Journal on Interactive Systems, 2021, 6:1, doi: 10.5753/jis.2021.xxx

User Evaluation of a Collaborator Recommender based on Co-Changed Files

Kattiana Constantino [**Federal University of Minas Gerais** | ***kattiana@dcc.ufmg.br***]
Raquel Prates [**Federal University of Minas Gerais** | ***rprates@dcc.ufmg.br***]
Eduardo Figueiredo [**Federal University of Minas Gerais** | ***figueiredo@dcc.ufmg.br***]

Abstract
Active collaboration is essential for the success of software projects across the 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. To this end, we implemented developer recommendation strategies and prototype tool to help project contributors improve their collaborations. Thus, in this work, we described a controlled experimental study concerned usability and user satisfaction 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. We asked participants to perform the experiment tasks to find collaborators with similar interests using a prototype recommendation tool and GitHub. We reported a quantitative and qualitative evaluation of strategies and tool using the state of the practice as a baseline. As a result, we observed that recommender based on co–changed files can provide suitable collaborator recommendations to developers of a specific project. About 66% of the participants confirmed they would use or recommend this tool.

Keywords: *Open-Source Software Projects, Collaborative Software Development, Distributed Collaboration, Developer Recommendation.*

1 Introduction

Active collaboration is essential for the success of software projects across the development life-cycle. Contributors who appreciate the work or feel responsible for the project are more likely to persist than those driven by particular interests (Shah, 2006; Crowston and Fagnot, 2018). Thus, it would be useful to support contributors to stay in the project and make quality contributions (Qiu et al., 2019; Crowston and Fagnot, 2018; Barcomb et al., 2019, 2020). However, social coding platforms, like GitHub[1], can present challenges in finding potential collaborators with whom they could create new/stronger ties and enhance the quality of contributions. One of the challenges associated with identifying collaborators is that reliable information is often not readily available (Minto and Murphy, 2007; Surian et al., 2011; Canfora et al., 2012).

Previous works have explored developer recommendations for collaborative interactions in software engineering development. For instance, Minto and Murphy (2007) ranked a list of the likely emergent team members based on a set of files of interest. 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 before. Canfora et al. (2012) identify and recommend mentors for newcomers in software projects by mining data from mailing lists and versioning systems. Finally, Thongtanunam et al. (2015) recommended pull-request reviewers based on past reviews of files with similar names and paths. Most of these studies mainly focus on specific software tasks and limit the recommended candidates to the core developers of the projects. Inspired by these previous work (Canfora et al., 2012; Thongtanunam et al., 2015), we recommend collaborators based on a set of files that have been mutually edited to increase engagement in the project and enhance the opportunities for collaborations, not only core members, code-reviewers, or mentors but also all active collaborators of the project that need some help. In this work, we have denominated these mutually edited files as co-changed files (Constantino et al., 2023a).

In previous work, we presented a prototype–tool named CoopFinder[2] (Constantino and Figueiredo, 2022), which supports two developer recommendation strategies. In another previous work (Constantino et al., 2023a), we evaluated these developer recommendation strategies based on co–changed files from the point of view of who receives the recommendations. We observed that these strategies helped developers and maintainers find opportunities for collaborations.

To support the strategies, CoopFinder is an interactive visual tool allowing developers to select collaborators and see in which part of the project they have similar interests. The interactive ability of the tool allows developers or maintainers to follow the activities of the collaborator in order to identify potentially interesting collaborators. Based on previous works (Constantino et al., 2020, 2021), we considered that the set of files of interest represent strong ties in connecting developers of a project. That is, coding tasks may point to opportunities for joint contributions to the project.

In this paper, we extend our prior research (Constantino et al., 2023b), which describes a controlled experimental study[3] to investigate the developers' perceptions of using

[1] https://github.com/

[2] https://homepages.dcc.ufmg.br/ kattiana/coopfinder/welcome.html

[3] As required, the study was approved by the University's (UFMG)

CoopFinder prototype. The study involved 35 participants, of which 18 were GitHub users and 17 were non-users. 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 out a background questionnaire before the experiment and execute a set of tasks. 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 (random) participants. Afterward, participants answered a post–assignment questionnaire about their opinions on the developer recommendations.

As results, participants pointed out that CoopFinder is easy to use, intuitive, exciting, and supports project maintainers. Besides, we observed that participants were able to perform tasks more easily using CoopFinder than GitHub. About 66% of the participants confirmed they would use or recommend this tool. Our primary contributions can be succinctly summarized as follows:

- We propose developer recommendation strategies, supported by a visual and interactive tool to connect collaborators based on a set of files of their interest. Furthermore, the tool provides metadata and links different attributes that could not be analyzed using the GitHub interface;
- We describe a quantitative and qualitative evaluation of strategies and tool using the state of the practice as a baseline;
- We designed and conducted a controlled experiment to evaluate the developer recommendation strategies and tool;
- We evaluated the usability of CoopFinder with 35 developers. About 51% of them are collaborators and maintainers of real-world open-source projects hosted on GitHub;
- We obtained insights from the users to improve the developer recommendation algorithms and the supporting tool.

Our comprehensive replication package is readily accessible online to facilitate future replications and extensions[4]. The structure of this paper unfolds as follows. Section 2 introduces the problem we address. Section 3 offers insight into the developer recommendation strategies, encompassing their design, implementation, and practical usage within the Coopfinder tool. Section 4 describes the study design. Furthermore, we analyze and report the results of this study (Section 5). Section 6 explores potential threats to the validity of our study. Finally, we end this paper with some concluding remarks and discuss directions for further work (Section 7).

2 Problem Statement

Previous works show that developers usually ask for help from the core team members, who should be expected to share their motivation, knowledge, and experience (Minto and Murphy, 2007; Kononenko et al., 2016). However, this may not always work as the core team members could be too busy to respond (Yu et al., 2015; Gousios et al., 2015; Steinmacher et al., 2018). Other experienced developers outside of the core team could also be helpful, and might be more available. That is, all collaboration is essential for the project to succeed (Gamalielsson and Lundell, 2014). Hence, all contributions should be appreciated and encouraged (Pham et al., 2013; Gousios et al., 2014; Pinto et al., 2016).

Previous research also mentions that not having enough people to perform core team roles, such as maintainers, supporters, and reviewers, impacts the sustainability of the project (Jiang et al., 2015; Costa et al., 2021). Developer turnover can also have a negative impact, as a small group of developers may become overloaded with information and knowledge (Avelino et al., 2016; Ferreira et al., 2017), while others may have limited access to knowledge-sharing opportunities (e.g., collaborations, discussions) (Tamburri et al., 2015). These situations can lead to frustration and encourage developers to leave the project. These issues all relate to how developers interact with each other and how these relationships affect the project. Therefore, it is crucial to optimize collaboration among project developers and maintain a balanced team.

3 Developer Recommendation Strategies

Developer Recommendation Strategies are supported by CoopFinder, a prototype tool that enhances opportunities for collaboration in a project based on co-changed files. These co-changed files refer to files that two or more developers have modified. CoopFinder is an interactive and visually-rich web application that helps connecting developers of these files.

3.1 Developer Recommendation Design

CoopFinder implements two developer recommendation strategies, namely Strategies 1 and 2, which are based on co–changed files. For Strategy 1, the *number of commits* was mined to determine the frequency of file modifications by a developer. For Strategy 2, the *number of changed lines of code (LoC)* was extracted. This metric computes the sum of code lines added and removed by a developer in a specific file (Constantino et al., 2023a). Figure 1 presents an overview of the steps required to recommend a developer to another developer in the software project, as follows.

Step 1 – Feature Extraction: The modification history made by all developers in a project was extracted concerning the inputs, as illustrated in Figure 1. The GitHub platform, a social network hosting projects and supporting the fork & pull model, was utilized for this purpose. GitHub developers create copies of the original repository and make changes in their respective copies. Once these changes are finalized, they have the option to submit them back into the original repository via a pull request. Thus, we want to assess the similarity of interest among developers in a project. For instance, when both Developer 1 and Developer 2 modify File$_A$, we infer that they share an common interest in File$_A$. Conversely, if

Committee for Ethics in Research - Protocol number: 55476922.0.0000.5149.
[4]https://github.com/kattiana/coopfinder

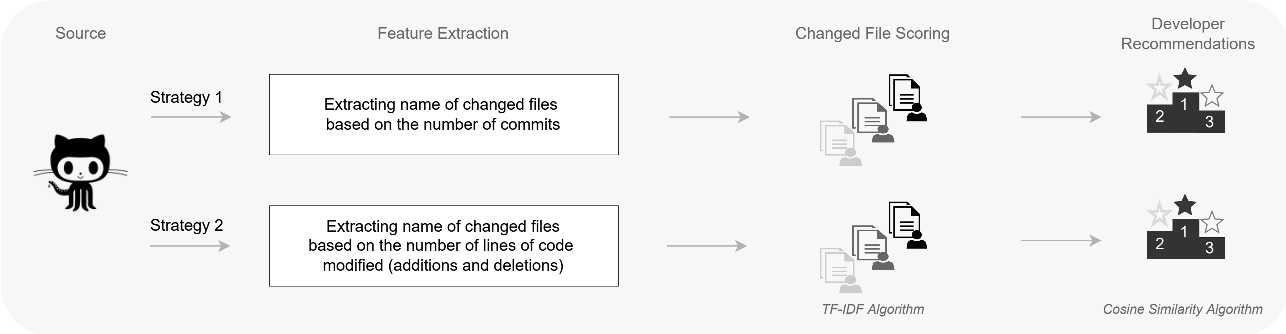

Figure 1. Two developer recommendation strategies (Constantino et al., 2023a).

only Developer 3 modifies File_F, it implies that Developer 3 is exclusively interested in File_F. There may also be instances where all developers express interest in File_C if each contributes to its modification. We employ the following metrics to capture these changes: *Number of commits*, utilized in Strategy 1, quantifies how frequently a developer modifies a file. On the other hand, *Number of changed lines of code (LoC)*, employed in Strategy 2, computes the total count of added and removed code lines in a specific file where the developer is involved. Both aforementioned metrics are calculated considering the whole life time of the project.

Step 2 – Changed File Scoring: We performed file mining for each developer of the project by extracting the set of co–changed files. This set of files was then ranked using the Term Frequency – Inverse Document Frequency (TF–IDF) algorithm (Salton, 1989). The resulting rank of relevant files for each developer is presented in Figure 1. For instance, considering Strategy 1 for Developer 2, File_B is more relevant than File_A. However, in Strategy 2, the rank of files changed, then File_A became more relevant then File_B.

Step 3 – Developer Recommender Model: The rank of relevant files for each developer of the project, calculated using the vector space model, was utilized to calculate their similarity via the widely-used cosine metric (Salton, 1971; Salton and Harman, 2003). This metric has been extensively employed (Rahman et al., 2016; Franco et al., 2019) due to its ability to quantify the similarity of two objects (Ricci et al., 2011). In summary, these developer recommendation strategies primarily center around contributions based on co-changed files. Strategy 1 relies on the number of commits in a code file, but this metric poses a potential drawback. It tends to favor a developer who frequently makes small commits, suggesting "higher engagement" with the file, over another developer who makes infrequent but substantial commits. To mitigate this issue, Strategy 2 adopts the "LOC metric", accounting for added or deleted code lines in a project file. This metric allows us to capture the volume of changes, providing insights into 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.

3.2 Implementation Technologies

Our web tool is based on client-server architecture and utilizes visualization techniques. The server-side is implemented in Python 3[5], with the help of the scikit-learn libraries[6], a free machine learning library for Python. For the views in CoopFinder, we employed the HighCharts[7] analytical data visualization components, a JavaScript library that enables the manipulation of documents based on data. Finally, we utilized the Bootstrap Framework[8] components, which include various stylesheets and jQuery plugins[9], to create an interactive user interface. Our choice of these technologies was driven by the goal of providing a dynamic exploration and visualization experience.

3.3 Interface and Interaction

The screenshots of CoopFinder related to the list of contributors of a selected project are depicted in Figure 2. This list includes all contributors who have modified any files in their copies from a selected project from GitHub, as described in Section 3.1. In Figure 2, Frame (A) displays project information, such as the repository name, number of stars, number of forks, and number of open issues, to which the collaborators belong. Frame (B) presents a table of all collaborators of the selected project. For each collaborator, the table provides their developer information, including their avatar, name, fork, number of followers, number of following, number of commits in upstream, number of non-merged commits, and the date of their last commit. Frame (C) displays code activity for upstream and non-merged commits, along with the last commit date. This helps users assess the status of the collaborators in the project, including their activity level based on merged commits and last commit date. Recent non-merged commits may signal a need for assistance. Furthermore, maintainers can review the interests of the collaborators in the project or build teams around of their co-changed files. Finally, the button *"Run"* runs the algorithms of the recommendations and the results are presented as following.

Figure 3 depicts a screenshot of CoopFinder with a list of recommended collaborators for the target developer, which may vary depending on the selected strategy and the rank of relevant files for each project developer, as described in Section 3.1. Frame (D) displays project information, such as the repository name, number of stars, number of forks, and number of open issues, to which the recommended collaborators

[5]https://www.python.org/
[6]https://scikit-learn.org/stable/index.html
[7]https://www.highcharts.com/
[8]https://getbootstrap.com/
[9]https://jquery.com

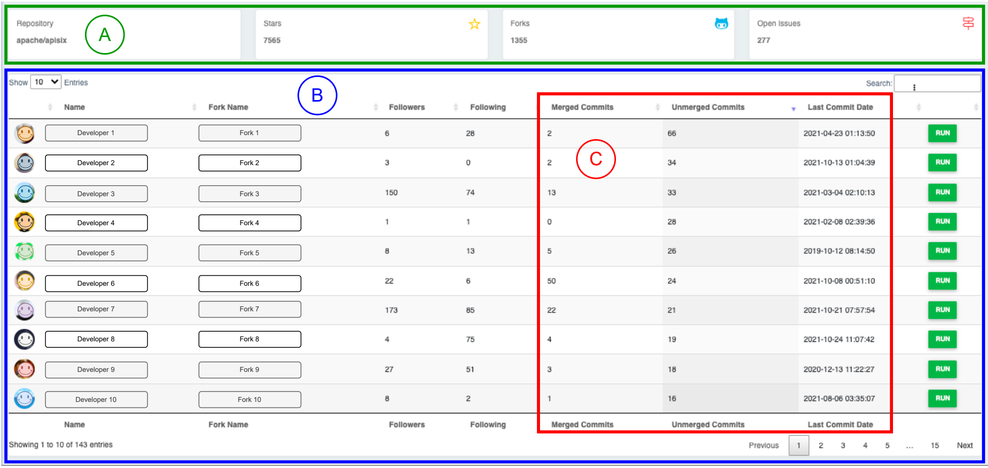

Figure 2. Overview of the contributors information from a specific GitHub project (Constantino and Figueiredo, 2022).

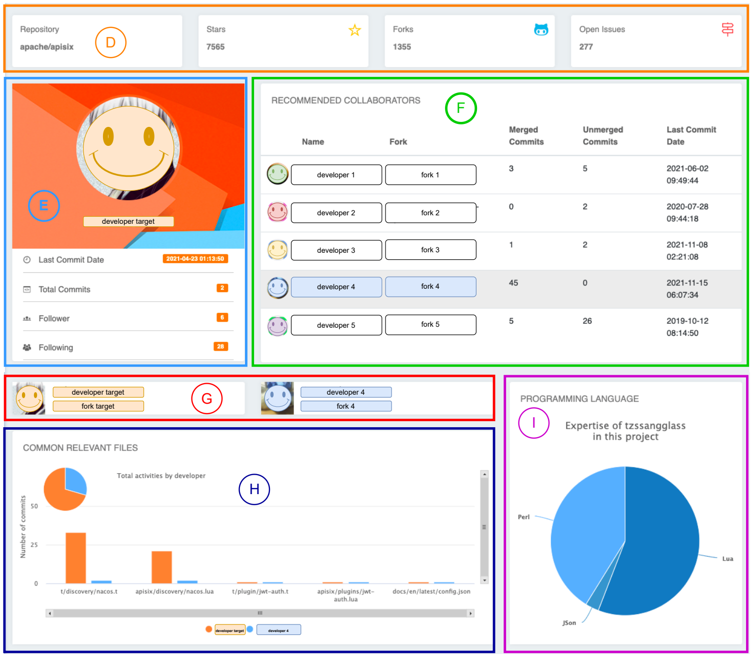

Figure 3. Overview of developer recommendations and their aggregated information (Constantino and Figueiredo, 2022).

belong. Frame (E) presents information about the target developer, such as their name, avatar, last commit date, number of total commits, followers, and followings. Finally, Frame (F) shows a list of recommended developers with similar interests based on co-changed files.

In Frame (F), users can select one of the recommended collaborators to compare with the target developer shown in Frame (E). Once selected, Frame (G) displays the two chosen collaborators along with their names and forks, linked with their GitHub profile. Frame (H) enables users to analyze the common files of the two developers. For instance, "*t/discovery/nacos.t*" and "*apisix/discovery/nacos.lua*" are common files that both developers. They are interested in and are familiar with these files (Figure 3). Finally, in Frame (I), the recommended developer's expertise (programming language) related to the focused project is presented. This expertise is calculated as a percentage of the total number of files changed in each programming language. Note that, this feature is not the primary focus of our current work. However, we leave this space open for potential avenues for future research. Other works, such as (Oliveira et al., 2019, 2020; de Neira et al., 2018), explored the expertise of the developers.

4 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 instructions and tools necessary were available to participants. Besides, the first author was available to clarify any doubts. To collect the data, we adopted questionnaires specially designed for this evaluation by using the Google Forms[10] service. Next, we describe our goal, research questions, formulated hypotheses, and the research method.

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

4.1 Study Goal

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
from the point of view of developers
in the context of recommendations based on co–changed files in the open-source environment.

4.2 Research Questions

To achieve our goal, we based our evaluation method on 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 this RQ, 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.

4.3 Hypotheses Formulation

We defined hypotheses for RQ_1: in which tool (CoopFinder or GitHub) would it be easier for finding collaborators with similar interests. To answer 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.

$\mathbf{H}_0$: There is no significant difference related to ease of use between CoopFinder or GitHub.
$\mathbf{H}_a$: 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-user) would it impact the use of the CoopFinder or GitHub. To answer RQ_2 we evaluated the answers (correct, incorrect and the blank) that participants should provide for each task proposed. Thus, the null and alternative hypotheses are:

$\mathbf{H}_0$: There is no significant difference in the hit rate between the GitHub users and non-users.
$\mathbf{H}_a$: There is significant difference in the hit rate between the GitHub users and non-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:

$\mathbf{H}_0$: There is no significant difference related to time (in minutes) to perform all tasks using CoopFinder or GitHub.
$\mathbf{H}_1$: There is significant difference related to time (in minutes) to perform all tasks using CoopFinder or GitHub.

4.4 Research Method

To answer the research questions, we planned and performed an experiment study, as shown in Figure 4.

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 5.1 presents the overview of the participants selected. We received responses from 43 participants. Eight participants did not complete all questionnaires; thus, they were excluded.

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 4). 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 4. 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–questionnaire with open-ended questions, allowing participants to give feedback about the CoopFinder tool.

Experiment tasks. We defined and adapted the tasks for each tool to have the same goal (Table 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 scenarios and tasks are available online for future replications/extensions[11].

Post–assignment questionnaire. After the experiment, we sent a short questionnaire to the participants regarding their

[11] https://github.com/kattiana/coopfinder

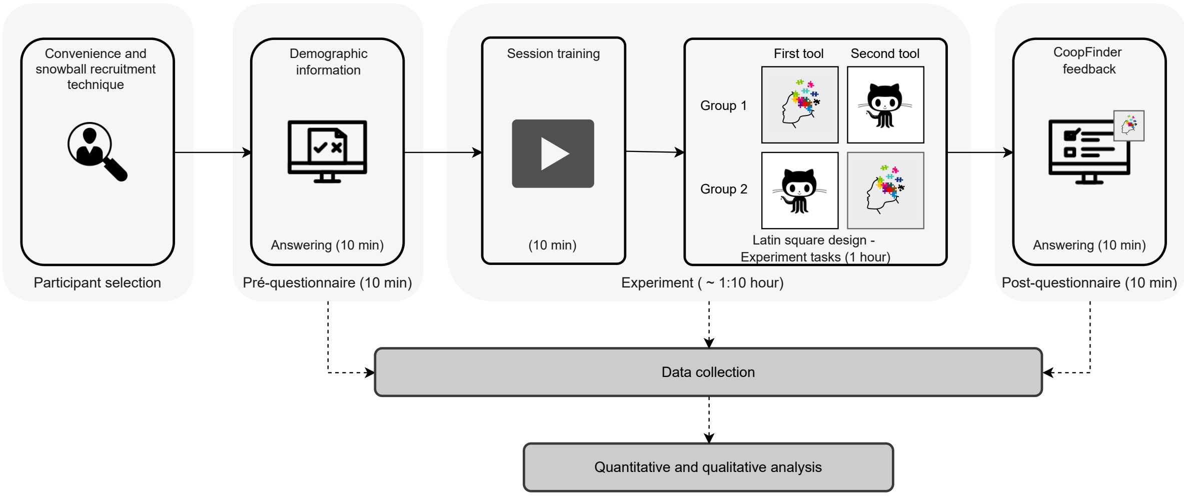

Figure 4. Experiment Design.

Table 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

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?
- 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 4). All data were analyzed, interpreted and reported in the results. Besides, all questionnaires, experiment tasks are available online for future replications/extensions.

Quantitative and qualitative analysis. First, we collect quantitative and qualitative data from the online survey and mined data about the participants in social platforms, such as GitHub and LinkedIn. Section 5 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 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). All three assumptions were met in our case. We used the R language, RStudio[12], and some statistical R packages, such as "ggplot2", "scales", and "rstatix".

Ethical considerations. This work involves experiments with human subjects. All participants gave the consent for their answers to be used in this research. Regarding participant data, all sensitive information (i.e., names or GitHub profile) has been previously anonymized to ensure the privacy of participants. Last, this research was approved by the Committee for Ethics in Research of our institution before performing this work.

5 Study Results

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

5.1 Participant Overview

A user study was conducted with 35 participants to evaluate the usefulness and satisfaction of users with the CoopFinder tool. Participants involved in this study are 35 developers

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

enrolled in courses related to Software Engineering. All participants are graduated (M.Sc. and Ph.D students) or close to graduate. Table 2 shows some profiling information of these participants related to gender (26 males and 9 females participants), the time of experience in software development contributions. Finally, if they were or not a GitHub contributor.

Table 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

About 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., 2015; 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 developers to help them and strengthen the ties among them for improving collaborations in the project.

5.2 How easy is it to find collaborators using CoopFinder? - RQ$_1$

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 the Table 1. However, the general goal of this set of tasks is to make it easier to find a suitable collaborator with similar interests in co–changed files. Table 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 3). Indeed, the CoopFinder prototype 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.

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.

5.3 Does the expertise with GitHub impact on the effectiveness of finding collaborators? - RQ$_2$

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 GitHub non-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. Table 4a and 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 4a shows the predominance of correct answers when participants performed the tasks using the CoopFinder tool. On the other hand, Table 4b shows the answers 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"*.

Table 3. Statistic Table.

	CoopFinder				GitHub				W
Tasks	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.*

Table 4. Results of tasks performed by GitHub user and non-user.

	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	**

(a) CoopFinder

	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

(b) GitHub

*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.*

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 3), the Fisher's exact test showed a significant statistical difference for the two samples (*p-value* = 0.02).

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.

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

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 5 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 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 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–value < 0.001*). It shows that the time required for performing all tasks using CoopFinder and GitHub was significantly different. Combined with Figure 5, we observed that participants spent less time using CoopFinder, than using GitHub to perform the tasks.

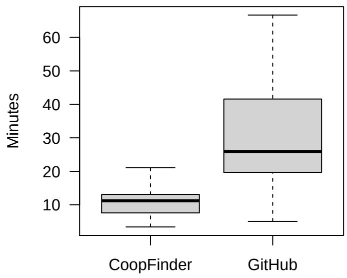

Figure 5. Distribution of time (in minutes) of the tasks performed by participants when they used CoopFinder and GitHub.

Table 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).

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.

5.5 How do participants perceive CoopFinder? - RQ$_4$

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.

5.6 How could the developer recommendations be improved? - RQ_5

In this research question, we asked for participants which social or technical features we could explore to improve the developer recommendation. Table 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 6).

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 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.

6 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. (2021).

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 project 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, there may be several other important issues in the collected data, not yet discovered or reported by us.

Table 6. Other features to improve the recommendations.

Tasks	GitHub User #	GitHub non–User #	Total #	Total %
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

7 Conclusion and Future Work

This work described a controlled experimental study to investigate the perceptions of the developers using CoopFinder a prototype tool to support two strategies for recommending collaborations. This 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-users. Participants answered the background questionnaire, the questionnaires for the experiment tasks for both tools.

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. As future work, we intend to evaluate CoopFinder in real context of use, to see how often the recommendations actually foster collaboration.

8 Acknowledgments

Many thanks to participants of our experiment and reviewers. This study is an extended version of the paper published in the XVIII Brazilian Symposium on Collaborative Systems (SBSC 2023). This research was partially supported by Brazilian funding agencies: CAPES (88881.189537/2018-01) and FAPEMIG (Grant PPM-00651-17).

References

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

Barcomb, A., Stol, K.-J., Fitzgerald, B., and Riehle, D. (2020). Managing episodic volunteers in free/libre/open source software communities. *IEEE Transactions on Software Engineering (TSE)*, 48(1):260–277.

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

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

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

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

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

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

Constantino, K., Belém, F., and Figueiredo, E. (2023a). Dual analysis for helping developers to find collaborators based on co-changed files: An empirical study. *Journal of Software: Practice and Experience (JSPE)*, pages 1–27.

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

Constantino, K., Prates, R., and Figueiredo, E. (2023b). Recommending collaborators based on co–changed files: A controlled experiment. In *Proc. of the 18th Brazilian Symposium on Collaborative Systems*, pages 154–168.

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

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

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

Grounded Theory.

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

Crowston, K. and Fagnot, I. (2018). Stages of motivation for contributing user-generated content: A theory and empirical test. *International Journal of Human-Computer Studies (IJHCS)*, 109:89–101.

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

de Neira, A. B., Steinmacher, I., and Wiese, I. S. (2018). Characterizing the hyperspecialists in the context of crowdsourcing software development. *Journal of the Brazilian Computer Society*, 24(1):1–16.

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

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

Flick, U. (2018). *Designing Qualitative Research*. Qualitative Research Kit.

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

Gamalielsson, J. and Lundell, B. (2014). 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.

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

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

Gousios, G., Zaidman, A., Storey, M.-A., and Deursen, A. v. (2015). Work practices and challenges in pull-based development: The integrator's perspective. In *Proc. of the 37th International Conference on Software Engineering (ICSE)*, volume 1, pages 358–368.

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

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

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

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

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

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

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

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

Pinto, G., Steinmacher, I., and Gerosa, M. (2016). More common than you think: An in-depth study of casual contributors. In *Proc. of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 112–123.

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

Rahman, M. M., Roy, C. K., Redl, J., and Collins, J. A. (2016). Correct: Code reviewer recommendation at github for vendasta technologies. In *Proc. of the 31st International Conference on Automated Software Engineering (ASE)*, page 792–797.

Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35.

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

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

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

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

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

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

Surian, D., Liu, N., Lo, D., Tong, H., Lim, E.-P., and Faloutsos, C. (2011). Recommending people in developers' collaboration network. In *Proc. of the 18th Working Confer-*

ence on Reverse Engineering (WCRE), pages 379–388.

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

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

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

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

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

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., and Regnell, B. (2012). *Experimentation in Software Engineering*.

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

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

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