Finding Collaborations based on Co-Changed Files*

Kattiana Constantino[1], **Eduardo Figueiredo**[1]

[1]Computer Science Department
Federal University of Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

{kattiana,figueiredo}@dcc.ufmg.br

Abstract. *Collaboration is essential in software development, but finding suitable collaborators can be challenging in large projects like open-source ones. In this work, we proposed investigating collaborative development based on similar code interests and tool-supported strategies to help developers find suitable collaborators. Five empirical studies were conducted, including interview and survey studies. Two strategies based on co-changed files and a prototype tool named* COOPFINDER *were provided and evaluated for their effectiveness.* `GitHub`*users and non-users found the strategies and the tool useful. Our results suggest that fostering collaborations in projects can prevent wasted resources and sustain project continuity.*

Resumo. *A colaboração é essencial no desenvolvimento de software, porém encontrar colaboradores adequados pode ser um desafio em grandes projetos de código aberto. Neste trabalho, investigamos o desenvolvimento colaborativo de código com base em interesses similares para ajudar os desenvolvedores a encontrar colaboradores adequados. Cinco estudos empíricos foram conduzidos, incluindo entrevistas e questionários. Duas estratégias baseadas em arquivos co-alterados e um protótipo denominada* COOPFINDER *foram propostas e avaliadas. Usuários ou não do* `GitHub`*acharam as estratégias e a ferramenta úteis. Os resultados sugerem que promover colaborações em projetos pode evitar o desperdício de recursos e manter a continuidade do projeto.*

1. Introduction

Consider two hypothetical scenarios. In the first scenario, Mary is a core team member of an open-source software project who wants to attract more contributors to help develop new features and manage the project. However, she notices that many developers have not made any contributions for a long time or have stopped contributing altogether. Thus, Mary decides to organize an event to encourage the involvement of these inactive developers and attract new ones. Moreover, Mary realizes it would be interesting for the project if active developers motivate others to contribute again or make their first contributions. Thus, the chances of engagement and assertive contributions would be more significant.

In the second hypothetical scenario, Joseph is a young developer and a volunteer in an OSS project hosted on `GitHub`. He has tried to make a few contributions to a specific project. For example, he was recently asked to design a new feature for this

*This work relates to a Ph.D. thesis defended in the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais on July 15, 2022. Authors order: Ph.D. Candidate and Advisor.

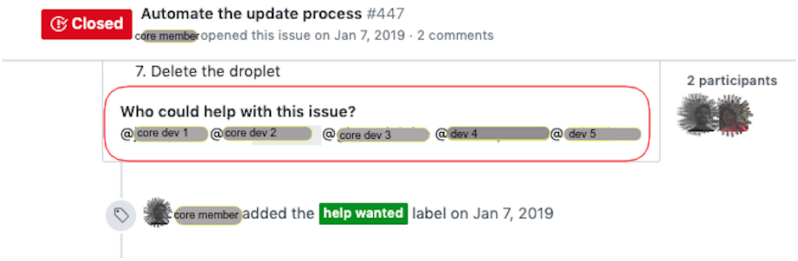

Figura 1. Core member called other core developers to help with this issue.

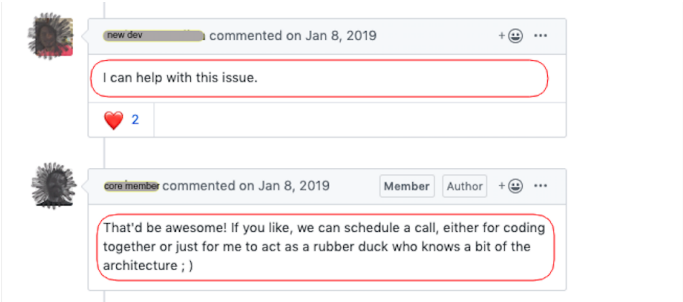

Figura 2. A new developer offered help with this issue. And, the core member suggested that they work collaboratively.

project. However, Joseph is not very familiar with this specific 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 to their problems. In other words, they want to find other developers with the same interests in the project. That is, developers prefer or are familiar with specific parts of the code, being able to make contributions regarding these parts. Consequently, they contribute to the engagement in the project as a whole and enhance the opportunities for collaborations.

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

Figure 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 happened. There was no record of commits on the new developer fork. Moreover, there is no evidence that any other developers helped the core member solve this project's issue.

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

2. Problem Statement

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

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

3. Research Goals

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

- **SG1** Investigate the motivations, processes, interactions, and barriers involved in collaboration during open–source software development.
- **SG2** Investigate how open the developers are for collaboration with others.
- **SG3** Provide tool–supported strategies based on co–changed files to find suitable collaborators.
- **SG4** Evaluate developers recommendations based on co–change files from the point of view of who receives the recommendations.
- **SG5** Evaluate the effectiveness of developer recommendation tools in supporting developers and maintainers, considering both perspectives (`GitHub` user and non–user).

4. Method

We divided this work into five main steps described in Figure 3. Therefore, this research begins with an interview study (Step 1). Afterward, we designed and applied a survey study (Step 2) to investigate if developers are open to collaborations. Following it, we designed and implemented tool–supported strategies of developer recommendation based

on similar interests (Step3). Next, we designed and applied a survey study (Step 4) to evaluate the developer recommendations. Finally, we performed a controlled experiment (Step 5) to complete the evaluation.

Step 1. As shown in Figure 3, we carried out an interview study to explore the collaborations, processes, communication channels, and barriers and challenges faced by developers in open–source software development. We focused on understanding (i) what motivates developers to collaborate, (ii) the collaboration process adopted, and (iii) challenges and barriers involved in collaboration. Furthermore, we set the goals of our interviews using the Goal/Question/Metric template (GQM) [Basili and Weiss 1984]. For the data analysis, we applied standard coding techniques for qualitative research [Corbin and Strauss 2014, Creswell and Creswell 2017].

Step 2. According to Easterbrook et al. (2008), survey studies, usually associated with the application of questionnaires, are used to identify characteristics of a great population. Surveys are meant to collect data to describe, compare, or explain knowledge, behaviors, and attitudes [Pfleeger and Kitchenham 2001]. We performed a survey study (Figure 3) to cross-validate the findings of ours interviews. We aimed to investigate how open developers work collaboratively based on their behaviors and to identify and check the main tasks to explore further collaboration opportunities.

Step 3. As detailed in Figure 3, based on the lessons learned from the previous steps, we designed and proposed two strategies of developer recommendation based on coding activities, especially in co–changed files, that is, modifications made by developers on the same file. Inspired in the TF–IDF (Term Frequency–Inverse Document Frequency)[Salton 1989] 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 [Salton 1971, Salton and Harman 2003], providing top-ranked developers according to this measure as recommendations. Furthermore, we designed and implemented a visual tool to support these strategies.

Step 4. We performed a survey study to evaluate two developer recommendation strategies based on co-change files from the who receives the recommendations (Figure 3). 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 [Minto and Murphy 2007, Canfora et al. 2012]. To extract these files, we considered the number of commits for STRATEGY 1. For STRATEGY 2, we used the number of changed lines of code. We mined data from `GitHub` public repositories and surveyed 102 developers from these repositories.

Step 5. We performed a controlled experimental study to evaluate two recommendation strategies and the proposed visual tool (Figure 3). Thus, we conducted a controlled experiment with 35 participants. To reduce the learning effect on the assessment results, we used the Latin square fisher1992arrangement to distribute the tasks and tools between two groups of participants (Figure 3). We asked participants to perform the questionnaires of experiment tasks to find collaborators with similar interests using a prototype recom-

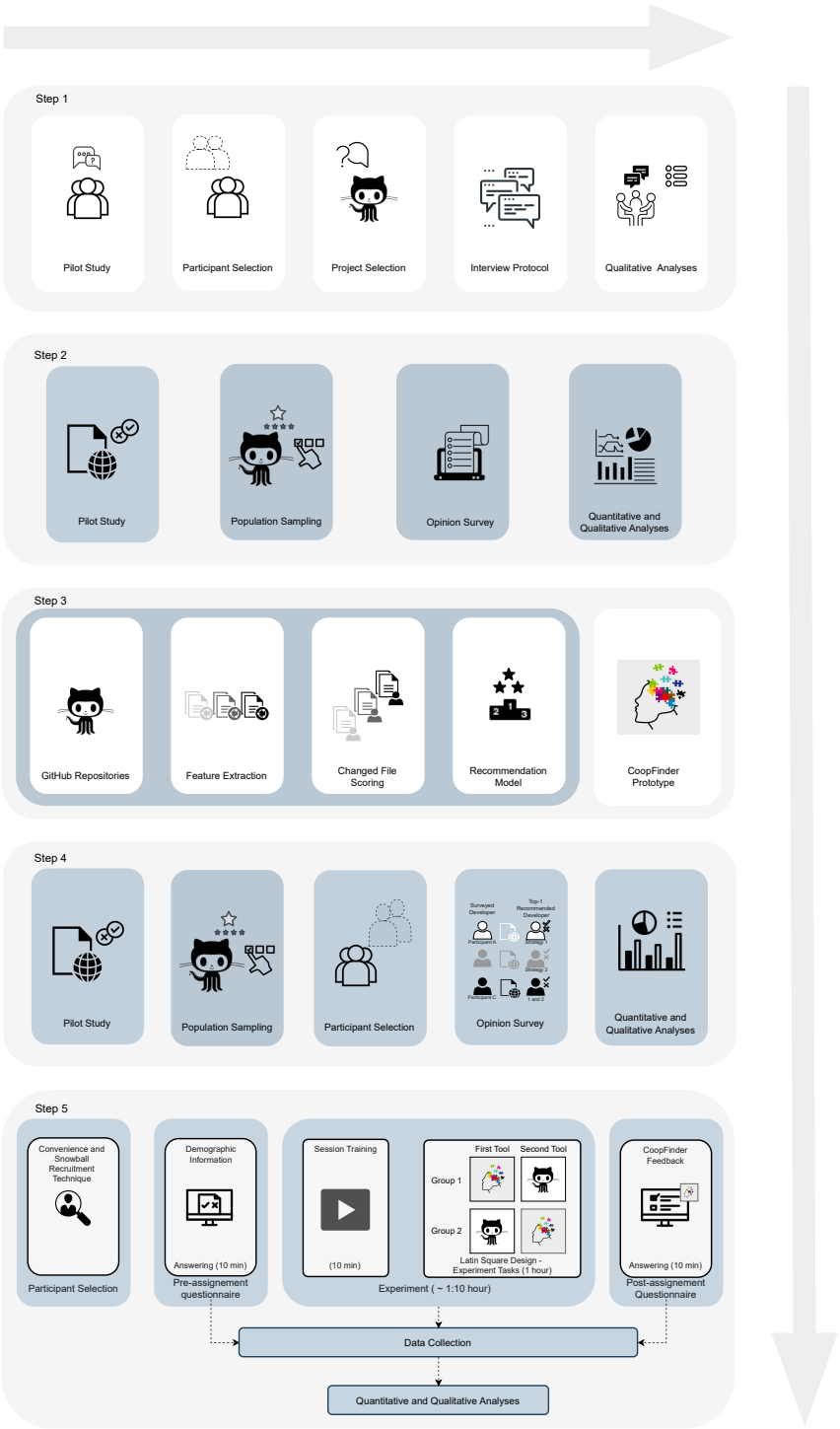

Figura 3. Overview of the PhD research steps.

mendation tool, and GitHub. We set the goal of our study using the Goal/Question/Metric (GQM) template [Basili and Weiss 1984]. We answered the some RQs applying Hypotheses tests. Besides, we analyzed and answered others qualitatively using standard coding techniques [Corbin and Strauss 2014, Creswell and Creswell 2017]. Last, we submitted this research for the Ethical Committee of our institution before performing this study (CAAE: 55476922.0.0000.5149).

5. Contributions and Publications

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

1. *Understanding Collaborative Software Development: An Interview Study*. 15TH IEEE/ACM International Conference on Global Software Engineering (ICGSE), Seoul, South Korea, 2020. [Constantino et al. 2020].
2. *Perceptions of Open-Source Software Developers on Collaborations: An Interview and Survey Study*. 2021. Journal of Software: Evolution and Process (JSEP), page e2393. [Constantino et al. 2021].
3. *CoopFinder: Finding Collaborators Based on Co–Changed Files*. 2022. IEEE Symposium on Visual Languages and Human–Centric Computing (VL/HCC), Rome, Italy, 2022. [Constantino and Figueiredo 2022].
4. *Dual Analysis for Helping Developers to Find Collaborators Based on Co–Changed Files: An Empirical Study*. 2023. Software: Practice and Experience (SPE). doi: 10.1002/spe.3194. [Constantino et al. 2023].
5. *Recommending Collaborators Based on Co–Changed Files: A Controlled Experiment*. 2023. XVIII Simpósio Brasileiro de Sistemas Colaborativos (SBSC 2023).

Our work (1) has been recognized with an honorable mention at the prestigious ICGSE/2020 conference. This conference is renowned worldwide for its focus on software engineering processes and globally distributed software development. In recognition of the quality of our work (1), we were invited by ICGSE/2020 to contribute with work (2) to a special issue in the Journal of Software: Evolution and Process (*Impact factor (2021):1.864*). Furthermore, our work (3) has been accepted for presentation at the IEEE Symposium on Visual Languages and Human-Centric Computing, widely recognized as the premier international forum for research on this topic. Another significant accomplishment is the publication of our work (4) in the journal of Software: Practice and Experience (*Impact factor (2021):3.200*), which is highly respected for its contributions to the practical application of software techniques and tools for both software systems and applications. We are also delighted to report that our most recent work (5) has been accepted for submission at a national conference and is currently under submission process. We are optimistic about the potential impact of this work and look forward to sharing the results with the broader research community.

Furthermore, our work provided us with the opportunity to visit the Institute of Software Research (ISR) at Carnegie Mellon University (CMU) in Pennsylvania, United States from October 2018 to March 2019. During this period, we had the privilege of being supervised by Professor Christian Kästner. Our exchange program was made possible with the support of the *Programa de Doutorado Sanduíche no Exterior* (PDSE) from CAPES grant 88881.189537/2018-01.

6. Summary of the Work and Contributions

Software developers must collaborate at all stages of the software life-cycle to create quality software systems. However, for large projects with hundreds of dynamic developers, such as several successful open–source projects, it can be very complex to find developers with the same interests and, thus, gain suitable collaborations and new insights. Resources and efforts may be wasted in the project context, discouraging many developers from staying. It can be costly to manage so many contributions, which is another question for the maintainer who wants to take advantage of this small, modest, but useful contribution made by a volunteer developer in the shortest possible time. Thus, this section summarizes the results of this work, regarding its five specific goals, as follows.

- **SG1** Investigate the motivations, processes, interactions, and barriers involved in collaboration during open–source software development.
- **SG2** Investigate how open the developers are for collaboration with others.
- **SG3** Provide tool–supported strategies based on co-changed files to find suitable collaborators.
- **SG4** Evaluate developers recommendations based on co-change files from point of view of who receives the recommendations.
- **SG5** Evaluate developers tool–supported strategies from the point of view of maintainers and developers (`GitHub`user and non-user).

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

- Collaboration transcends coding, and includes documentation and management tasks.
- The collaboration process has different nuances and challenges when considering members of the core team interacting with each other and members of the team interacting with peripheral developers. Collaboration is heavily driven by issue management, and management skills impact it in defining, categorizing, and sizing tasks accordingly, in such way that the community (including newcomers) can collaborate independently.
- Knowledge management is a challenge in collaboration, and it is important to carefully define communication policies to mitigate and avoid problems related to knowledge retention and decentralization.

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

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

- Despite personal preferences to work independently, some developers still consider collaborating with others in some scenarios, especially in development tasks.

The findings from SG1 and SG2 are inputs for the next step related to the SG3, which proposes tool-supported strategies to help developers find collaborators in the open-source projects. Thus, we propose two developer recommendation strategies based on coding activities, especially in co–changed files, that is, modifications made by developers on the same file. This set of files can indicate that developers have interests and familiarity with a specific part of the project, impacting directly on collaborative work among developers. To extract these changes, we used the number of commits for STRATEGY 1. For STRATEGY 2, we used the number of lines of changed code (code churns). Furthermore, we proposed COOPFINDER, a visual and interactive tool that implements the two strategies (STRATEGY 1 and 2) to connect collaborators based on a set of files of their interest.

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

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

Finally, for SG5, we conducted a controlled experiment to evaluate the developer recommendation strategies and the COOPFINDER. This user evaluation concerned usability and user satisfaction involving 35 participants, of which 18 were `GitHub`users, and 17 were non-users. All of them are maintainers and/or developers of software projects. As required, the study was submitted and approved to the Brazilian Committee for Ethics in Research[2]. Some interesting findings from SG5 are:

[2]Protocol CAAE:55476922.0.0000.5149

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

7. Acknowledgments

This research was partially supported by Brazilian funding agencies: CAPES (88881.189537/2018-01) and FAPEMIG (Grant PPM-00651-17).

Referências

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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