Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Dual Analysis for Helping Developers to Find Collaborators Based on Co–Changed Files: An Empirical Study

Kattiana Constantino | Fabiano Belém | Eduardo Figueiredo

[1]Computer Science Department, Federal University of Minas Gerais, Belo Horizonte/MG, Brazil

Correspondence
*Corresponding Kattiana Constantino. Email: kattiana@dcc.ufmg.br or kattiana@gmail.com

Summary

Software developers must collaborate at all stages of the software life-cycle to create successful complex software systems. To enable this collaboration, social coding platforms, e.g., GitHub, include an increasing number of tools to support collaboration. However, for large projects with hundreds of dynamic developers, such as several successful open–source projects, it can be complex to find developers with the same interest and familiarity and thus, gain suitable collaborations and new insights. In this context, resources and efforts may be wasted, 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. In this context, this paper presents an empirical study aiming to evaluate two strategies to recommend collaborators based on co–changed files. Inspired in the TF–IDF (Term Frequency–Inverse Document Frequency) weighting scheme established in the Information Retrieval field, these strategies first estimate the importance of relevant files modified by developers and use these estimates to represent each developer "profile". As a second step, they estimate the similarity between developers using the Cosine metric, providing top-ranked developers according to this measure as recommendations. We evaluated these strategies based on an extensive survey with 102 real–world developers. We observed that developers have interest and familiarity with the co–changed files for all strategies evaluated. These considerations are of relevance because many opportunities for contributions to the project are linked to coding. Thus, theses results may indicate one less barrier for improving collaboration among developers. Overall, the strategies present an acceptance rate of up to 81%, contributing to the discovery of further collaborators.

KEYWORDS:
Collaborative software development; Collaboration in software development; Developer recommendation; Distributed collaboration.

1 | INTRODUCTION

Successful software projects require appropriate collaborators interacting with each other across the entire development life-cycle. Contributors who enjoy the work or feel an obligation to the project are more likely to remain than those driven by personal interest[1,2]. Thus, it is necessary to offer proper 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[3,2,4,5,6,7]. Unfortunately, for social coding platforms, e.g., GitHub[1], it is still challenging to identify a suitable collaborator to strengthen their ties and improve the engagement and quality of contributions. One of the reasons for this problem is that reliable information for identifying the collaborator is often not readily available for users[8,9,10].

Therefore, although finding the suitable developer to form a team, a partner or a mentor is not a new problem, this subject still finds a lot of space with the growth of social coding platforms, such as GitHub, and their impact on open–source software projects. Finding suitable developers also call the attention of several researchers interested in verifying or identifying what factors may impact collaborations, knowledge sharing, or strengthening the ties among project developers[9]. Oliveira et al.[11,12] identified developers with expertise in specific libraries from GitHub. Other previous work also explore social and coding activities to recommend developers for different purposes in software development projects, as recommending a core developer[8,10,13], or the most appropriate developer to integrate a pair of branches[14]. However, previous work still lacks empirical knowledge on the effectiveness of strategies based on co–change files to indicate suitable collaborators in software projects.

This paper presents an empirical study aiming to evaluate two developer recommendation strategies based on coding activities, especially in co–changed files, that is, modifications made by developers on the same file. These sets of files can indicate that developers have interests and familiarity with specific part of the project, impacting directly on collaborative work among developers. Thus, we considered the co–changed files to strengthen the ties among developers[8,10]. 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 (LOC).

In order to evaluate these recommendation strategies, we mined data from GitHub public repositories and surveyed 102 developers from these repositories. As a result, we could observe that developers have a similar interest and are familiar with the co–change files. These considerations are of relevance because many opportunities for contributing to the project are linked to coding tasks[15,16]. Thus, they may indicate one less barrier for improving collaboration among developers of the project. Moreover, the acceptance rates of the GitHub surveyed developers are 80% and 65% for STRATEGY 1 and STRATEGY 2, respectively. We also analyzed the combination of both strategies that presented an 81% acceptance rate. Furthermore, these results provide valuable insights for the future creation of systems that strengthen the ties of online collaborative communities.

Our main contributions can be summarized as follow.

1. We perform an empirical study to evaluate recommendation strategies based on co–changed files to find collaborators. We emailed 1,113 developers from 50 public projects hosted on GitHub. We have a response rate around 9%. In total, 102 real-world developers answered the survey;

2. We provide insights into the preferences of collaborators and encountered opportunities to improve the collaborations based on their similar interests in collaborative tasks;

3. We propose 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.

Our comprehensive replication package is available online for future replications/extensions[17]. The remainder of this paper is organized as follows. Section 2 motivates this paper and presents the problem statement. Section 3 presents the two recommendation strategies based on co–changed files. Section 4 describes the study settings. Section 5 reports and analyzes the results of this study. Section 6 reports a qualitative analysis and presents an overview of the tool support. In Sections 7 and 8, some threats to the study validity and related works are discussed. Finally, we end this paper with some concluding remarks and discussing directions for further work (Section 9).

[1]https://github.com/

2 | MOTIVATION AND PROBLEM STATEMENT

This section presents a motivating scenario and defines the problem addressed in this work.

2.1 | Motivating Scenario

Let's consider two hypothetical situations. First, Mary is a core team member in an open–source software (OSS) project. She would like to have more contributors to develop new features, enhance, and maybe help managing the project. Mary also knows that many developers made the last contribution a long time 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 the others to contribute again to the project or to make their first contributions. The chances of engagement and assertive contributions would be greater.

Joseph is a young developer and a volunteer in an OSS project in the second hypothetical situation. He has made a few contributions to the project. For example, he was recently asked to design a new feature for a project hosted on `GitHub`. However, Joseph is not familiar with the project. Thus, he needs some help. Perhaps he could find another developer to discuss various design ideas to have new insights. Therefore, Mary and Joseph look for the solution for 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 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[2]. Probably, the post author thought that the work of these developers 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?*

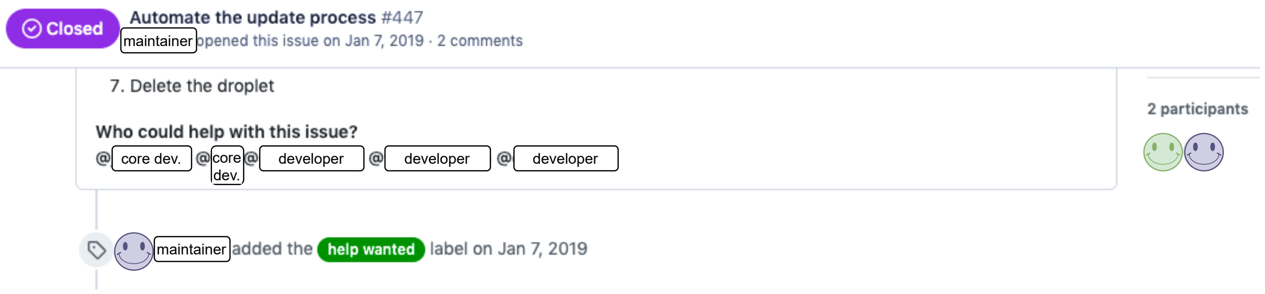
Figure 2 presents a second part of the same example. After one day, another developer different from the five called ones offered to help. Afterward, the issue author offered to code together or to help this developer as a mentor. By observing this second situation, we could wonder: *since the core members 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 actually 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 to solve that issue of this project.

2.2 | Problem Statement

Previous work has shown that developers usually prefer to request collaboration from core team members, who are supposed to have sufficient motivation, knowledge, and experience in the project[8,18]. However, based on other prior studies, core team members may be overwhelmed and, as a result, they may not provide collaborative support promptly[19,20,21]. 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[22]. Hence, all contributions should be valuable and encouraged[23,24,25].

Previous work also mentioned that the lack of some roles that compose the core team, such as maintainers, supporters, reviewers, and others, impacts the sustainability of the project[13,14]. 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[26,27]. Moreover, other developers may be being underused, even scarce, or with restricted access to information due to limited knowledge sharing opportunities (e.g., collaborations, discussions)[28]. All of the issues raised above are on how the community of developers relate to each other. Moreover, how these relationships impact positively or negatively on the sustainability of the software project and product. These situations also lead us to question: how to balance and optimize collaboration among project developers?

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


Closed Automate the update process #447
maintainer opened this issue on Jan 7, 2019 · 2 comments

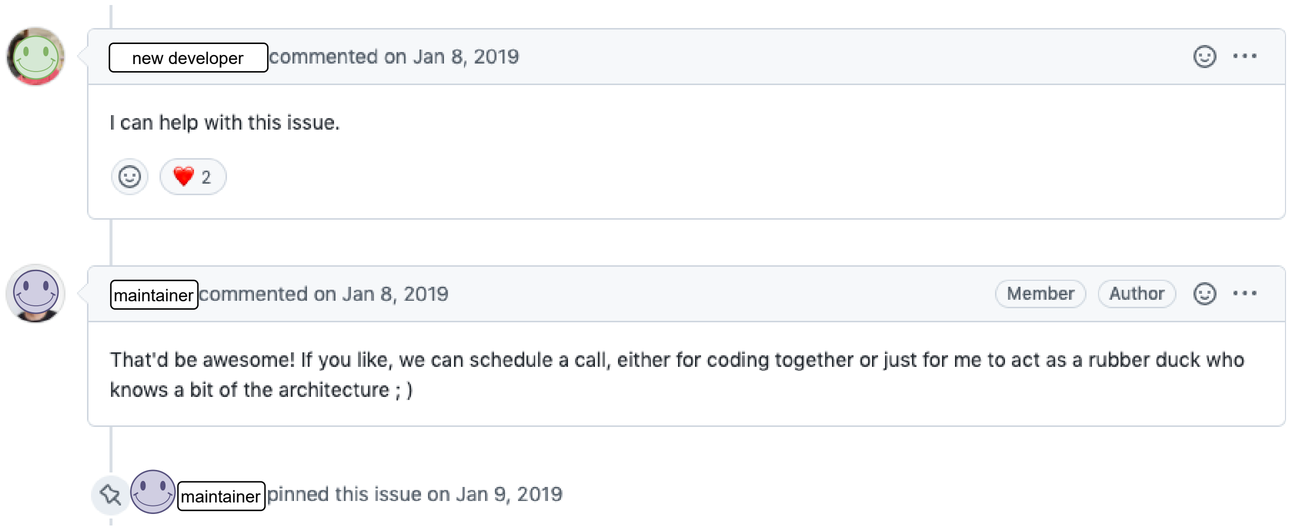
7. Delete the droplet

Who could help with this issue?
@ core dev. @ core dev. @ developer @ developer @ developer

2 participants

maintainer added the help wanted label on Jan 7, 2019

FIGURE 1 Core member called other core developers to help with an issue.


new developer commented on Jan 8, 2019

I can help with this issue.

❤️ 2

maintainer commented on Jan 8, 2019 Member Author

That'd be awesome! If you like, we can schedule a call, either for coding together or just for me to act as a rubber duck who knows a bit of the architecture ;)

maintainer pinned this issue on Jan 9, 2019

FIGURE 2 A different developer offered to help with this issue.

3 | STRATEGIES OF RECOMMENDING DEVELOPERS

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[8,10]. 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 3 presents an overview of the

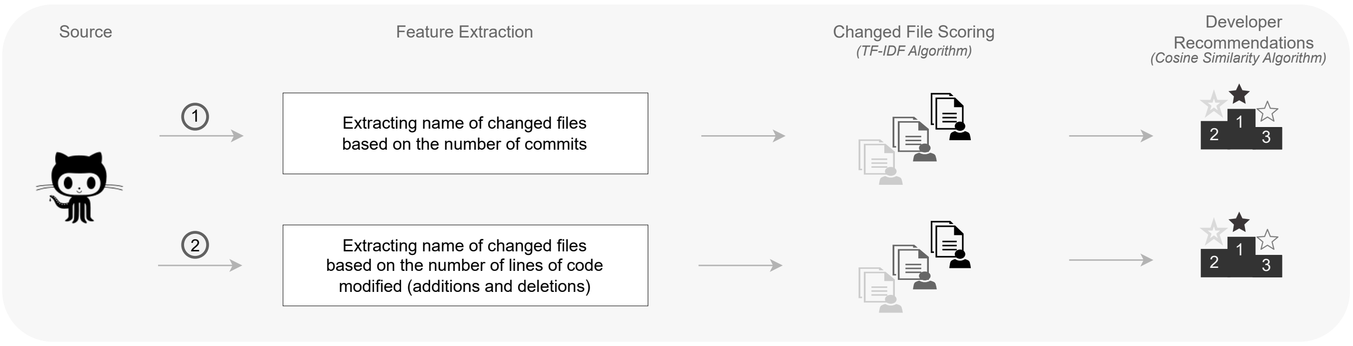

Source
Feature Extraction
Changed File Scoring
(TF-IDF Algorithm)
Developer Recommendations
(Cosine Similarity Algorithm)
1
Extracting name of changed files based on the number of commits
2
Extracting name of changed files based on the number of lines of code modified (additions and deletions)
2 1 3
2 1 3

FIGURE 3 Overview of recommendation strategies.

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 3. We exploit Information Retrieval (IR) techniques[29] to depict developers based on the files they edit. Each step of the general recommendation strategy presented in Figure 3 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 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.

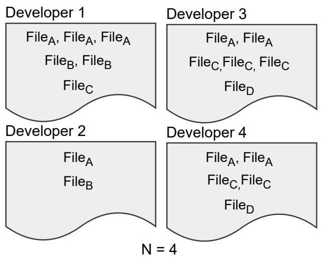
To extract these changes, we used the following metrics: *Number of commits*, used in Strategy 1, calculates the number of times a developer has modified a file. *Number of changed lines of code (LoC)*, used in Strategy 2, calculates the sum of the number of code lines added and removed in a specific 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 5. 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 3. To this end, we use an algorithm called Term Frequency - Inverse Document Frequency (TF-IDF)[30] to get the rank of relevant files by developers. The definition of the algorithm is adapted for our context. Figures 4 and 5 present how this algorithm works as a "relevance scoring" for Strategies 1 and 2, respectively.

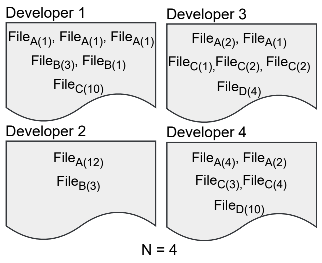
Term Frequency (TF) assigns a higher relevance score to the most frequent terms (words) in a text document[31]. In our context, we consider a term as a source code file in the project. Thus, for Strategy 1, TF means the number of times the developer changed a file in the project (*number of commits*). For Strategy 2, TF is defined as the total number of code lines added and removed in the file that the developer 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 (.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 4). 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 (Figure 5). 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 (Figure 4). In this commit, Developer 2 made three modifications that can be two additions and one deletion of lines of code. (Figure 5). Thus, the TF results for File_B related to Developer 2 are $\text{TF}_{Dev2} = 1$ (Figure 4) and $\text{TF}_{Dev2} = 3$ (Figure 5), 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



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 4 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.


Developer 1
File$_{A(1)}$, File$_{A(1)}$, File$_{A(1)}$
File$_{B(3)}$, File$_{B(1)}$
File$_{C(10)}$
Developer 3
File$_{A(2)}$, File$_{A(1)}$
File$_{C(1)}$,File$_{C(2)}$, File$_{C(2)}$
File$_{D(4)}$
Developer 2
File$_{A(12)}$
File$_{B(3)}$
Developer 4
File$_{A(4)}$, File$_{A(2)}$
File$_{C(3)}$,File$_{C(4)}$
File$_{D(10)}$
N = 4

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

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 = $\log((N+1)/(n_i+1)) + 1$ [3]. For instance, all developers modified File$_A$, thus IDF$_{FileA} = 1$ (Figures 4 and 5). Differently, only two developers have modified File$_B$, thus IDF$_{FileB} = 1.51082562$.

Finally, we combine both metrics described above by 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$_A$ (Figure 4). 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 (Figure 4), while STRATEGY 2 provides the opposite result (Figure 5). Another example, for STRATEGY 1, the rank of files to Developer 4 was File$_C$, File$_A$, and File$_D$ (Figure 4). 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.

DEVELOPER 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[32,33,34] because of its potential to quantify the similarity of two objects[35]. 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.

$$cosine\ similarity\,(A,B) = \frac{AB}{\sqrt{A^2}\sqrt{B^2}} \tag{1}$$

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

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

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 1 summarizes the input (in) and output (out) of each strategy.

TABLE 1 Summarizing the developer recommendation strategies.

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

4 | STUDY DESIGN

This section presents the design of a survey study to evaluate the developer recommendations based on co–changed files (STRATEGY 1 and STRATEGY 2). Data were collected using an opinion survey. We describe below our goal, research questions, formulated hypotheses, and the research method.

4.1 | Goal and Research Questions

We set the goal of our study using the Goal/Question/Metric template (GQM)[36]. Following such a goal definition template, the scope of our 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 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[37,38]. 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.

4.2 | Hypotheses Formulation

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[4].

4.3 | Research Method

To answer the research questions, we planned and performed a survey study, as shown in Figure 6. In summary, we select public projects hosted on GitHub. We used GitHub REST API version 3 to retrieve data from the original repositories and

[4]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.

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 produced the same top-1 recommendation. The survey study was executed under the following steps (see Figure 6).

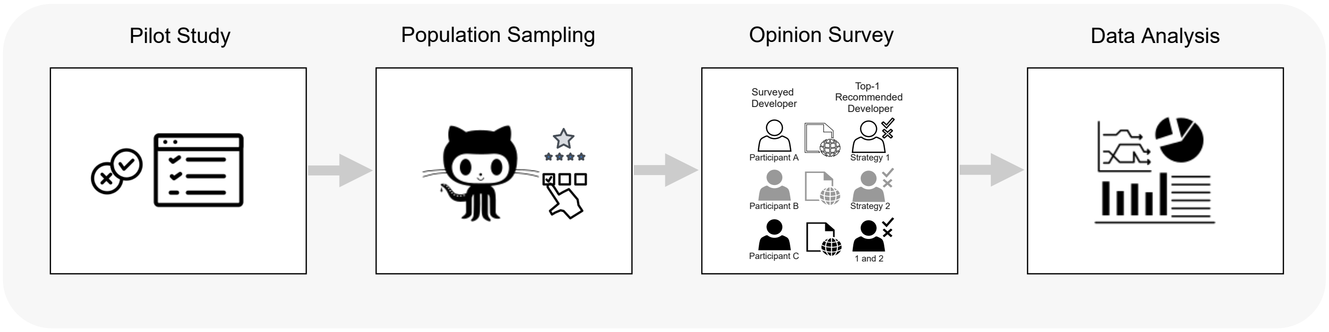


FIGURE 6 Research method overview.

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 4). 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 5.6, 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. 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 [5] (Version 1.0.0, 12K stars, 5K forks and Java as predominate language), Pandas [6] (Version 1.1.3, 31K stars, 13K forks and Python as predominate language), and Vue [7] (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[39,40]. 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

[5]https://github.com/eclipse/deeplearning4j
[6]https://github.com/pandas-dev/pandas
[7]https://github.com/vuejs/vue-next

TABLE 2 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

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 2 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 3 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 7 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).

TABLE 3 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.*

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 (see Figure 6). Table 4 shows the list of questions for the final version (after the pilot study) related to evaluate the recommendation strategies. 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

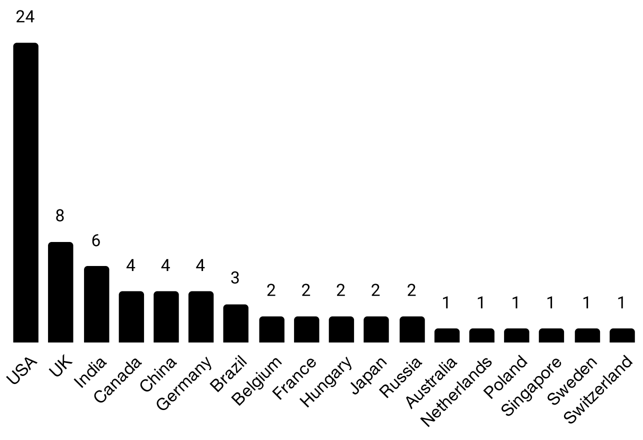


FIGURE 7 Location of the survey participants.

TABLE 4 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)

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

DATA ANALYSIS. First, we collect quantitative and qualitative data from the online survey and mined data, such as demographic information. Section 5 presents the descriptive analysis of these data and Chi-Squared (χ^2) test[41,42]. 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)[43]. 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[44]. We used the R language, RStudio[8], and some statistical R packages, such as "dplyr" and "rstatix". Finally,

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

we qualitatively analyzed the open question in items SQ4 and SQ6 of the questionnaire. In Section 6.1, we applied the open-coding techniques for qualitative research[45]. Afterward, 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 is resolved by joint discussion among the researchers involved in this paper.

5 | RESULTS

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

5.1 | 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?*) (see Table 4). 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 5 shows all combinations of these groups based on developer preferences. As detailed in Section 4, 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.

TABLE 5 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

5.2 | 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*) (see Table 4). 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 summarizes of data collected with Likert rating scales[46] 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. 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[43]. According to Chi-Squared test, the p-value is 0.003, which allows us to conclude that the precision is statistically different for strategies ($\chi^2 = 8.37$, $\alpha = 0.05$). Besides, 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 (see Table 6). We obtain 64% and 48% of precision for STRATEGY 1 and STRATEGY 2, respectively. The comparison of precision with Fisher's Exact test shown that there were statistically significant differences between strategies (*p-value* $= 0.001$, $\alpha = 0.05$). Therefore, being interested in

TABLE 6 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.*

code changes of other developers may indicate an openness to creating ties 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.

5.3 | 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*) (see Table 4). We followed the same procedures as explained before for RQ_1 (Section 5.2). That is, we also presented another recommended developer of the same project and their set of co–changed files. Table 7 summarizes of data collected with Likert rating scales for the SQ3 question.

TABLE 7 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.*

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 7 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 7, 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 ($\chi^2 = 2.15$, *p-value* $= 0.14$, $\alpha = 0.05$). 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-value* = 0.135, $\alpha = 0.05$). 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[47]. 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.

5.4 | 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...*) (Table 4). Besides, software development tasks are the most prominent category regarding what participants understand as collaborative contributions to projects, as observed in our prior works[15,16]. Each participant analyzed the contributions of another developer of the same project, as presented in Figure 6. 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 8 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.

TABLE 8 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

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 9

TABLE 9 Percentage of non–acceptance or acceptance the recommended developers in each strategy.

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

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 ($\chi^2 = 0.73$, *p-value* $= 0.392$, $\alpha = 0.05$). As results show that when applied in isolation, STRATEGY 1 performs better than STRATEGY 2.

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?*) (Table 4), 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%.

5.5 | 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[48,49]. 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 4). 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[50,48,21]. The others were categorized as casual developers, as defined in previous studies[48,51,52,49].

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 10 and 11 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 4), 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 10 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%

Table 10 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 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 11 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 11 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).

5.6 | Joint Strategy

As mentioned in Section 4, 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 12 presents the percentages of 47 participant answers (Table 2) 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 12 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 13 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 13 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 | QUALITATIVE ANALYSIS AND TOOL SUPPORT

In this section, we analyze the answers from the open questions of the survey to extract new insights, suggestions, or criticism. Furthermore, we present an overview of the COOPFINDER related to its implementation and how to use this tool.

6.1 | Qualitative Analysis

For the data analysis, we employed an approach inspired by the open and axial coding phases of ground theory[45]. 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 [45]. 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 14 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.

TABLE 14 Categories and codes for the feedback of 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

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 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 4), 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*”.

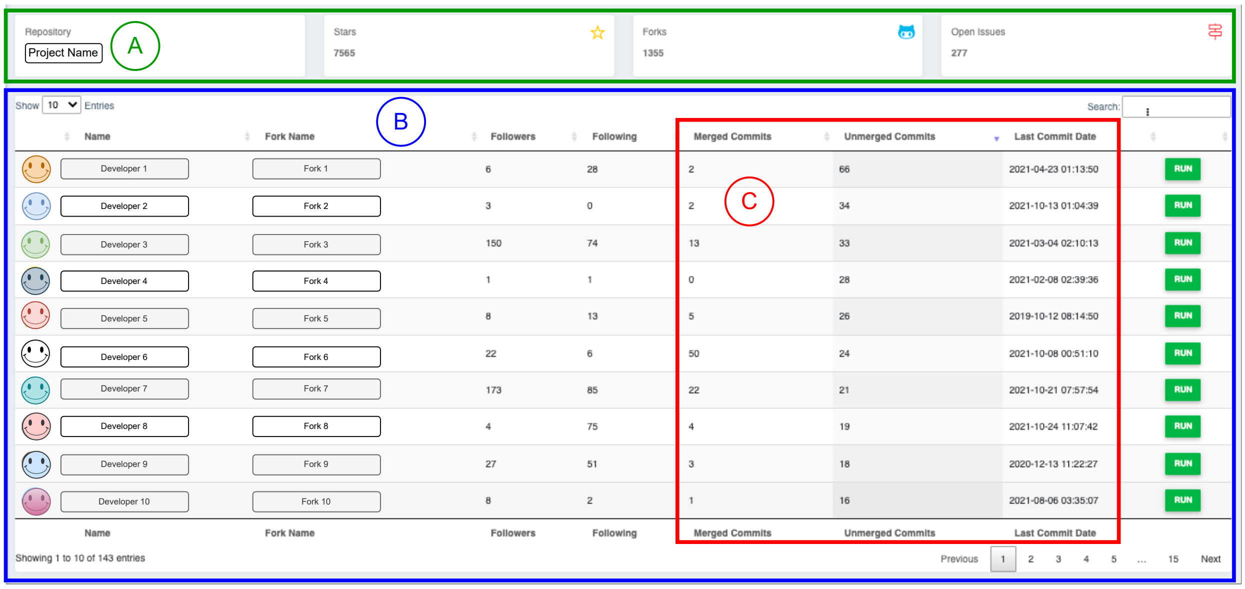
6.2 | Tool Support

COOPFINDER[9] [53] implements the two developer recommendation strategies (STRATEGIES 1 and 2) based on co–changed files (Section 3). We designed this Web tool using client-server architecture and visualization techniques. In addition, we used Python 3 language[10] and a free machine learning library for Python, called scikit-learn libraries[11] for the server-side. To implement views in COOPFINDER, we used the analytical data visualization components called HighCharts[12], a JavaScript library for manipulating documents based on data. Finally, we built the tool using components provided by the Bootstrap Framework[13], 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 8 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, number of their following. 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 9 illustrates the screenshot of COOPFINDER with the list of recommended collaborators for the target developer. This list may vary depending on the selected strategy. It also depends on the rank of relevant files for each project developer (Section 3). Frame (D) shows the project information to which the collaborators belong to, as mentioned before. 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.

[9] https://homepages.dcc.ufmg.br/ kattiana/coopfinder/welcome.html
[10] https://www.python.org/
[11] https://scikit-learn.org/stable/index.html
[12] https://www.highcharts.com/
[13] https://getbootstrap.com/



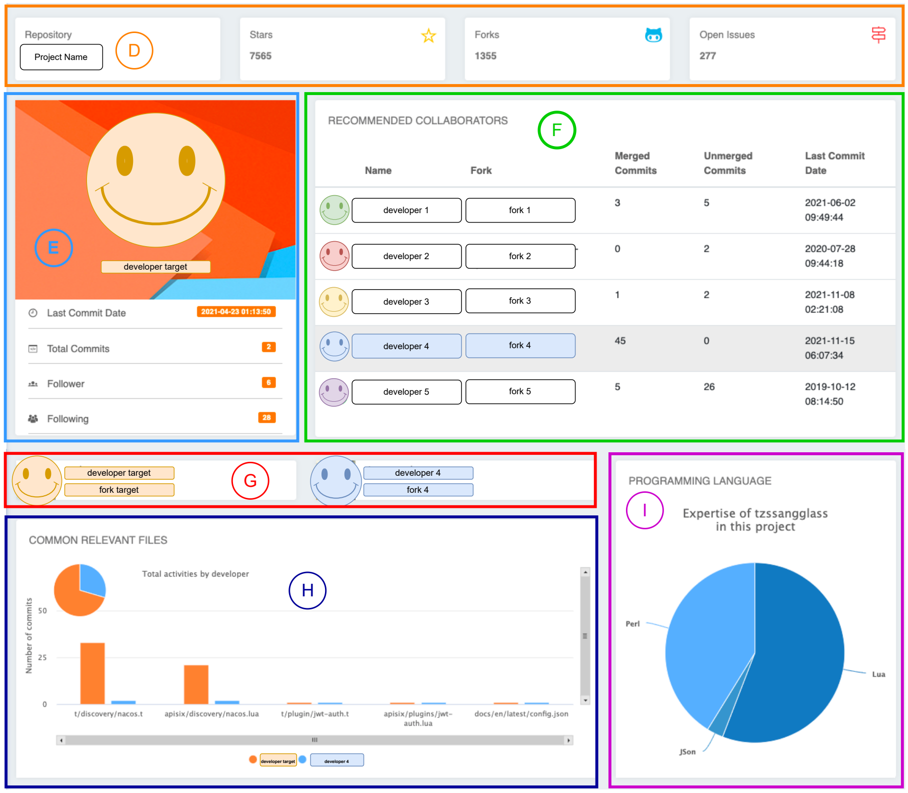
Repository	Stars	Forks	Open Issues
Project Name	7565	1355	277

Name	Fork Name	Followers	Following	Merged Commits	Unmerged Commits	Last Commit Date
Developer 1	Fork 1	6	28	2	66	2021-04-23 01:13:50
Developer 2	Fork 2	3	0	2	34	2021-10-13 01:04:39
Developer 3	Fork 3	150	74	13	33	2021-03-04 02:10:13
Developer 4	Fork 4	1	1	0	28	2021-02-08 02:39:36
Developer 5	Fork 5	8	13	5	26	2019-10-12 08:14:50
Developer 6	Fork 6	22	6	50	24	2021-10-08 00:51:10
Developer 7	Fork 7	173	85	22	21	2021-10-21 07:57:54
Developer 8	Fork 8	4	75	4	19	2021-10-24 11:07:42
Developer 9	Fork 9	27	51	3	18	2020-12-13 11:22:27
Developer 10	Fork 10	8	2	1	16	2021-08-06 03:35:07
Name	Fork Name	Followers	Following	Merged Commits	Unmerged Commits	Last Commit Date

Showing 1 to 10 of 143 entries

Previous 1 2 3 4 5 … 15 Next

FIGURE 8 Contributors information.



RECOMMENDED COLLABORATORS

Name	Fork	Merged Commits	Unmerged Commits	Last Commit Date
developer 1	fork 1	3	5	2021-06-02 09:49:44
developer 2	fork 2	0	2	2020-07-28 09:44:18
developer 3	fork 3	1	2	2021-11-08 02:21:08
developer 4	fork 4	45	0	2021-11-15 06:07:34
developer 5	fork 5	5	26	2019-10-12 08:14:50

FIGURE 9 Exploring developer recommendations.

7 | 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[43].

CONSTRUCT VALIDITY. The construct validity regards the relationship between theory and observation[43]. 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[43]. 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[43]. 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[43]. 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.

8 | RELATED WORK

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

COLLABORATION CHALLENGES. Collaboration brings known challenges. For instance, the turnover of contributors is one of the factors that impacts the sustainability of the project[22]. 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[2,20,54]. Furthermore, it is necessary to offer proper support in order 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[3,4,5,6]. 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[55,56,57]. Collaborative Development Environments integrate source code management tools and bug trackers with collaborative development features[58]. They can be a solution to the communication and coordination challenges of distributed software projects[59]. Besides, various social coding platform that allow developers to make contributions flexibly and efficiently, such as GitHub, GitLab [14], Bitbucket [15], SourceForge [16], and others. In our context, we focus on one social coding platform, GitHub, to improve open–source software development collaboration. GitHub is a example of a Collaborative Development Environment that provides an

[14]https://about.gitlab.com/
[15]https://bitbucket.org/
[16]http://sourceforge.net

infrastructure for collaborative development. Besides, GitHub is a Web-based code-hosting service that uses the GIT distributed version control system.

TABLE 15 An overview of prior works that explored recommendation strategies.

Paper	Purpose	Feature Set	Techniques*	Repository
[8]	user-user	source code data	IR	Subversion
[60]	user-task	source code data	IR/SM	Subversion
[9]	user-user	technical skills and project data	SM	SourceForge.Net
[10]	user-user	social interactions and project data	IR/SM	Git-based and community repository
[13]	user-task	social interactions and project data	ML	GitHub
[61]	user-task	source code data	SM	Gerrit code review
[32]	user-task	technical skills and source code data	SM	GitHub
[14]	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.*

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 15 shows an overview of many works in the literature related to recommendations for collaborative interactions in software engineering. For instance, Minto and Murphy et al. (2007)[8] ranked a list of the likely emergent team members with to communicate based on a set of files of interest. They applied a simple frequency-based weighting to form these recommendations. Besides, they extracted information available from the Subversion repository. In our work, we also recommend user to user of a project. 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)[9] 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 on only source code changes from the target project that the developer collaborates with other developers or with the project. Besides, we collected data from GitHub. Canfora et al. (2012)[10] 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. However, we extended our recommendations for all active collaborators of the project that need help or want to follow the work of another one. As mentioned before, we used TF-IDF and cosine similarity measures.

On the other hand, other works focused on recommendations whose general purpose is to recommend developers for a specific task. For instance, Kagdi and Poshyvanyk (2009)[60] 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)[13] 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)[61] 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 appropriate code-reviewers. Rahman et al. (2016)[32] 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. (2006)[62] identified reviewers who have changed files with similar names. Kagdi et al. (2012)[63] 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. (2019)[14] 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[64,65,66,67,68,69,70,71] have been done in context of recommendation projects for developers and vice-versa. For instance, Liu et al. (2020)[72] recommend software features from the users' perspective for designers of the project. Another example, Berkani (2020)[34] 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 shown a list of possible opportunities for collaboration and the surveyed participant chose in which of them they could work together with the recommended collaborator, as a mentor, as a tester, as a developer, as a reviewer. Besides, we evaluate these strategies of recommendations in the context of GitHub repositories, using data collected from the answers of the 102 surveyed developers, while previous work[8,60,13,61,32] rely 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[73]. Following prior studies[8,10], in this work, we focus on the *non cold start scenario*, leaving specific solutions to address cold start as future work.

9 | CONCLUSIONS AND FUTURE WORK

This work discussed an survey study to evaluate two strategies based on co–changed files. We collected the data from an opinion survey answered by 102 GitHub developers of popular projects (>1,000 stars). Our results shown that (i) we can conclude that developers identified by STRATEGY 1 manifested more interest in co–changed files from recommended developers. This consideration is relevant because coding tasks link many opportunities for contributions to the project. Thus, the interest of the developer in a specific code or part of them may motivate them to become a long-term developer in the project or be willing to encourage collaborations. Again, (ii) developers identified by STRATEGY 1 are more familiar with co–changed files than other developers from STRATEGY 2. Familiarity with code from other developers may indicate one less barrier for improving collaborations. (iii) The acceptance rates were 80% and 65% for STRATEGY 1 and STRATEGY 2, respectively. For joint strategy presented the best acceptance rate (81%), which rises evidence of the benefits of combining both STRATEGIES 1 and 2. (iii) 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.

In this work, we compute the #changed LoC metric, as well as the number of commits, considering the whole lifetime of the project, for both strategies. However, it is possible to configure this time window parameter, for both evaluated strategies, which we leave as future work. In another direction, as future work, we intend to evaluate further combinations of these strategies. For example, a good strategy would be the concatenation of the lists of recommendation of STRATEGY 1 and 2, in this order, that is, from the highest precision method to the lowest. Furthermore, we intend to combine our proposed strategies and exploit new evidence of the relevance of the recommendations, for example, other types of interactions among developers such as comments and discussions. Moreover, we also intend to evaluate the recommendations focusing on the newcomers perspective. Besides that, we intend to evaluate other aspects of the recommendation effectiveness, such as recall, and diversity.

10 | ACKNOWLEDGMENTS

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

11 | ORCID

Kattiana Constantino https://orcid.org/0000-0003-4511-7504
Fabiano Belém https://orcid.org/0000-0003-1076-2052
Eduardo Figueiredo https://orcid.org/0000-0002-6004-2718

References

1. Shah SK. Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science* 2006; 52(7): 1000–1014.

2. Crowston K, Fagnot I. Stages of Motivation for Contributing User-Generated Content: A Theory and Empirical Test. *International Journal of Human-Computer Studies* 2018; 109: 89–101.

3. Qiu HS, Nolte A, Brown A, Serebrenik A, 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). ; 2019: 688–699.

4. Barcomb A, Stol KJ, Riehle D, Fitzgerald B. Why do Episodic Volunteers Stay in FLOSS Communities?. In: Proceedings of the 41st International Conference on Software Engineering (ICSE). ; 2019: 948–959.

5. Zhou M, Mockus A. What Make Long Term Contributors: Willingness and Opportunity in OSS Community. In: Proceedings of the 34th International Conference on Software Engineering (ICSE). ; 2012: 518–528.

6. Zhou M, Mockus A. Who will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. *IEEE Transactions on Software Engineering* 2014; 41(1): 82–99.

7. Barcomb A, Stol KJ, Fitzgerald B, Riehle D. Managing Episodic Volunteers in Free/Libre/Open Source Software Communities. *IEEE Transactions on Software Engineering* 2020; 48(1): 260-277.

8. Minto S, Murphy GC. Recommending Emergent Teams. In: Proceedings of the 4th International Conference on Mining Software Repositories (MSR). ; 2007: 5-5.

9. Surian D, Liu N, Lo D, Tong H, Lim EP, Faloutsos C. Recommending People in Developers' Collaboration Network. In: Proceedings of the 18th Working Conference on Reverse Engineering (WCRE). ; 2011: 379–388.

10. Canfora G, Di Penta M, Oliveto R, 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). ; 2012: 1–11.

11. Oliveira J, Viggiato M, 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). ; 2019: 49–58.

12. Oliveira J, Pinheiro D, Figueiredo E. JExpert: A Tool for Library Expert Identification. In: Proceedings of the 34th Brazilian Symposium on Software Engineering (SBES). ; 2020: 386–392.

13. Jiang J, He JH, Chen XY. Coredevrec: Automatic Core Member Recommendation for Contribution Evaluation. *Journal of Computer Science and Technology* 2015; 30(5): 998–1016.

14. Costa C, Figueirêdo J, Pimentel JF, Sarma A, Murta L. Recommending Participants for Collaborative Merge Sessions. *IEEE Transactions on Software Engineering* 2021; 47(6): 1198-1210.

15. Constantino K, Zhou S, Souza M, Figueiredo E, Kästner C. Understanding Collaborative Software Development: An Interview Study. In: Proceedings of the 15th International Conference on Global Software Engineering (ICGSE). ; 2020: 55–65.

16. Constantino K, Souza M, Zhou S, Figueiredo E, Kästner C. Perceptions of Open-Source Software Developers on Collaborations: An Interview and Survey Study. *Journal of Software: Evolution and Process* 2021; 33: e2393.

17. *Replication Package, https://github.com/kattiana/coopfinder*. 2022.

18. Kononenko O, Baysal O, Godfrey MW. Code Review Quality: How Developers See It. In: Proceedings of the 38th International Conference on Software Engineering (ICSE). ; 2016: 1028–1038.

19. Yu Y, Wang H, Filkov V, Devanbu P, 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). ; 2015: 367–371.

20. Gousios G, Zaidman A, Storey MA, Deursen Av. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In: Proceedings of the 37th International Conference on Software Engineering (ICSE). ; 2015: 358-368.

21. Steinmacher I, Pinto G, Wiese IS, Gerosa MA. Almost There: A Study on Quasi-Contributors in Open-Source Software Projects. In: Proceedings of the 40th International Conference on Software Engineering (ICSE). ; 2018: 256–266.

22. Gamalielsson J, Lundell B. Sustainability of Open Source Software Communities Beyond a Fork: How and Why has the LibreOffice Project Evolved?. *Journal of Systems and Software* 2014; 89: 128–145.

23. Pham R, Singer L, Liskin O, Figueira Filho F, 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). ; 2013: 112–121.

24. Gousios G, Pinzger M, Deursen Av. An Exploratory Study of the Pull-Based Software Development Model. In: Proceedings of the 36th International Conference on Software Engineering (ICSE). ; 2014: 345–355.

25. Pinto G, Steinmacher I, 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). ; 2016: 112–123.

26. Avelino G, Passos L, Hora A, Valente MT. A Novel Approach for Estimating Truck Factors. In: Proceedings of the 24th International Conference on Program Comprehension (ICPC). ; 2016: 1–10.

27. Ferreira M, Valente MT, Ferreira K. A Comparison of Three Algorithms for Computing Truck Factors. In: Proceedings of the 25th International Conference on Program Comprehension (ICPC). ; 2017: 207–217.

28. Tamburri DA, Kruchten P, Lago P, Van Vliet H. Social Debt in Software Engineering: Insights from Industry. *Journal of Internet Services and Applications* 2015; 6(1): 1–17.

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

30. Salton G. Automatic Text processing: The transformation, Analysis, and Retrieval of. *Reading: Addison-Wesley* 1989; 169.

31. Yu CT, Salton G. Precision Weighting—An Effective Automatic Indexing Method. *Journal of Association for Computing Machinery* 1976; 23(1): 76–88.

32. Rahman MM, Roy CK, Redl J, Collins JA. CORRECT: Code Reviewer Recommendation at GitHub for Vendasta Technologies. In: Proceedings of the 31st International Conference on Automated Software Engineering (ASE). ; 2016: 792–797.

33. Franco MF, Rodrigues B, 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). ; 2019: 1–7.

34. Berkani L. A Semantic and Social-Based Collaborative Recommendation of Friends in Social Networks. *Software: Practice and Experience* 2020; 50(8): 1498–1519.

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

36. Basili VR, Weiss DM. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering (TSE)* 1984(6): 728–738.

37. Ge M, Delgado-Battenfeld C, Jannach D. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In: ; 2010: 257–260.

38. Belém FM, Batista CS, Santos RL, Almeida JM, Gonçalves MA. Beyond Relevance: Explicitly Promoting Novelty and Diversity in Tag Recommendation. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2016; 7(3): 1–34.

39. Krüger J, Wiemann J, Fenske W, Saake G, Leich T. Do You Remember this Source Code?. In: Proceedings of the 40th International Conference on Software Engineering (ICSE). ; 2018: 764–775.

40. Miller C, Widder DG, Kästner C, 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). ; 2019: 116–129.

41. Rayner J, Best D. Smooth Tests of Goodness of Fit: An Overview. *International Statistical Review/Revue Internationale de Statistique* 1990; 58: 9–17.

42. Mendenhall W, Beaver RJ, Beaver BM. *Introduction to Probability and Statistics*. Cengage Learning . 2012.

43. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B. *Experimentation in Software Engineering*. Springer Science & Business Media.

44. Miller R, Siegmund D. Maximally Selected Chi Square Statistics. *Biometrics* 1982; 38: 1011–1016.

45. Corbin J, Strauss A. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, Inc Thousand Oaks . 2014.

46. Robbins NB, Heiberger RM, others . Plotting Likert and Other Rating Scales. In: Proceedings of the Joint Statistical Meeting (JSM). ; 2011: 1058–1066.

47. Goyal R, Ferreira G, Kästner C, Herbsleb J. Identifying Unusual Commits on GitHub. *Journal of Software: Evolution and Process* 2018; 30(1): e1893.

48. Yamashita K, McIntosh S, Kamei Y, Hassan AE, 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. ; 2015: 46–55.

49. Barcomb A, Kaufmann A, Riehle D, Stol KJ, Fitzgerald B. Uncovering the Periphery: A Qualitative Survey of Episodic Volunteering in Free/Libre and Open Source Software Communities. *IEEE Transactions on Software Engineering* 2018; 46(9): 962–980.

50. Begel A, Simon B. Novice Software Developers, All Over Again. In: Proceedings of the 4th International Workshop on Computing Education Research (ICER). ; 2008: 3–14.

51. Lee A, Carver JC. Are One-Time Contributors Different? A Comparison to Core and Periphery Developers in FLOSS Repositories. In: Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM). ; 2017: 1–10.

52. Lee A, Carver JC, 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). ; 2017: 187–197.

53. Constantino K, Figueiredo E. CoopFinder: Finding Collaborators Based on Co–Changed Files. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). ; 2022: 1–3.

54. Gousios G, Storey MA, 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). ; 2016: 285–296.

55. Herbsleb JD. Global Software Engineering: The Future of Socio-Technical Coordination. In: Future of Software Engineering (FOSE). ; 2007: 188–198.

56. Cataldo M, Herbsleb JD, Carley KM. 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). ; 2008: 2–11.

57. Cataldo M, Herbsleb JD. Coordination Breakdowns and Their Impact on Development Productivity and Software Failures. *IEEE Transactions on Software Engineering* 2012; 39(3): 343–360.

58. Lanubile F, Ebert C, Prikladnicki R, Vizcaíno A. Collaboration Tools for Global Software Engineering. *IEEE software* 2010; 27(2): 52–55.

59. Abbattista F, Calefato F, Gendarmi D, Lanubile F. Incorporating Social Software Into Distributed Agile Development Environments. In: Proceedings of the 23rd International Conference on Automated Software Engineering (ASE). ; 2008: 46–51.

60. Kagdi H, Poshyvanyk D. Who Can Help Me with This Change Request?. In: Proceedings of the 17th International Conference on Program Comprehension (ICPC). ; 2009: 273-277.

61. Thongtanunam P, Tantithamthavorn C, Kula RG, Yoshida N, Iida H, Matsumoto Ki. 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). ; 2015: 141–150.

62. Zanjani MB, Kagdi H, Bird C. Automatically Recommending Peer Reviewers in Modern Code Review. *IEEE Transactions on Software Engineering* 2016; 42(6): 530-543.

63. Kagdi H, Gethers M, Poshyvanyk D, Hammad M. Assigning Change Requests to Software Developers. *Journal of software: Evolution and Process* 2012; 24(1): 3–33.

64. Terra R, Valente MT, Czarnecki K, Bigonha RS. A Recommendation System for Repairing Violations Detected by Static Architecture Conformance Checking. *Software: Practice and Experience* 2015; 45(3): 315–342.

65. Xu W, Sun X, Hu J, Li B. REPERSP: Recommending Personalized Software Projects on GitHub. In: Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME). ; 2017: 648-652.

66. Ponzanelli L, Scalabrino S, Bavota G, et al. Supporting Software Developers with a Holistic Recommender System. In: Proceedings of the 39th International Conference on Software Engineering (ICSE). ; 2017: 94–105.

67. Jiang JY, Cheng PJ, Wang W. Open Source Repository Recommendation in Social Coding. In: Proceedings of the 40th International Conference on Research and Development in Information Retrieval (SIGIR). ; 2017: 1173–1176.

68. Rahman MM, Riyadh RR, Khaled SM, Satter A, Rahman MR. MMRUC3: A Recommendation Approach of Move Method Refactoring using Coupling, Cohesion, and Contextual Similarity to Enhance Software Design. *Software: Practice and Experience* 2018; 48(9): 1560–1587.

69. Sajedi-Badashian A, Stroulia E. Vocabulary and Time Based Bug-Assignment: A Recommender System for Open-Source Projects. *Software: Practice and Experience* 2020; 50(8): 1539–1564.

70. Gong W, Lv C, Duan Y, et al. Keywords-driven Web APIs Group Recommendation for Automatic APP Service Creation Process. *Software: Practice and Experience* 2021; 51(11): 2337–2354.

71. Dey T, Karnauch A, Mockus A. Representation of Developer Expertise in Open Source Software. In: Proceedings of the 43rd International Conference on Software Engineering (ICSE). ; 2021: 995-1007.

72. Liu C, Yang W, Li Z, Yu Y. Recommending Software Features to Designers: From the Perspective of Users. *Software: Practice and Experience* 2020; 50(9): 1778–1792.

73. Hu L, Jian S, Cao L, Gu Z, Chen Q, 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). ; 2019: 3830–3837.

How to cite this article: Constantino K., Belém F., and Figueiredo E. (2023), Dual Analysis for Helping Developers to Find Collaborators Based on Co–Changed Files: An Empirical Study, *Softw. Pract. Exper.*, *2023;00:1–28*.