay special attention to newcomers trying to participate and provide them more support (i.e., recommending mentors) to strengthen their ties to the team.

Supporting women or minority groups. Using this work as base, another possibility for future work is to provide recommendations based on a specific group, such as women, or other groups that consider themselves to be minorities and would like a safer environment to start their collaboration on the project. That is, find people aware of these groups.

Group recommendation. In the same direction as the work mentioned above, it is possible to provide recommendations from developers based on their roles in the project. For example, a tester would like to connect with other testers on the project and thus, share more knowledge related to the tests needed for the project.

Recommendation based on technical expertise. This doctoral thesis focused on developers who performed some coding activity on the project. However, we know that there are experienced developers who fork or follow the project but have never tried or had the opportunity to collaborate on the project. In this context, as a future direction, we could identify or even classify the technical expertise of these developers, e.g., coding languages or libraries. Next, we could highlight them to the core team, recommending these developers as promising for the project.

Task recommendation. Another future direction is task recommendation. This work aimed to recommend developers to developers. Thus, they together can choose which activities they want to iterate or collaborate on in the project. Thus, we could also complement our developer recommendation, including specific activities. For example, if the project needs a set of suite tests, we could suggest that the developer collaborates to create these tests based on their coding activities/experience.

Evaluating in real context of use. We also intend to evaluate CoopFinder in real context of use, to see how often the recommendations actually foster collaboration.

Bibliography

Abbattista, F., Calefato, F., Gendarmi, D., and Lanubile, F. Incorporating social software into distributed agile development environments. In *Proceedings of the 23rd International Conference on Automated Software Engineering (ASE)*, pages 46–51, 2008.

Adomavicius, G. and Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

Allenby, G. M. and Rossi, P. E. Marketing models of consumer heterogeneity. *Journal of Econometrics*, 89(1-2):57–78, 1998.

Amati, G. and Van Rijsbergen, C. J. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems (TOIS)*, 20(4):357–389, 2002.

Avelino, G., Passos, L., Hora, A., and Valente, M. T. A novel approach for estimating truck factors. In *Proceedings of the 24th International Conference on Program Comprehension (ICPC)*, pages 1–10, 2016.

Baeza-Yates, R. and Ribeiro-Neto, B. *Modern Information Retrieval*, volume 463. 1999.

Balali, S., Annamalai, U., Padala, H. S., Trinkenreich, B., Gerosa, M. A., Steinmacher, I., and Sarma, A. Recommending tasks to newcomers in oss projects: How do mentors handle it? In *Proceedings of the 16th International Symposium on Open Collaboration (OpenSym)*, pages 1–14, 2020.

Barcomb, A., Kaufmann, A., Riehle, D., Stol, K.-J., and Fitzgerald, B. Uncovering the periphery: A qualitative survey of episodic volunteering in free/libre and open source software communities. *IEEE Transactions on Software Engineering (TSE)*, 46(9):962–980, 2018.

Barcomb, A., Stol, K.-J., Riehle, D., and Fitzgerald, B. Why do episodic volunteers stay in floss communities? In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, pages 948–959, 2019.

Basili, V. R. and Weiss, D. M. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering (TSE)*, (6):728–738, 1984.

Basili, V. R., Shull, F., and Lanubile, F. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering (TSE)*, 25(4):456–473, 1999.

Beel, J., Gipp, B., Langer, S., and Breitinger, C. Paper recommender systems: A literature survey. *International Journal on Digital Libraries (JODL)*, 17(4):305–338, 2016.

Begel, A. and Simon, B. Novice software developers, all over again. In *Proceedings of the 4th International Workshop on Computing Education Research (ICER)*, pages 3–14, 2008.

Belém, F. M., Batista, C. S., Santos, R. L., Almeida, J. M., and Gonçalves, M. A. Beyond relevance: Explicitly promoting novelty and diversity in tag recommendation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(3):1–34, 2016.

Berkani, L. A semantic and social-based collaborative recommendation of friends in social networks. *Software: Practice and Experience*, 50(8):1498–1519, 2020.

Bernard, H. R. *Research Methods in Anthropology: Qualitative and Quantitative Approaches*. Rowman & Littlefield, 2017.

Billsus, D. and Pazzani, M. J. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction*, 10(2):147–180, 2000.

Billsus, D., Pazzani, M. J., et al. Learning collaborative information filters. In *Proceedings of the 15th International Conference of Machine Learging (ICML)*, volume 98, pages 46–54, 1998.

Bird, C. Sociotechnical coordination and collaboration in open source software. In *Proceedings of the 27th International Conference on Software Maintenance (ICSM)*, pages 568–573. IEEE, 2011.

Bissyandé, T., Lo, D., Jiang, L., Réveillere, L., Klein, J., and 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)*, 2013.

Blincoe, K., Sheoran, J., Goggins, S., Petakovic, E., and Damian, D. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology (IST)*, 70:30–39, 2016.

Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

Breese, J. S., Heckerman, D., and Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998.

Burke, R. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

Butler, B., Sproull, L., Kiesler, S., and Kraut, R. Community effort in online groups: Who does the work and why? pages 187–210. 2002.

Cai, Y. and Zhu, D. Reputation in an open source software community: Antecedents and impacts. *Decision Support Systems (DSSs)*, 91:103–112, 2016.

Canfora, G., Di Penta, M., Oliveto, R., and Panichella, S. Who is going to mentor newcomers in open source projects? In *Proceedings of the 20th International Symposium on the Foundations of Software Engineering (FSE)*, pages 1–11, 2012.

Cataldo, M. and Herbsleb, J. D. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering (TSE)*, 39(3):343–360, 2012.

Cataldo, M., Wagstrom, P. A., Herbsleb, J. D., and Carley, K. M. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *Proceedings of the 20th Conference on Computer Supported Cooperative Work (CSCW)*, pages 353–362, 2006.

Cataldo, M., Herbsleb, J. D., and Carley, K. M. Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the 2nd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 2–11, 2008.

Choi, K., Yoo, D., Kim, G., and Suh, Y. A hybrid online-product recommendation system: Combining implicit rating-based collaborative filtering and sequential pattern analysis. *Electronic Commerce Research and Applications*, 11(4):309–317, 2012.

Cohen, J. The effect size. *Statistical power analysis for the behavioral sciences*, pages 77–83, 1988.

Constantino, K. and Figueiredo, E. Coopfinder: Finding collaborators based on co–changed files. Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 1–3. IEEE, 2022.

Constantino, K. and Figueiredo, E. Finding collaborations based on co-changed files. In *Anais Estendidos do XVIII Simpósio Brasileiro de Sistemas Colaborativos*, pages 57–66, Porto Alegre, RS, Brasil, 2023. SBC. doi: 10.5753/sbsc_estendido.2023.229735. URL `https://sol.sbc.org.br/index.php/sbsc_estendido/article/view/24226`.

Constantino, K., Zhou, S., Souza, M., Figueiredo, E., and Kästner, C. Understanding collaborative software development: An interview study. In *Proceedings of the 15th International Conference on Global Software Engineering (ICGSE)*, page 55–65, 2020.

Constantino, K., Souza, M., Zhou, S., Figueiredo, E., and Kästner, C. Perceptions of open-source software developers on collaborations: An interview and survey study. *Journal of Software: Evolution and Process*, 33:e2393, 2021.

Constantino, K., Belém, F., and Figueiredo, E. Dual analysis for helping developers to find collaborators based on co-changed files: An empirical study. *Software: Practice and Experience*, pages 1–27, 2023a. doi: https://doi.org/10.1002/spe.3194.

Constantino, K., Prates, R., and Figueiredo, E. Recommending collaborators based on co–changed files: A controlled experiment. In *Anais do XVIII Simpósio Brasileiro de Sistemas Colaborativos*, pages 154–168, Porto Alegre, RS, Brasil, 2023b. SBC. doi: 10.5753/sbsc.2023.229104. URL `https://sol.sbc.org.br/index.php/sbsc/article/view/24191`.

Corbin, J. and Strauss, A. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. 2014.

Costa, C., Figueirêdo, J., Pimentel, J. F., Sarma, A., and Murta, L. Recommending participants for collaborative merge sessions. *IEEE Transactions on Software Engineering (TSE)*, 47(6):1198–1210, 2021.

Creswell, J. W. and Creswell, J. D. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, 2017.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Leveraging transparency. *IEEE Software*, 30(1):37–43, 2012a.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Social soding in github: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW)*, pages 1277–1286, 2012b.

Davis, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.

De Campos, L. M., Fernández-Luna, J. M., Huete, J. F., and Rueda-Morales, M. A. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning*, 51(7):785–799, 2010.

Dey, T., Karnauch, A., and Mockus, A. Representation of developer expertise in open source software. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE)*, pages 995–1007, 2021.

Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. Selecting empirical methods for software engineering research. In *Guide to advanced Empirical Software Engineering*, pages 285–311. 2008.

Eckert, R., Stuermer, M., and Myrach, T. Alone or together? inter-organizational affiliations of open source communities. *Journal of Systems and Software (JSS)*, 149: 250–262, 2019.

Engelfriet, A. Choosing an open source license. *IEEE software*, 27(1):48–49, 2009.

Fang, Y. and Neufeld, D. Understanding sustained participation in open source software projects. *Journal of Management Information Systems (JMIS)*, 25(4):9–50, 2009.

Farias, V., Wiese, I., and Santos, R. What characterizes an influencer in software ecosystems? *IEEE Software*, 36:42–47, 2019.

Ferreira, M., Valente, M. T., and Ferreira, K. A comparison of three algorithms for computing truck factors. In *Proceedings of the 25th International Conference on Program Comprehension (ICPC)*, pages 207–217, 2017.

Fisher, R. A. The arrangement of field experiments. In *Breakthroughs in statistics*, pages 82–91. 1992.

Flick, U. *Designing Qualitative Research*. 2018.

Foucault, M., Palyart, M., Blanc, X., Murphy, G. C., and Falleri, J.-R. Impact of developer turnover on quality in open-source software. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE/ESEC)*, pages 829–841, 2015.

Franco, M. F., Rodrigues, B., and Stiller, B. Mentor: The design and evaluation of a protection services recommender system. In *Proceedings of the 15th International Conference on Network and Service Management (CNSM)*, pages 1–7, 2019.

Gamalielsson, J. and Lundell, B. Sustainability of open source software communities beyond a fork: How and why has the libreoffice project evolved? *Journal of Systems and Software (JSS)*, 89:128–145, 2014.

Ge, M., Delgado-Battenfeld, C., and Jannach, D. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *Proceedings of the 4th Conference on Recommender Systems (RecSys)*, pages 257–260, 2010.

Giuffrida, R. and 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*, 63:11–30, 2015.

Gong, W., Lv, C., Duan, Y., Liu, Z., Khosravi, M. R., Qi, L., and Dou, W. Keywords-driven web apis group recommendation for automatic app service creation process. *Software: Practice and Experience*, 51(11):2337–2354, 2021.

Gousios, G., Pinzger, M., and Deursen, A. v. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 345–355, 2014.

Gousios, G., Zaidman, A., Storey, M.-A., and Deursen, A. v. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, pages 358–368, 2015.

Gousios, G., Storey, M.-A., and Bacchelli, A. Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 285–296, 2016.

Goyal, R., Ferreira, G., Kästner, C., and Herbsleb, J. Identifying unusual commits on github. *Journal of Software: Evolution and Process*, 30(1):e1893, 2018.

Greiler, M., van Deursen, A., and Storey, M.-A. Test confessions: A study of testing practices for plug-in systems. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 244–254, 2012.

Gunawardana, A. and Shani, G. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10(12), 2009.

Herbsleb, J. D. Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering (FOSE)*, pages 188–198, 2007.

Hippel, E. v. and Krogh, G. v. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2):209–223, 2003.

Hu, L., Jian, S., Cao, L., Gu, Z., Chen, Q., and Amirbekyan, A. Hers: Modeling influential contexts with heterogeneous relations for sparse and cold-start recommendation. In *Proceedings of the 33rd Conference on Artificial Intelligence (AAAI)*, volume 33, pages 3830–3837, 2019.

Hu, R. and Pu, P. Using personality information in collaborative filtering for new users. *Recommender Systems and the Social Web*, 17, 2010.

Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P. S., and Zhang, L. Why and how developers fork what from whom in github. *Empirical Software Engineering (EMSE)*, 2017a.

Jiang, J., He, J.-H., and Chen, X.-Y. Coredevrec: Automatic core member recommendation for contribution evaluation. *Journal of Computer Science and Technology (JCST)*, 30(5):998–1016, 2015.

Jiang, J.-Y., Cheng, P.-J., and Wang, W. Open source repository recommendation in social coding. In *Proceedings of the 40th International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1173–1176, 2017b.

Jones, K. S. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 2004.

Jung, K.-Y., Park, D.-H., and Lee, J.-H. Hybrid collaborative filtering and content-based filtering for improved recommender system. In *International Conference Computational Science (ICCS)*, 2004.

Kagdi, H. and Poshyvanyk, D. Who can help me with this change request? In *Proceedings of the 17th International Conference on Program Comprehension (ICPC)*, pages 273–277, 2009.

Kagdi, H., Gethers, M., Poshyvanyk, D., and Hammad, M. Assigning change requests to software developers. *Journal of software: Evolution and Process*, 24(1):3–33, 2012.

Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., and German, D. Open source-style collaborative development practices in commercial projects using github. In *International Conference on Software Engineering (ICSE)*, 2015.

Kenton, J. D. M.-W. C. and Toutanova, L. K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

Kim, H.-N., Alkhaldi, A., El Saddik, A., and Jo, G.-S. Collaborative user modeling with user-generated tags for social recommender systems. *Expert Systems with Applications*, 38(7):8488–8496, 2011.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., and Rosenberg, J. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering (TSE)*, 28(8):721–734, 2002.

Kononenko, O., Baysal, O., and Godfrey, M. W. Code review quality: How developers see it. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 1028–1038, 2016.

Krüger, J., Wiemann, J., Fenske, W., Saake, G., and Leich, T. Do you remember this source code? In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, pages 764–775, 2018.

Lakhani, K. R. and Hippel, E. v. How open source software works:"free" user-to-user assistance. In *Produktentwicklung mit virtuellen Communities*, pages 303–339. 2004.

Lanubile, F., Ebert, C., Prikladnicki, R., and Vizcaíno, A. Collaboration tools for global software engineering. *IEEE Software*, 27(2):52–55, 2010.

Laurent, A. *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software*. 2004.

Lee, A. and Carver, J. C. 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)*, 2017.

Lee, A., Carver, J. C., and Bosu, A. Understanding the impressions, motivations, and barriers of one time code contributors to floss projects: A survey. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*, pages 187–197, 2017.

Li, Q. and Kim, B. M. Clustering approach for hybrid recommender system. In *Proceedings of the International Conference on Web Intelligence (WI)*, pages 33–38, 2003.

Lima, A., Rossi, L., and Musolesi, M. Coding together at scale: Github as a collaborative social network. In *Proceedings of the 8th International Conference on Weblogs and Social Media (ICWSM)*, 2014.

Lin, B., Robles, G., and Serebrenik, A. Developer turnover in global, industrial open source projects: Insights from applying survival analysis. In *Proceedings of the 12th International Conference on Global Software Engineering (ICGSE)*, pages 66–75, 2017.

Linåker, J., Munir, H., Wnuk, K., and Mols, C. Motivating the contributions: An open innovation perspective on what to share as open source software. *Journal of Systems and Software (JSS)*, 2018.

Liu, C., Yang, W., Li, Z., and Yu, Y. Recommending software features to designers: From the perspective of users. *Software: Practice and Experience*, 50(9):1778–1792, 2020.

Marlow, J., Dabbish, L., and Herbsleb, J. Impression formation in online peer production: Activity traces and personal profiles in github. In *Proceedings of the Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*, pages 117–128, 2013.

Matragkas, N., Williams, J. R., Kolovos, D. S., and Paige, R. F. Analysing the 'biodiversity' of open source ecosystems: The github case. In *Proceedings of the 11th International Conference on Mining Software Repositories (MSR)*, pages 356–359, 2014.

McDonald, N. and Goggins, S. Performance and participation in open source software on github. In *Proceedings of the International Conference on Human Factors in Computing Systems (CHI)*, pages 139–144. 2013.

Melville, P., Mooney, R. J., and Nagarajan, R. Content-boosted collaborative filtering for improved recommendations. *Aaai/iaai*, 23:187–192, 2002.

Mendenhall, W., Beaver, R. J., and Beaver, B. M. *Introduction to Probability and Statistics*. Cengage Learning, 2012.

Miller, C., Widder, D. G., Kästner, C., and Vasilescu, B. Why do people give up flossing? a study of contributor disengagement in open source. In *Proceedings of the 15th International Conference on Open Source Systems (OSS)*, pages 116–129, 2019.

Miller, R. and Siegmund, D. Maximally selected chi square statistics. *Biometrics*, pages 1011–1016, 1982.

Minto, S. and Murphy, G. C. Recommending emergent teams. In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR))*, pages 5–13, 2007.

Mockus, A., Fielding, R. T., and Herbsleb, J. A case study of open source software development: The apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, pages 263–272, 2000.

Mooney, R. J. and Roy, L. Content-based book recommending using learning for text categorization. In *Proceedings of the 5th International Conference on Digital Libraries (ICDL)*, pages 195–204, 2000.

Morrison, P., Pandita, R., Murphy-Hill, E., and McLaughlin, A. Veteran developers' contributions and motivations: An open source perspective. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 171–179, 2016.

Morse, J. M. Designing funded qualitative research. 1994.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y. Evolution patterns of open-source software systems and communities. In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE)*, pages 76–85, 2002.

Nembhard, D. A. and Osothsilp, N. An empirical comparison of forgetting models. *IEEE Transactions on Engineering Management (IEEE-TEM)*, 48(3):283–291, 2001.

Oliveira, J., Fernandes, E., Vale, G., and Figueiredo, E. Identification and prioritization of reuse opportunities with jreuse. In *International Conference on Software Reuse (ICSR)*, pages 184–191, 2017.

Oliveira, J., Viggiato, M., and Figueiredo, E. How well do you know this library? mining experts from source code analysis. In *Proceedings of the 18th Brazilian Symposium on Software Quality (SBQS)*, pages 49–58, 2019.

Oliveira, J., Pinheiro, D., and Figueiredo, E. Jexpert: A tool for library expert identification. In *Proceedings of the 34th Brazilian Symposium on Software Engineering (SBES)*, pages 386–392, 2020.

Oliveira, J. A., Viggiato, M., Pinheiro, D., and Figueiredo, E. Mining experts from source code analysis: An empirical evaluation. *Journal of Software Engineering Research and Development (JSERD)*, 9:1–16, 2021.

Oliveira, M. D. R. F. M. F. E., Johnatan; Souza. Can source code analysis indicate programming skills? a survey with developers. In *Proceedings of the 15th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 804–809, 2022.

Onoue, S., Hata, H., and Matsumoto, K.-i. A study of the characteristics of developers' activities in github. In *Proceedings of the 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 2, pages 7–12, 2013.

Pazzani, M. J. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review (AIR)*, 13(5):393–408, 1999.

Pfleeger, S. L. and Kitchenham, B. A. Principles of survey research part 1: Turning lemons into lemonade. *SIGSOFT Software Engineering Notes*, 26(6):16–18, 2001.

Pham, R., Singer, L., Liskin, O., Figueira Filho, F., and Schneider, K. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 112–121, 2013.

Pinto, G., Steinmacher, I., and Gerosa, M. More common than you think: An in-depth study of casual contributors. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 112–123, 2016.

Ponzanelli, L., Scalabrino, S., Bavota, G., Mocci, A., Oliveto, R., Di Penta, M., and Lanza, M. Supporting software developers with a holistic recommender system. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*, pages 94–105, 2017.

Popescul, A., Pennock, D. M., and Lawrence, S. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 437–444, 2001.

Porcel, C., Tejeda-Lorente, A., Martínez, M., and Herrera-Viedma, E. A hybrid recommender system for the selective dissemination of research resources in a technology transfer office. *Information Sciences*, 184(1):1–19, 2012.

Qiu, H. S., Nolte, A., Brown, A., Serebrenik, A., and Vasilescu, B. Going farther together: The impact of social capital on sustained participation in open source. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, pages 688–699, 2019.

Qureshi, I. and Fang, Y. Socialization in open source software projects: A growth mixture modeling approach. *Organizational Research Methods*, 2011.

Rahman, M. M., Riyadh, R. R., Khaled, S. M., Satter, A., and Rahman, M. R. Mmruc3: A recommendation approach of move method refactoring using coupling, cohesion, and contextual similarity to enhance software design. *Software: Practice and Experience*, 48(9):1560–1587, 2018.

Rahman, M. M., Roy, C. K., Redl, J., and Collins, J. A. Correct: Code reviewer recommendation at github for vendasta technologies. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, page 792–797, 2016.

Rastogi, A. and 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)*, 2016.

Rayner, J. and Best, D. Smooth tests of goodness of fit: an overview. *International Statistical Review/Revue Internationale de Statistique*, pages 9–17, 1990.

Rehman, I., Wang, D., Kula, R. G., Ishio, T., and Matsumoto, K. Newcomer oss-candidates: Characterizing contributions of novice developers to github. *Empirical Software Engineering (EMSE*, 27(5):1–20, 2022.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW)*, pages 175–186, 1994.

Ricci, F., Rokach, L., and Shapira, B. *Introduction to Recommender Systems Handbook*. 2011.

Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., and Odenwald, T. Open collaboration within corporations using software forges. *IEEE Software*, 26(2):52–58, 2009.

Robbins, N. B., Heiberger, R. M., et al. Plotting likert and other rating scales. In *Proceedings of the Joint Statistical Meeting (JSM)*, pages 1058–1066, 2011.

Robertson, S. E. and Jones, K. S. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

Robillard, M., Walker, R., and Zimmermann, T. Recommendation systems for software engineering. *IEEE software*, 27(4):80–86, 2009.

Sajedi-Badashian, A. and Stroulia, E. Vocabulary and time based bug-assignment: A recommender system for open-source projects. *Software: Practice and Experience*, 50 (8):1539–1564, 2020.

Salton, G. The smart retrieval system: Experiments in automatic information retrieval, 1971.

Salton, G. Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*, 169, 1989.

Salton, G. and Buckley, C. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

Salton, G. and Harman, D. Information retrieval. In *Encyclopedia of Computer Science*. 2003.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW)*, pages 285–295, 2001.

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. Collaborative filtering recommender systems. In *The Adaptive Web*, pages 291–324. 2007.

Schilling, A., Laumer, S., and Weitzel, T. Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss projects. In *Proceedings of the 45th Hawaii International Conference on System Sciences (HICSS)*, pages 3446–3455, 2012.

Shah, S. K. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7):1000–1014, 2006.

Stănciulescu, Ş., Schulze, S., and Wąsowski, A. Forked and integrated variants in an open-source firmware project. In *Proceding of 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 151–160, 2015.

Steinmacher, I., Gerosa, M. A., and Redmiles, D. Attracting, onboarding, and retaining newcomer developers in open source software projects. In *Workshop on Global Software Development in a CSCW Perspective held in conjunction with the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, 2014.

Steinmacher, I., Conte, T., Gerosa, M. A., and Redmiles, D. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, pages 1379–1392, 2015a.

Steinmacher, I., Silva, M. A. G., Gerosa, M. A., and Redmiles, D. F. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology (IST)*, 59:67–85, 2015b.

Steinmacher, I., Pinto, G., Wiese, I. S., and Gerosa, M. A. Almost there: A study on quasi-contributors in open-source software projects. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, pages 256–266, 2018.

Stol, K.-J., Avgeriou, P., and Ali Babar, M. Identifying architectural patterns used in open source software: Approaches and challenges. In *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, page 91–100, 2010.

Storey, M.-A., Singer, L., Cleary, B., Figueira Filho, F., and Zagalsky, A. The (r) evolution of social media in software engineering. *Future of Software Engineering Proceedings (FOSE)*, pages 100–116, 2014.

Storey, M.-A., Zagalsky, A., Figueira Filho, F., Singer, L., and German, D. M. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering (TSE)*, 43(2):185–204, 2016.

Su, X., Greiner, R., Khoshgoftaar, T. M., and Zhu, X. Hybrid collaborative filtering algorithms using a mixture of experts. In *Proceedings of the International Conference on Web Intelligence (WI)*, pages 645–649, 2007.

Surian, D., Liu, N., Lo, D., Tong, H., Lim, E.-P., and Faloutsos, C. Recommending people in developers collaboration network. In *Proceedings of the 18th Working Conference on Reverse Engineering (WCRE)*, pages 379–388, 2011.

Tamburri, D. A., Kruchten, P., Lago, P., and Van Vliet, H. Social debt in software engineering: Insights from industry. *Journal of Internet Services and Applications*, 6 (1):1–17, 2015.

Terra, R., Valente, M. T., Czarnecki, K., and Bigonha, R. S. A recommendation system for repairing violations detected by static architecture conformance checking. *Software: Practice and Experience*, 45(3):315–342, 2015.

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., and Matsumoto, K.-i. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 141–150, 2015.

Tsay, J., Dabbish, L., and Herbsleb, J. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 356–366, 2014.

Tymchuk, Y., Mocci, A., and Lanza, M. Collaboration in open-source projects: Myth or reality? In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pages 304–307, 2014.

Vale, G., Hunsen, C., Figueiredo, E., and Apel, S. Challenges of resolving merge conflicts: A mining and survey study. *IEEE Transactions on Software Engineering (TSE)*, 2021.

Vasilescu, B., Filkov, V., and Serebrenik, A. Perceptions of diversity on github: A user survey. In *Proceedings of the 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 50–56, 2015a.

Vasilescu, B., Posnett, D., Ray, B., van den Brand, M. G., Serebrenik, A., Devanbu, P., and Filkov, V. Gender and tenure diversity in github teams. In *Proceedings of the 33rd Conference on Human Factors in Computing Systems (CHI)*, pages 3789–3798, 2015b.

Viggiato, M., Oliveira, J., Figueiredo, E., Jamshidi, P., and Kästner, C. Understanding similarities and differences in software development practices across domains. In *International Conference on Global Software Engineering (ICGSE)*, 2019a.

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

Von Krogh, G., Spaeth, S., and Lakhani, K. R. Community, joining, and specialization in open source software innovation: A case study. *Research Policy (RP)*, 32(7):1217–1241, 2003.

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

Whitehead, J. Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE)*, pages 214–225, 2007.

Wilcoxon, F. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. 1992.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.

Wu, Y., Kropczynski, J., Shih, P. C., and Carroll, J. M. Exploring the ecosystem of software developers on github and other platforms. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, pages 265–268, 2014.

Xu, W., Sun, X., Hu, J., and Li, B. Repersp: Recommending personalized software projects on github. In *Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME)*, pages 648–652, 2017.

Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A. E., and Ubayashi, N. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th International Workshop on Principles of Software Evolution (IWPSE)*, pages 46–55, 2015.

Yang, X., Guo, Y., Liu, Y., and Steck, H. A survey of collaborative filtering based social recommender systems. *Computer Communications*, 41:1–10, 2014.

Yu, C. T. and Salton, G. Precision weighting—an effective automatic indexing method. *Journal of Association for Computing Machinery*, 23(1):76–88, 1976.

Yu, Y., Wang, H., Filkov, V., Devanbu, P., and Vasilescu, B. Wait for it: Determinants of pull request evaluation latency on github. In *Proceedings of the 12th International Conference on Mining Software Repositories (MSR)*, pages 367–371, 2015.

Zanjani, M. B., Kagdi, H., and Bird, C. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering*, 42(6):530–543, 2016.

Zhou, M. and Mockus, A. Does the initial environment impact the future of developers? In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 271–280, 2011.

Zhou, M. and Mockus, A. What make long term contributors: Willingness and opportunity in oss community. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 518–528, 2012.

Zhou, M. and Mockus, A. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering (TSE)*, 41(1):82–99, 2014.

Zhou, S., Vasilescu, B., and Kästner, C. What the fork: a study of inefficient and efficient forking practices in social coding. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE/ESEC)*, pages 350–361, 2019.

Appendix A

Interview Documents

In this appendix, we present the email (in Portuguese) used to recruit the participants of the interview, details of the interview process, verbal consent (Portuguese and English), and the interview guides.

A.1 Recruitment email

The email sent to potential answerers was based on the following:

> Prezado <nome do convidado>,
>
> Meu nome é Kattiana Constantino, sou estudante de doutorado do Departamento de Ciências da Computação da Universidade Federal de Minas Gerais (DCC/UFMG).
>
> Gostaria de convidá-lo para participar de uma entrevista para a avaliação de uma abordagem e do suporte ferramental para identificar colaborações e também, minimizar as contribuições não mescladas em projeto de desenvolvimento de software baseado em forks. Esse projeto de pesquisa é uma parceria com o *Institute for Software Research* da *Carnegie Mellon University (ISR/CMU)*. Considerando os desenvolvedores no GitHub, você foi escolhido por ter mais de <número de contribuições> contribuições nos últimos 2 anos, como também por ter feito forks e contribuído em alguns deles. Atualmente, você é um dos mantenedores do projeto <nome do projeto>. Essas informações foram coletadas do seu perfil no GitHub (https://github.com/<user no github>). Abaixo segue mais detalhes sobre o projeto de pesquisa e demais informações. Por favor, responda esse e-mail me informando seu interesse em participar.
> Desde já agradeço, *Kattiana Constantino (kattiana@dcc.ufmg.br ou kattiana@gmail.com)*
> *Departamento de Ciências da Computação/Universidade Federal de Minas Gerais*

A.2 Recruitment email - Details of the research

Avaliação da Abordagem e Suporte Ferramental para Identificar Colaborações e Minimizar as Contribuições Não Mescladas em Projeto de Desenvolvimento de Software Baseado em Forks.

Estamos realizando um estudo de entrevistas com desenvolvedores de software que contribuem para repositórios de código aberto baseados em forks, para aprender mais sobre as questões levantadas, minimizar o desperdício de recursos de software e esforços humanos e também entendermos como a nossa abordagem e o suporte ferramental pode ajudar em nessas questões.

Elegibilidade: Você deve ter 18 anos ou mais e ter contribuído para um projeto de código aberto por meio de uma estratégia baseada em forks.

Procedimento: Se você decidir participar, definiremos um horário para uma entrevista de acordo a sua conveniência. A entrevista pode ser presencial ou por Skype e tem a duração de 30 a 60 minutos. O áudio da entrevista será gravado para posteriormente fazermos uma transcrição mais precisa. Além disso, como trata-se da avaliação de uma ferramenta web, o recurso de compartilhamento de tela do próprio Skype será utilizado para gravar a sua interação com a ferramenta. O vídeo será usado para posteriormente contextualizarmos melhor as suas respostas.

Queremos enfatizar que a sua identidade e a sua participação nesta pesquisa serão mantidas em sigilo e os dados divulgados pela pesquisa não conterão nomes ou quaisquer outras informações que permitam identificá-lo(a). Os arquivos contendo as gravações e transcrições da entrevista não serão acessados por outras pessoas do departamento ou da universidade, além de mim, meu orientador e demais pesquisadores colaboradores desse projeto de pesquisa. Você não terá nenhum gasto com a sua participação no estudo e também não receberá pagamento ou indenização pelo mesmo.

Se você quiser participar ou tiver alguma dúvida sobre o estudo, envie um e-mail para kattiana@dcc.ufmg.br ou kattiana@gmail.com.

A.3 Consentimento Verbal [Portuguese]

Obrigado por aceitar participar deste estudo de pesquisa para avaliar a abordagem e suporte ferramental para identificar colaborações e minimizar as contribuições não mescladas em projeto de desenvolvimento de software baseado em bifurcação. Do nosso lado, Kattiana Constantino e Eduardo Figueiredo estão na reunião.

O objetivo desta reunião/entrevista é entender melhor como os desenvolvedores buscam por colaborações, quais os tipos de colaborações são mais interessantes, como eles lidam com as contribuições não mescladas e como a ferramenta é útil para suportar essas questões e demandas. Gostaríamos de conversar com você sobre suas experiências em buscar colaborações e minimizar as contribuições não mescladas no contexto de desenvolvimento baseado em bifurcação.

A entrevista não deve durar mais que 60 minutos; Sua participação é voluntário e você tem o direito de desistir a qualquer momento. Nós também precisamos que voce responda se tem mais de 18 anos? <Esperar pela resposta>.

Além disso, gostaríamos de gravar o áudio da conversa. Na transcrição do áudio, removeremos as informações que identifiquem você ou sua empresa antes de começarmos as análises. Pedimos, ainda, que durante a entrevista você evite qualquer informação privada sobre outras pessoas e informações confidenciais. Você concorda em prosseguir com a entrevista?

A.4 Verbal Consent [English]

Thanks for agreeing to participate in our research study to evaluate our approach and tooling support to identify collaborations and minimize the non-merged contributions within fork-based development. On our side only Kattiana Constantino is in the meeting.

The purpose of this meeting/interview is to better understand how participants look for collaborations, what kinds of collaborations are interesting, how they deal with non-merged contributions, and how the tool is useful to support those issues and demands. We would like to talk to you about your experiences in seeking collaborations and minimizing non-merged contributions within fork-based development context.

This should not last longer than 60 minutes; it is voluntary, and you have the right to withdraw at any time. I also need to ask if you are 18 or older? <wait for response>.

We would like to audio-record this meeting, but we will transcribe and remove identifying information before we analyze it further. Personal identifiers (your name, organization, and phone number) are stored separate from interview transcripts and are only linked by a code. During the interview, please do not share any private, identifiable information about other individuals or disclose confidential proprietary information. Do you agree with this procedure?

A.5 Interview Protocol [English]

A.5.1 Part 1

Objective: To understand how the participant is familiar with the OSS project hosted on GitHub.

1. What are the main OSS projects hosted on GitHub that you are involved in? What is your role in each of them (project owner/maintainer/developer)?
2. Have you recently been involved in OSS project development on GitHub?

3. In the OSS project hosted on GitHub, which you are most involved in, do you prefer to work alone or with other developers? Why?

A.5.2 Part 2

Objective: To understand participants experiences of collaborations in forks-based development.

1. 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?
2. Do you usually open some communication channel with any of the project developers <project name> for more collaborations?
3. In the <project name> project, what kind of collaboration are the most common among developers?
4. Are there other collaborations that are important, but little explored in the <project name> project that you participate in?

Task 1: Let suppose you want to find a collaborator. Looking for a developer or fork in the <project name> project that is most similar to your fork.

1. What is the name of the fork chosen?
2. Detail the strategy adopted to find this fork.
3. How easy is to find this information? Why?

Task 2: Now, let suppose you have two forks (collaborators), but you want to choose the most similar of them with your work(the interview presents two recommended developer for interviewee (top 1 and 5)).

1. What these two forks are more similar with your work? Why?
2. Based on the similar of these forks with your work, how likely would you collaborate with these developers? Why?
3. How easy is to make this decision? Why?

Appendix B

Survey Documents

In this appendix, we present the email used to recruit the participants to survey studies, consent term, and the opinion survey. We applied these documents for two studies (Chapter 4 and Chapter 6).

B.1 Recruitment email

Dear <Developer name>,

You are invited to participate in our scientific research to encourage developers to collaborate with each other into the project community (main repository or forks).

We have explored the commits from the <projec name>project in which you are a contributor with at least 5 commits. And, we have identified some forks of the project that share similar commits to yours.

Thus, we would like to hear your thoughts on the similarity of this developer's works below, with yours. And, how openness you are for working in partnership with him/her into the project community.

Your Fork: <fork of the developer>

Fork: <fork of the recommended developer >

Examples of relevant files for you both are:

1. <change file path> (e.g., commit <link for the commit>)
2. <change file path> (e.g., commit <link for the commit>)
3. <change file path> (e.g., commit <link for the commit>), or others.

The one-page survey should only take a few minutes. All your personal details will be kept confidential. Moreover, we will release data in aggregated format only, without any personally identifiable information. By answering this survey, you agree to provide the data for the propose of our scientific research.

Our Survey <link for the opinion survey>)

I would greatly appreciate your contribution to this matter – future support for working in partnership among developers in the open-source community could certainly benefit from your comments.

Yours sincerely,
Kattiana Constantino, Ph.D. student (kattiana@dcc.ufmg.br)
Federal University of Minas Gerais - Brazil

B.2 Opinion survey

We have explored the commits from a project in which you are a contributor. We have identified some contributing forks which share similar changes as yours.

We would like to invite you to share your thoughts on a specific fork we have listed in the email we sent you. Thank you for your time to complete this survey. Every answer is important to me.

Kattiana Constantino (Federal University of Minas Gerais - Brazil)

Table B.1: 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 am interested in some of these changed files in this fork [ ] Strongly Disagree, [ ] Disagree, [ ] Neither Agree or Disagree, [ ] Agree, and [ ] Strongly Agree
SQ4	I am familiar with some fork changes in these files [ ] Strongly Disagree, [ ] Disagree, [ ] Neither Agree or Disagree, [ ] Agree, and [ ] Strongly Agree
SQ5	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)
SQ6	Are there other active forks in this project that you know about that you would consider to be of your interest? Which are they? (open question)
SQ7	Other important observations or suggestions (open question)

Appendix C

User Evaluation Documents

In this appendix, we present the instruments used to perform the user evaluation, i.e., the contexts and tasks for each tool. Moreover, we present the questionnaire applied before and after the evaluation. The post-assignment questionnaire also brings the open questions used to collect feedback from participants regarding the recommendation strategies and the prototype. We performed the following steps before starting the experiment:

Project presentation: Presenting the consent form to the participant, explaining the purpose of the project and the evaluation. Besides, explaining briefly some GitHub basics, e.g., fork-based development and commits/pull-requests.

Participant profile: Collecting basic information about the profile of participants, such as gender and education. In addition to identifying the participant's knowledge of software development, GitHub, and the frequency they are carried out.

C.1 Free and enlightened consent

Title: A Recommendation Framework to Support Collaborations - Evaluation of the CoopFinder Prototype-Tool

Date: April/2022
Institution: DCC/ICEx/UFMG

Responsible researchers:
Researcher's name: Kattiana Constantino (kattiana@dcc.ufmg.br)
Function: Doctoral student at the Department of Computer Science at UFMG

Professor Eduardo Figueiredo (figueiredo@dcc.ufmg.br)
Function: Advisor and professor at the Department of Computer Science at UFMG

Professor Raquel O. Prates (rprates@dcc.ufmg.br)
Function: Professor at the Department of Computer Science at UFMG

Introduction: This "Free and Enlightened Consent" contains information about the survey indicated above. To ensure that you are informed about your participation in this research, we ask that you read this document. If you have any questions, do not hesitate to ask the responsible researcher.

Evaluation goals: We aim evaluate the collaborator recommendations using the CoopFinder prototype-tool. This Web tool supports project contributors to find suitable collaborators based on similar interests. This research study investigates how data visualization, interaction, and analysis can help developers find appropriate collaborators based on their changed files.

Survey information: We will ask for you to perform some simple tasks using the CoopFinder web tool and GitHub for comparison purposes. We can record the performance of these tasks for later analysis by the researchers. In addition, during this phase, we can ask some questions (through interviews and/or questionnaires) about your perceptions and experience using the tools.

Data collection and use: We will use the data collected during the evaluation for the research study and maintenance of CoopFinder. We ensure the anonymity of participants for any data used for publication. The activities carried out may be recorded in audio and/or video. All information collected is confidential, and only the research team will have access to it. Confidentiality also applies to the identity of the participants. No one (apart from the research team) will know who the participants are, as they will not have access to the data, and we will not tell other people. We will store the data on the researcher's computer for a maximum legal period of 05 years. Besides, we will analyze the data focusing on the proposed methodology and activities. We will use the research results in scientific work published or orally presented at conferences or lectures. In this case, we will remain anonymous in submitting any data relevant to the submission publi-

cation. We guarantee that we will not reveal your identity at any time. If your wish, you can request a copy of the data generated by you.

Risks and benefits: The risks involved in the research consist of fatigue or embarrassment caused by participating in the proposed evaluation (through interviews/questionnaires). If this happens, the assessment will stop immediately. We will only continue if you feel up to it again and agree to continue. In case of damages resulting from participation in the research, we will indemnify you. Regarding the benefits of your participation in the study, you will be able to reflect on how to support and strengthen the ties between project collaborators. In addition, you will be able to share your experiences and perceptions about the points that you consider relevant, which can contribute to the project and the customization and improvement of the evaluated tool. If you decide not to participate in the research: You are free to decide, at any time, whether or not to participate in this research. Your decision will not affect any relationship with the evaluators, researchers, teachers, or the Institution behind it.

Compensation: Participation in this research is voluntary, and we will not offer any compensation to participants. You can contact the researchers at any time in the emails mentioned earlier if you have any problems or any other questions. For example, if you have any problems that you think may be related to your participation in this research, or if you have any questions about the research, please, contact them.

New conditions: If you wish, you can specify new conditions in order for you to participate in this assessment.

Research ethics committee: This study was approved by the Research Ethics Committee of the Federal University of Minas Gerais. If you have questions about the ethical aspects of the research, you can contact COEP.

Research Ethics Committee of the Federal University of Minas Gerais (COEP-UFMG).
Av. Antônio Carlos, 6627. Administrative Unit II – 2nd floor – Room 2005. Pampulha Campus. Belo Horizonte, MG – Brazil. CEP: 31270–901.
E–mail: coep@prpq.ufmg.br. phone: (31)3409–4592.

This consent term is printed in two original copies: The responsible researcher will keep one, and the other will be with you. The researchers will treat your name in secret, complying with Brazilian legislation (Resolution No. 466/12 of the National Health Council). If the evaluation is done remotely due to the Covid–19 pandemic, the participant consent term can be sent and answered by email OR consented in the Google

Docs form. The evaluation data will only be used if the participant gives explicit authorization. If you do not wish to give authorization for the use of the data, please choose the option "No, the data must not be used in any scientific research." on the first page of the "Participant Characterization" form.

Free and informed consent (voluntary agreement): The document mentioned above describing the benefits, risks, and procedures of the research entitled "A Recommendation Framework to Support Collaborations - Evaluation of the CoopFinder Prototype-Tool" was read and explained. I had the opportunity to ask questions about the survey (or email them), answered to my satisfaction. Therefore, I agree to participate as a volunteer.

Participant's signature: _____
Name of participant: _____
Signature of the researcher: _____
Name of researcher: _____
Place: _____
Date: _____
Document printed in two copies (Participant/Researcher).

C.2 Pre-assignment questionnaire

Table C.1: Pre-assignment Questionnaire - Part 1.

	Topics covered
Personal data	Participant ID: _____
	Formation: _____
	University (and period) or company: _____
	Course: _____
	Gender: [Female | Male | I prefer do not inform | Other]
Previous knowledge	How do you rate your knowledge of technical English reading? [Basic| Intermediate| Advanced]
	Do you have work experience (including internship) in the field of software development? [I have no work experience I have experience of up to 1 year I have experience from 1 year to 3 years I have more than 3 years experience]
Previous knowledge	How do you rate your knowledge in relation to the following topics? Object Oriented Programming Software Modeling (UML) Automated Software Testing Version Control System (example, GitHub) Software reuse (example, library, framework, etc.) Software Measurement [None| little| Moderate| Expert]
	Describe any other observations that you think are important about your training and professional experience in software development. (optional answer)

Table C.2: Pre-assignment Questionnaire - Part 2.

Physical condition	Do you have any vision problems? [astigmatism| color blindness| myopia| others] (Optional answer)
OSS collaboration on GitHub	Are you a project contributor on GitHub? [Yes| No]
	[If so], Are you or have you been a maintainer of GitHub project(s)? [Yes| No]
	How many years of experience do you have as a GitHub project(s) contributor? [I have no work experience I have experience of up to 1 year I have experience from 1 year to 3 years I have more than 3 years experience]
	How often do you contribute to GitHub project(s)? [Every day A few times a week A few times a month A few times a year Rarely]
	[If no], Have you tried contributing to any projects on GitHub? [Yes| No]
	What situations did you face? (optional answer)

C.3 Context and tasks using GitHub

Context: You are part of the core team of a project hosted on GitHub, you know the demands of the project and also the importance of collaborations made by volunteer contributors with different motivations to collaborate. For example, some prefer to work collaboratively to increase the quality of the project, while others contribute to the project out of personal interests. In addition, the project attracts several potential volunteer collaborators. However, due to the large number of them, it is difficult to interact or follow the collaboration of all, running the risk of demotivating them and, consequently, leaving the project. Therefore, you consider and understand the importance that contributors should approach and interact more with each other in this development environment. You are also aware that some contributors may have difficulty expressing their opinions publicly in this environment. On the other hand, they may feel more secure when interacting with other specific contributors, especially those with the same interests or familiarity with certain parts of the project. They may be already known contributors and those who have some reference or recommendation. **Note**: If you feel it necessary when performing the task and answering the question, report any information to understand the reason for your answer.

Table C.3: Context and Tasks Using GitHub - Part 1.

ID	Task description
Task 1	You are part of the core team of projects that are hosted on GitHub. You are using GitHub to review projects. Thus, access the tool and answer: How many stars does the project <project name> have? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]

Table C.4: Context and Tasks Using GitHub - Part 2.

ID	Task description
Task 2	Review the contributors to the <project name> project. Thus, as part of the core team of the <project name> project, you want to know about the contributors who are collaborating on this project. Therefore, you search for contributors based on the number of commits updated in the project (merged commits). Thus, you have noticed that a given contributor has 28 commits updated and 6 commits not yet updated (commits ahead) in the main project. What is the name of this contributor? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]
Task 3	You realize the risk of losing those contributors who put some effort and time into the project. Thus, now you want to know who they are, thus look for the contributors by the number of commits not yet updated (commits ahead) in the project and answer. Which contributor has the most commits not yet up–dated in the main project? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]
Task 4	You have a high-priority open issue and would like to assemble a dedicated team to solve it. For that, you are analyzing the list of contributors and based on the number of commits, you think that the contributor <collaborator name (fork name)> could be the one you are looking for. <collaborator name> has 23 commits updated in the main project, plus its last commit was on Aug20. Concerned about the lack of engagement in the project, you want to find other contributors who have similar interests as the contributor <collaborator name> (i.e., they are interested in the same parts, features, or source files of the project). How many other collaborators have interests in common with <collaborator name>? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]

Table C.5: Context and Tasks Using GitHub - Part 3.

ID	Task description
Task 5	Comparing the contributions of <collaborator name> and <another collaborator name> (<fork name>), which files are relevant and common to them?
	Answer: _____
	This task was... [Very Difficult| hard | Easy | Very easy]
Task 6	Considering the changed lines of code in the files that correspond to common interests among the project's contributors, which other contributors can be considered to work with <collaborator name>?
	Answer: _____
	This task was... [Very Difficult| hard | Easy | Very easy]
Task 7	What is the expertise related to the programming languages of the contributor <another collaborator name>?
	Answer: _____
	This task was... [Very Difficult| hard | Easy | Very easy]
	Would you like to make any observations or comments? Any questions or suggestions?

C.4 Context and tasks using CoopFinder

Context: You are part of the core team of a project hosted on GitHub. You know the demands of the project and the importance of collaborations made by volunteer contributors with different motivations to collaborate. For example, some prefer to work collaboratively to increase the quality of the project, while others contribute to the project out of personal interests. In addition, the project attracts several potential volunteer collaborators. However, due to the large number of them, it is difficult to interact or follow the collaboration of all, running the risk of demotivating them and, consequently, leaving the project. Therefore, you consider and understand the importance that contributors should approach and interact more with each other in this development environment. You are also aware that some contributors may have difficulty expressing their opinions publicly in this environment. On the other hand, they may feel more secure when interacting with other specific contributors, especially those with the same interests or familiarity with certain parts of the project. They may be already known contributors and those who have some reference or recommendation. **Note**: The following tasks must be performed exclusively using the CoopFinder tool. If you feel it necessary when performing the task and answering the question, report any information to understand the reason for your answers.

Table C.6: Context and Tasks Using CoopFinder - Part 1.

ID	Task description
Task 1	You are part of the core team of 4 GitHub projects. You are using the CoopFinder tool to analyze these projects. Thus, using the tool, order the projects based on the creation date, and answer: Which project is the most recent? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]

Table C.7: Context and Tasks Using CoopFinder - Part 2.

Task 2	Choose the <project name> project to review the contributors. As part of the core team of the <project name> project, you want to know about the contributors contributing to the project. Thus, you have sorted the list of contributors by the number of commits updated in the project (merged commits). Thus, you have noticed that a given contributor has 50 updated commits and 24 non-merged commits in the main project. What is the name of this contributor? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]
Task 3	Now you have realized that some contributors contribute to the project. However, they do not constantly update all of their contributions to the main project for some reason. Thus, you realize the risk of losing those contributors who put some effort and time into the project. Thus, now you want to know who they are. Thus, you can sort the contributors by the number of commits not yet updated (non-merged commits) in the project and answer. Which contributor has the most commits not yet updated in the main project? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]
Task 4	You have a high–priority open issue and would like to form a dedicated team to solve it. You are analyzing the list of contributors based on the number of commits. You think that the contributor <collaborator name> could be what you are looking for. Thus, <collaborator name> has 13 commits updated in the main project and 33 not, and his last commit was in Mar/21. Concerned about the lack of engagement in the project, you want to find other contributors who have similar interests as a contributor <collaborator name>. The CoopFinder tool can help you find them. How many other contributors did the CoopFinder tool recommend to work with <collaborator name>? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]

Table C.8: Context and Tasks Using CoopFinder - Part 3.

Task 5	Comparing the contributions of <collaborator name> and <recommended collaborator name>, which files are relevant and common to them? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]
Task 6	Based on the "Changed LoC" strategy (changed lines of code), which different contributors were recommended to work with <collaborator name>? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]
Task 7	Still based on the "Changed LoC" strategy, what is the expertise related to the programming languages of the recommended contributor <recommended collaborator name>? Answer: _____ This task was... [Very Difficult| hard | Easy | Very easy]

C.5 Post-assignment questionnaire

Table C.9: Post–assignment questionnaire.

ID	Open–question
Q1	What did you think about the CoopFinder tool?
Q2	What are the strengths of the CoopFinder tool?
Q3	What are the points to improve in this tool?
Q4	What other technical or social information do you think could be explored to improve collaborator recommendation algorithms?
Q5	Would you use and/or recommend this tool? Why?
Q6	Would you like to make any observations or comments? Any questions or suggestions?