How to choose a task? Mismatches in perspectives of newcomers and existing contributors

Fabio Santos
Northern Arizona University
fabio_santos@nau.edu

Bianca Trinkenreich
Northern Arizona University
bianca_trinkenreich@nau.edu

João Felipe Pimentel
Northern Arizona University
joao.pimentel@nau.edu,

Igor Wiese
Universidade Tecnolóogica Federal do Paraná
igor@utfpr.edu.br

Igor Steinmacher
Northern Arizona University
Igor.Steinmacher@nau.edu

Anita Sarma
Oregon State University
anita.sarma@oregonstate.edu

Marco A Gerosa
Northern Arizona University
Marco.Gerosa@nau.edu

ABSTRACT

[Background] Selecting an appropriate task is challenging for Open Source Software (OSS) project newcomers and a variety of strategies can help them in this process. **[Aims]** In this research, we compare the perspective of maintainers, newcomers, and existing contributors about the importance of strategies to support this process. Our goal is to identify possible gulfs of expectations between newcomers who are meant to be helped and contributors who have to put effort into these strategies, which can create friction and impede the usefulness of the strategies. **[Method]** We interviewed maintainers (n=17) and applied inductive qualitative analysis to derive a model of strategies meant to be adopted by newcomers and communities. Next, we sent a questionnaire (n=64) to maintainers, frequent contributors, and newcomers, asking them to rank these strategies based on their importance. We used the Schulze method to compare the different rankings from the different types of contributors. **[Results]** Maintainers and contributors diverged in their opinions about the relative importance of various strategies. The results suggest that newcomers want a better contribution process and more support to onboard, while maintainers expect to solve questions using the available communication channels. **[Conclusions]** The gaps in perspectives between newcomers and existing contributors create a gulf of expectation. OSS communities can leverage our results to prioritize the strategies considered the most important by newcomers.

CCS CONCEPTS

• **Software and its engineering** → **Open source software**.

KEYWORDS

open source software, issue tracker, task management, newcomers, social coding platform, strategies

ACM Reference Format:
Fabio Santos, Bianca Trinkenreich, João Felipe Pimentel, Igor Wiese, Igor Steinmacher, Anita Sarma, and Marco A Gerosa. 2022. How to choose a task? Mismatches in perspectives of newcomers and existing contributors. In *Proceedings of ESEM (ESEM'22)*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Selecting an appropriate task to contribute is one of the most challenging but crucial steps for newcomers joining open source software projects [25, 27–29, 37]. This is a known problem, and projects employ various strategies to assist them in finding starter tasks. Existing work presents a compendium of such strategies [11, 17, 31]. For example, research has proposed labeling issues that signal newcomer friendliness (e.g., starter task, newcomer task, good first issue) [34] as an important strategy to aid newcomers in identifying tasks they can undertake [1]. Others have proposed mechanisms that aid newcomers in understanding the issue to be solved [18]. Most of these strategies are based on research on newcomer barriers [29, 31] and contributors' recommendations for overcoming them.

A missing piece in our understanding of what newcomers need is how newcomers' perspectives fit in with those of existing contributors. A discrepancy between these two perspectives—newcomers and existing contributors—can create a gulf of expectations. Such a gulf, in turn, means that the projects' strategies are less likely to succeed, and newcomers continue to struggle.

Our goal in this paper is to investigate the difference in perspectives of newcomers and existing contributors in (i) the strategies newcomers use to choose a task and (ii) the strategies communities need to employ to support newcomers. Exploring the different perspectives can help OSS communities devise tailored strategies that match newcomers' needs.

We aim to answer the following research questions:

RQ1. What strategies help newcomers choose a task in OSS?

RQ2. How do newcomers and existing contributors differ in their opinions of which strategies are important for newcomers?

We conducted a qualitative study based on interviews with maintainers (n=17) to identify strategies that people playing a maintainer role consider important to assist newcomers. We focused on maintainers as they typically have knowledge and ownership of the project, and strategies they propose have a higher likelihood of being implemented. We then conducted a follow-up survey with contributors, maintainers, and newcomers (n=64) to understand to what extent different stakeholders agreed on the importance of these strategies.

Our results show that newcomers' and other contributors' perceptions are aligned for some strategies. We also found situations in which newcomers, frequent contributors, and maintainers differ in their thoughts about what newcomers need to select a task and start contributing to an OSS project. For example, frequent contributors and maintainers believed that "Understand the issue" would be the most important strategy used by newcomers. Still, newcomers had a different perspective ("set up the environment" was ranked as the most important). Regarding community strategies, newcomers and frequent contributors agreed on the importance of good documentation and project quality but differed widely about the importance of improving newcomer onboarding and contribution processes. Our results shed light on these discrepancies, and OSS communities can leverage these results to reflect on their onboarding strategies.

2 RELATED WORK

Newcomers face a variety of barriers to start contributing to open-source software, including social interaction problems, documentation issues, lack of knowledge, lack of direction for how to contribute, and high technical complexity [29]. One of those barriers is related to choosing a task to start contributing to in an OSS repository [28]. Sometimes it is not clear if the contributor is able to contribute to the issue reported [28, 29].

Recently, the literature proposed strategies like issue recommendation, different mitigation processes (addressing technical hurdles, social hurdles, and toxic environment), maintainer empowerment, issue labeling, and badges to aid newcomers. These strategies focus on communities [8, 9, 26, 36], contributors [30], or specific niches [8]. They aim not only to assist newcomers in finding a task, but also to attract, retain, and keep newcomers engaged. Fig. 1 summarizes the strategies to aid newcomers identified in the literature.

Community strategies. Steinmacher et al. [27] explored the social barriers and identified possible mitigation strategies that communities could employ to support newcomers, such as recommending mentors, providing automatic greetings, and offering feedback on newcomers' contributions. Later, Steinmacher et al. [26] proposed 13 strategies to help newcomers find a task through recommendations made by mentors. Among the strategies, they indicate that communities should identify the complexity and skills required for finishing a task and scaffold newcomers' skill acquisition.

Contributor strategies. Steinmacher et al. [30] devised 14 guidelines targeted at contributors and communities and divided these guidelines into three groups: social (e.g., being proactive - from the point of view of contributors), technical (e.g., documenting the code structure - from the point of view of communities), and process (e.g., finding an easy task to start with - from the point of view of contributors).

Project strategies. Guizani et al. [8] proposed a total of 48 strategies for projects. They focused on supporting the contribution in *large OSS projects*, and included a customized solution for the Apache Software Foundation - ASF. Pham et al. [16] investigated the contribution process with regard to the *testing procedures* and proposed strategies to integrate newcomers into the testing culture of the project, which includes lowering the barriers and communicating the project's culture.

Some studies advocate for employing supportive solutions as strategies for aiding newcomers in choosing tasks. One approach consists of creating a supportive environment to smoothly introduce the newcomers to the social and technical skills related to a project [18]. Other recent approaches aim to label issues to assist in the selection of new issues [11–13, 17, 38]. For example, Santos et al. [17] label issues according to the API domains of the project, intending to let users choose issues related to their skill set. Other researchers label issues as "good first issue" (GFI), with the goal of providing newcomers with information about which issues are easier for people who are new to the complexity of the project [11]. This type of labeling is encouraged by social coding platforms like GitHub[1], and practiced by several communities (e.g., LibreOffice[2], KDE[3], and Mozilla[4]).

While those strategies may help transpose the barriers, one question remains: are these guidelines and solutions addressing the problems in the way contributors expect? In this paper, we dive into this question to find if there are mismatches between maintainers' and contributors' perspectives regarding maintainers' efforts and contributors' strategies. Since the resources in projects are finite, prioritizing the communities' strategies according to the importance attributed by contributors may be important to optimize the results, save time, and reduce efforts.

3 METHOD

This section presents the design of our study, which included interviews and a survey[5], as depicted in Fig. 2.

3.1 Stage 1: Interviews - Building the strategies models

Our research goal was to understand the maintainers' perspective on (i) what strategies newcomers use to choose a task to work on; and (ii) what strategies the community can take to help a newcomer choose a task. Due to the complexity of the phenomenon under study, we employed in-depth interviews.

3.1.1 Interviews Planning. We aimed to recruit project maintainers to talk about task selection while they browsed through their issue trackers. Therefore, we looked for active projects on GitHub and

[1]http://bit.ly/NewToOSS
[2]https://wiki.documentfoundation.org/Development/EasyHacks
[3]https://community.kde.org/KDE/Junior_Jobs
[4]https://wiki.mozilla.org/Good_first_bug
[5]The research protocol was approved by the institutional review board (IRB) of the authors' institution.

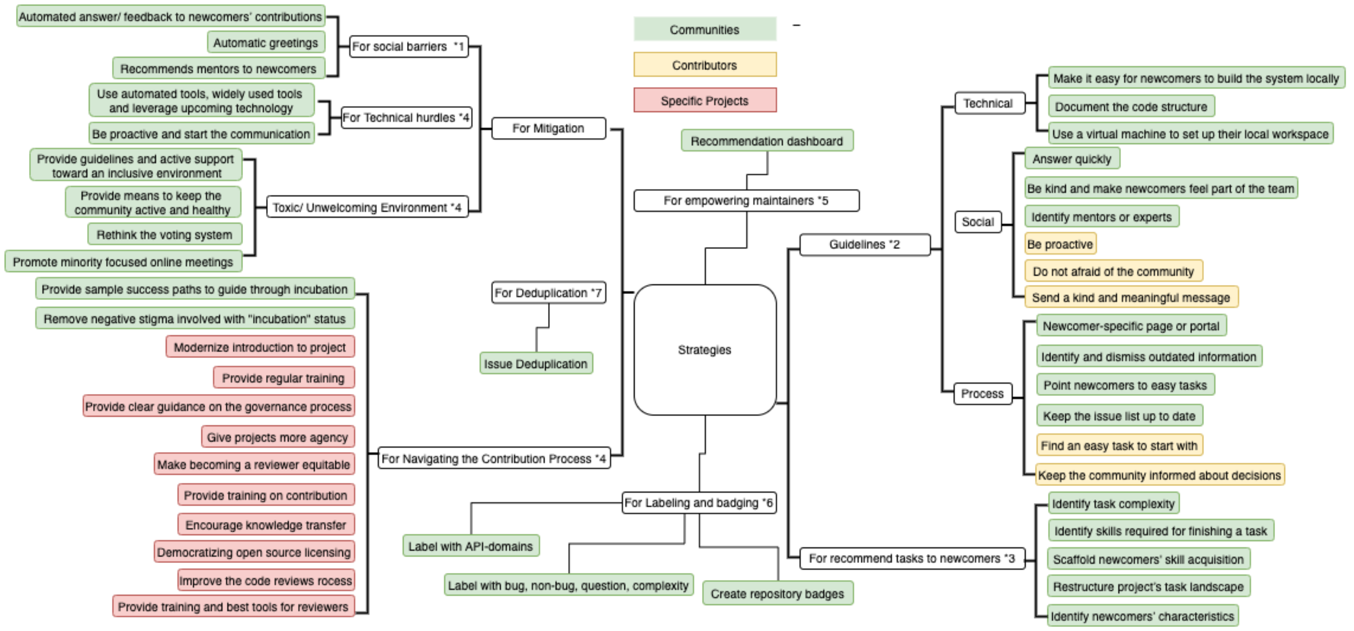
Communities
Contributors
Specific Projects
Strategies
For Mitigation
For social barriers *1
Automated answer/ feedback to newcomers' contributions
Automatic greetings
Recommends mentors to newcomers
For Technical hurdles *4
Use automated tools, widely used tools and leverage upcoming technology
Be proactive and start the communication
Toxic/ Unwelcoming Environment *4
Provide guidelines and active support toward an inclusive environment
Provide means to keep the community active and healthy
Rethink the voting system
Promote minority focused online meetings
For Navigating the Contribution Process *4
Provide sample success paths to guide through incubation
Remove negative stigma involved with "incubation" status
Modernize introduction to project
Provide regular training
Provide clear guidance on the governance process
Give projects more agency
Make becoming a reviewer equitable
Provide training on contribution
Encourage knowledge transfer
Democratizing open source licensing
Improve the code reviews rocess
Provide training and best tools for reviewers
For Deduplication *7
Issue Deduplication
Recommendation dashboard
For empowering maintainers *5
For Labeling and badging *6
Label with API-domains
Label with bug, non-bug, question, complexity
Create repository badges
Guidelines *2
Technical
Make it easy for newcomers to build the system locally
Document the code structure
Use a virtual machine to set up their local workspace
Social
Answer quickly
Be kind and make newcomers feel part of the team
Identify mentors or experts
Be proactive
Do not afraid of the community
Send a kind and meaningful message
Process
Newcomer-specific page or portal
Identify and dismiss outdated information
Point newcomers to easy tasks
Keep the issue list up to date
Find an easy task to start with
Keep the community informed about decisions
For recommend tasks to newcomers *3
Identify task complexity
Identify skills required for finishing a task
Scaffold newcomers' skill acquisition
Restructure project's task landscape
Identify newcomers' characteristics

Figure 1: Previous strategies identified in recent literature [27] *1 [30] *2 [26] *3 [8] *4 [9] *5 [36] [11–13, 17, 38] *6, [40] *7

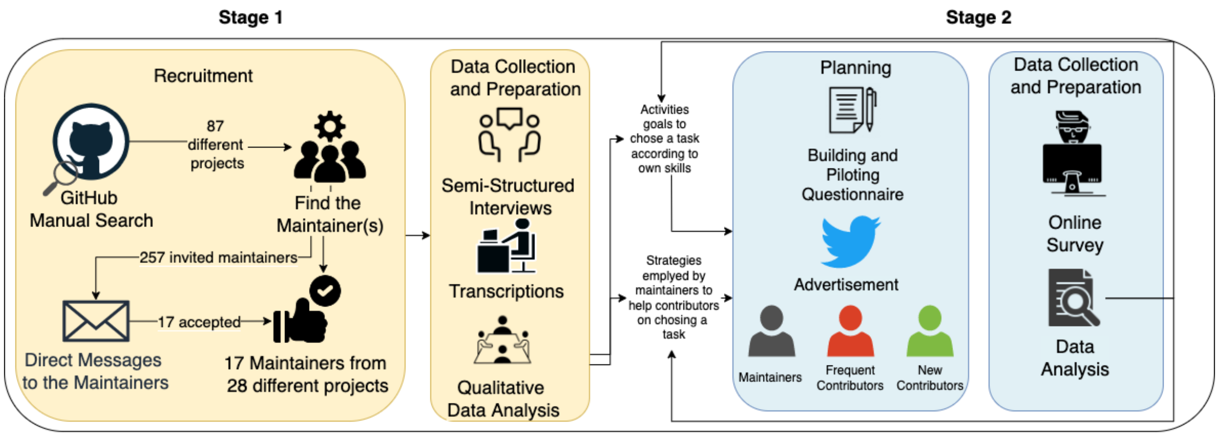
Stage 1
Stage 2
Recruitment
GitHub Manual Search
87 different projects
Find the Maintainer(s)
257 invited maintainers
Direct Messages to the Maintainers
17 accepted
17 Maintainers from 28 different projects
Data Collection and Preparation
Semi-Structured Interviews
Transcriptions
Qualitative Data Analysis
Activities goals to chose a task according to own skills
Strategies emplyed by maintainers to help contributors on chosing a task
Planning
Building and Piloting Questionnaire
Advertisement
Maintainers
Frequent Contributors
New Contributors
Data Collection and Preparation
Online Survey
Data Analysis

Figure 2: Research Method

with recent pull requests and issues. We went through the list of popular projects and analyzed the projects' metadata, including the description, number of stars, number of forks, closed issues, pull requests closed, number of contributors, number of commits, and programming languages. Our goal was to select a set of diverse projects in terms of languages, sizes, and domains. We selected 87 projects.

Then, we identified the maintainers of these projects by observing the behavior of approving or rejecting pull requests, the comments, and the auto-denominated role in their profiles or in the project repository. We selected 257 maintainers who had public attract usernames or email addresses publicly available on their profiles. We invited and interviewed them in random order until we could not find any new strategy related to our goals for three consecutive interviews. We offered interviewees a 25-dollar gift card as a token of appreciation. A consent letter was sent in advance along with a questionnaire to collect project and interviewee demographic information.

We conducted two pilot interviews to validate the script and time-box the interviews, ensuring that their duration would be about 60 minutes. Two researchers evaluated the responses and made minor adjustments to the instrument. The pilot interviews were discarded.

Our final sample comprised 17 maintainers in OSS, responsible for validating changes and performing merges in 26 different projects. This number is in line with what is foreseen in the literature as a valid number to unveil the characteristics of a study domain [2]. Table 1 presents their demographics.

3.1.2 Data Collection. We collected the data using semi-structured interviews [22]. Three researchers experienced in qualitative studies conducted the interviews using a videoconferencing tool (15) or textual chat (2). We used a script (see Table 2) to guide the different areas of inquiry, while also listening for unanticipated information during the flow of the conversation.

The interviews revolved around the central question of *"how do newcomers choose an issue, and how can the community help?"* We approached this topic after establishing rapport with the interviewee by asking about their contributions. Then, the researcher asked the interviewees to open issues from their project and show how they could be analyzed and what newcomers could do to choose a task. We used the think-aloud technique while the interviewee navigated the issue track system. Interviewees reported what strategies contributors should use to choose a task and how to prepare a project (usually using their own project as an example) to help newcomers. Despite the pre-planned script, the interviewers took advantage of the opportunities that emerged during their conduction, using the principle of flexibility to obtain extra data [22]. A concluding part of the interview sought to obtain additional information and pointers to other potential respondents (snowballing). Two respondents were recruited using these leads.

With participant consent, we recorded all interviews. The first author of this paper transcribed the interviews, which lasted between 45 and 65 minutes. We used OTTER.AI and listened to each recording, adjusting the corresponding transcriptions, mainly regarding technical terms and project names.

Our sample comprised maintainers across 26 different OSS projects, including Spark, Apache Cordova, Brain.js, Microsoft PowerToys, Prisma, Azure Data Studio, ggplot2, Presto, bookdown, Godot Engine, Oppia, Jina.ai, Turn, and CASA. Some interviewees maintain more than one project. These projects vary in terms of the number of contributors (30 to 1,791 contributors), product domains (including infrastructure and user-application projects), and types (backed by foundations, communities, and companies). Table 1 presents the demographics of our sample. Because of the terms of consent, we cannot link each participant to their projects.

3.1.3 Data Analysis. We qualitatively analyzed the transcripts of the interviews by inductively applying open coding in groups. We built post-formed codes as the analysis progressed and associated them with respective parts of the transcribed text. The codes revealed strategies according to the participants' perspectives, who were identified as P1 to P17.

After identifying the strategies, we grouped them into a set of higher-level categories and produced two codebooks, one for newcomers' strategies to choose a task and another for the communities' strategies to help newcomers find a suitable task [6]. Three of the authors met once a week for three weeks to discuss and validate the results. The coding process was conducted using continuous comparison [33] and negotiated agreement [6] as a group. In the negotiated agreement process, the researchers discussed their rationale for categorizing each code until they reached a consensus [6].

3.1.4 Member Checking. After analyzing the strategies reported in the interviews, we conducted member checking to evaluate the validity of our interpretation and collect additional insights. We contacted via email the four participants who had agreed to a follow-up meeting (P2, P4, P14, and P16), sending them an editable visual representation of the description of each strategy. Participants could give feedback by email, annotating the visualization directly, or through an online meeting. Participants P14 and P16 scheduled a virtual meeting, whereas P2 and P4 gave their feedback over email. The virtual meetings lasted about 15 minutes. During the call, we explained the overall definitions of the first level of Fig. 4 and 5, and asked for suggestions. The email had two questions: What do you think about this model? Did we correctly place your view in the model? The four participants (P2, P4, P14, and P16) verified that the strategies we had generated reflected their views. We identified some misunderstandings related to some terms and updated our model to make them clearer.

Table 1: Interview demographics (n=17) P* Prefer not to say

Participant ID	Years of Experience	Gender	1st contribution	Team member since
P1	20	M	2004	2007
P2	8	M	2014	2014
P3	15	M	2016	2017
P4	8	W	2014	2014
P5	8	M	2014	2016
P6	15	M	2006	2018
P7	3	M	2018	2019
P8	10	M	2018	2018
P9	7	M	2016	2016
P10	10	P*	2015	2015
P11	3	M	2018	2020
P12	2	M	2018	2020
P13	7	M	2017	2020
P14	15	M	2005	2017
P15	4	M	2017	2017
P16	20	M	2008	2019
P17	8	W	2016	2020

Table 2: Interview Script (excluding demographic questions)

[RAPPORT]
Q1 - What are the last projects that you contributed/maintained?
Q2 - What are your areas of expertise?
Q3 - In these projects, how did you choose tasks that fit your expertise?
[SKILL TYPES]
Q4 - Look at the issue #XX. Suppose you are a newcomer: How can you choose a task to contribute? Can you, please, think aloud so that we know what you are thinking

3.2 Stage 2: Survey - Understanding the relative importance of the strategies

We conducted an online survey to obtain the perspectives of a variety of developers on the strategies identified during the interview.

3.2.1 Survey Planning. In the survey, we present the newcomers' strategies for choosing an issue and the strategies that communities use to help them. We asked respondents to rank the relevance

[6]https://doi.org/10.5281/zenodo.6508776

of each strategy. We also included demographic questions about experience, age, gender identity, and country of residence.

We advertised the survey on social media and community blogs (e.g., Linkedin, Twitter, Facebook, and others). We also sent direct messages to OSS contributors and discussion lists. We offered the participants a chance to enter a raffle for US$25 gift cards to encourage participation.

3.2.2 Data Collection. The survey was available between October 8 and November 2, 2021. We received 209 non-blank responses and filtered out data to consider only valid responses. We analyzed the attention check answers, time to complete the questionnaire, equal/similar e-mail addresses, and inappropriate answers to the open questions (e.g., "XXX," "No," "There is No," "N"), resulting in 64 valid responses. We present the demographics of the survey participants in Fig. 3.

3.2.3 Data Analysis. We used the Schulze method to rank the strategies and their association with groups from the demographic data [19, 39].

Schulze Method. The Schulze method [19] is an election method that computes a single ordered list of preferences (ranking of candidates) from a set of votes, in which each vote represents an ordered list of preferences on its own. That is, each voter selects all the candidates that they prefer in order, ranking them, and the Schulze method aggregates all the rankings into a single winning ranking, or optionally an ordered list with or without ties. This method is considered a Condorcet method. Hence, it prioritizes votes for candidates who win the pairwise comparisons against each candidate in every head-to-head election scenario possible. This election method has been used for elections and decision-making processes by the Debian project, Ubuntu, Gentoo, the Wikimedia Foundation, political parties, and others [20]. In our case, we combined the rankings provided by the survey participants to find which strategies have higher relative importance for each group.

Schulze Setup. The Schulze configuration considers the ordered preference of the factors each participant selected that define the relevance of the strategies. In our case, the strategies identified in the previous interview stage are the factors. We created the ballot list by aggregating the number of times each ranking order was chosen. We used the R package "votesys"[4] to compute the list of the most voted strategies using the Schulze method.

See supplemental material [7] for the questionnaire, codebooks, and sample answers.

4 RESULTS

In this section, we present the results of our investigation grouped by research question.

4.1 RQ1: What strategies help newcomers choose a task in OSS?

To answer this research question, we interviewed maintainers to understand their perspectives on (i) what strategies a newcomer uses to choose an open issue; and (ii) what strategies the OSS communities can use to help newcomers choose tasks.

[7]https://doi.org/10.5281/zenodo.6508776

4.1.1 Newcomer strategies to choose a task. From the interviews, we could identify 27 strategies that maintainers expect newcomers to use to choose a task and grouped them into five categories, as presented in Fig. 4. In the following, we present more details about our findings, organized by strategy category.

Understand the issue. According to the maintainers, newcomers should understand the issues beyond their titles. The specific strategies under this category are presented in the first column of Fig. 4. The main focus for newcomers is finding signals to help them match their skills with appropriate issues. In this sense, reading through all the issues' information (title, description, comments) and checking issue labels, type (bug/feature), and keywords help newcomers to find meaningful signals relevant to solving the issue (e.g., class names, method names, component, library, etc.). For example, one interviewee mentioned that if the newcomers want to learn if the issue is interesting for them *"that can be concluded from reading the entirety of the proposal and reading the discussion about it. And then the actual code fix is very simple"* (P12).

Communicate with the community. Maintainers mentioned the importance of communicating with the community as part of the decision process. A newcomer who does not completely understand an issue should contact to receive support from the community. Specific strategies related to it include (i) posing questions to maintainers or other contributors and (ii) staying in touch with the community to learn about the project and project roles. As P10 stated: *"usually you can figure out it [...] by talking to other contributors or peers".*

Understand the context. To choose a task, maintainers highlighted that it is important for newcomers to know the context of the problem. Newcomers need to capture details that may help them to define a solution. To do this, newcomers need to read and have a high-level understanding of the codebase and the libraries used. Furthermore, it is suggested that they understand aspects related to the software architecture, like dependencies and configuration files. Maintainers also reported that newcomers should attempt to foresee the scope of the change. If newcomers are not able to understand the context where the issue is, they will probably change more code than necessary to solve the issue, increasing the chance of introducing new bugs: *"The point is, you should know what feature we are working on."*(P13). Fig. 4 (third column) lists the strategies related to understanding the context.

Set up the environment. To prepare a solution and submit a pull request, first, the newcomer must identify the appropriate tools to build the software locally. "Set up the environment" is a landscape exploration task since the contribution guidelines documentation is not always up-to-date or does not comprehend all possible operating systems, library versions, and other details. It is also a playground to understand configuration files and the project structure. Contributors need to try to set up the environment to check their skills before proceeding. In addition, reproducing the error is part of the process, as P5 witnessed: *"looking at the debugger [...] we can get clues of what's happening. But for sure, we will want to reproduce it."*(P5). The fourth column of Fig. 4 shows the strategies under the Setup environment category.

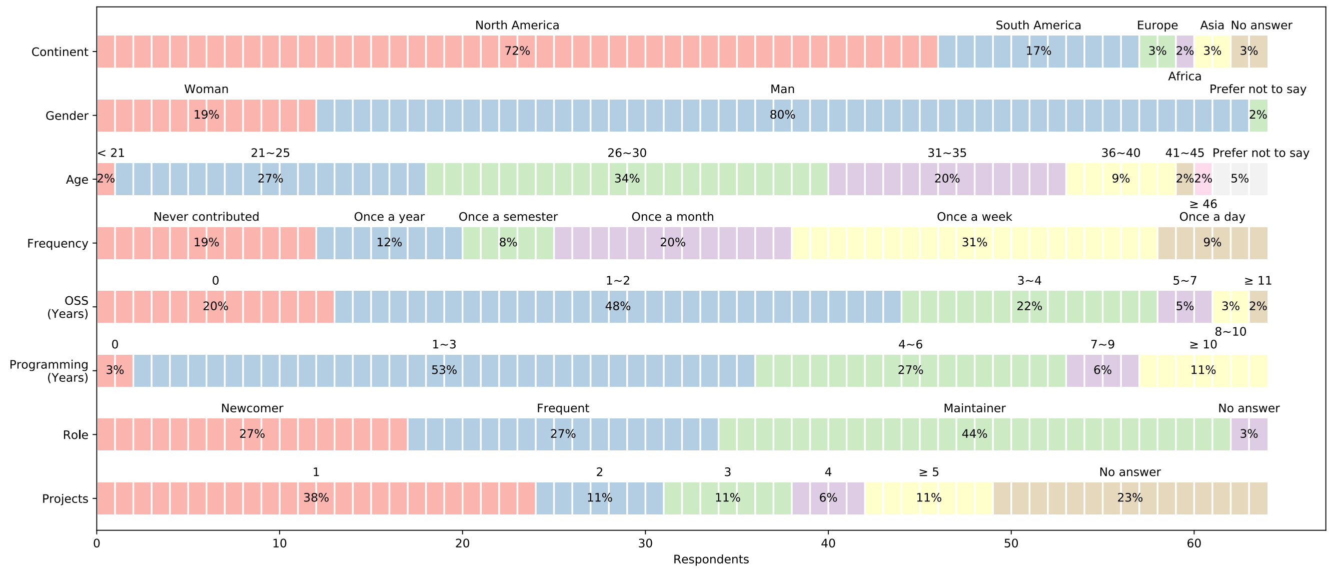

Figure 3: Personal characteristics of the survey respondents (n=64)

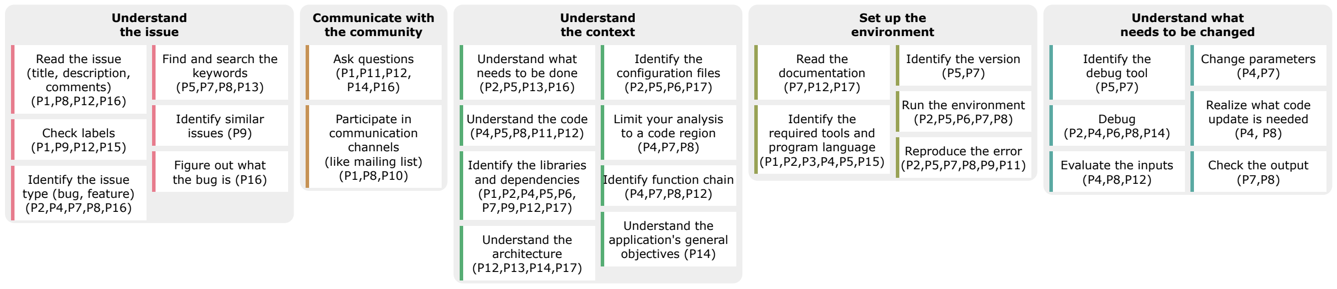
Understand the issue
Read the issue (title, description, comments) (P1,P8,P12,P16)
Check labels (P1,P9,P12,P15)
Identify the issue type (bug, feature) (P2,P4,P7,P8,P16)
Find and search the keywords (P5,P7,P8,P13)
Identify similar issues (P9)
Figure out what the bug is (P16)
Communicate with the community
Ask questions (P1,P11,P12, P14,P16)
Participate in communication channels (like mailing list) (P1,P8,P10)
Understand the context
Understand what needs to be done (P2,P5,P13,P16)
Understand the code (P4,P5,P8,P11,P12)
Identify the libraries and dependencies (P1,P2,P4,P5,P6, P7,P9,P12,P17)
Understand the architecture (P12,P13,P14,P17)
Identify the configuration files (P2,P5,P6,P17)
Limit your analysis to a code region (P4,P7,P8)
Identify function chain (P4,P7,P8,P12)
Understand the application's general objectives (P14)
Set up the environment
Read the documentation (P7,P12,P17)
Identify the required tools and program language (P1,P2,P3,P4,P5,P15)
Identify the version (P5,P7)
Run the environment (P2,P5,P6,P7,P8)
Reproduce the error (P2,P5,P7,P8,P9,P11)
Understand what needs to be changed
Identify the debug tool (P5,P7)
Debug (P2,P4,P6,P8,P14)
Evaluate the inputs (P4,P8,P12)
Change parameters (P4,P7)
Realize what code update is needed (P4, P8)
Check the output (P7,P8)

Figure 4: How newcomers choose their tasks (according to the maintainers).

Understand what needs to be changed. Once the newcomer has set up the environment and realized the overall architecture and extent of possible updates that need to be made, it is time to dig deeper and identify application behavior by changing inputs and verifying how outputs respond to changes. A debugging tool is really useful here, because even having a general idea about the context, the code can often be complicated. When the code is complex, the contributors must analyze the values of the variables and run the code step by step, also changing the values of the parameters of the function. Maintainers claim that once newcomers have a general understanding of the underlying logic, they will be confident about the task. *"... this exception is happening, because somebody added this line. Okay, well, what happens if I remove this line? Does it work? Does something else break? Where is this line used?"* (P8).

4.1.2 *Community strategies to facilitate task selection.* In addition to the strategies that newcomers are expected to take, we found 40 strategies that the communities take to help newcomers choose their tasks. From these strategies, we derived seven categories of strategies, as presented in Fig. 5.

Have good documentation. A way that the community can support newcomers is by providing appropriate documentation. It is important, for example, to arrange guidelines that contain the necessary information to help them understand the contribution process, standards, and how to contact the community. In addition to traditional documentation, providing tutorials to cover crucial aspects related to project architecture and technology is also important. Pointing newcomers to good examples to be followed (issues, commit messages, etc.) is another point. Participant P2 pointed out that *"the minimum entry-level is just a knowledge of software engineering. Python in this case, and then just following the tutorial, so some patience basically to understand the documentation and so on"*(P2).

Have good communication. This strategy is fundamental to making newcomers feel welcome and comfortable discussing problems not covered by the documentation. Having channels specific for onboarding questions and asking for help when starting is something that communities may put in place. One of our interviewees confirmed the importance of the community showing good/appropriate communication skills: *"I see a lot on GitHub, big, big projects with*

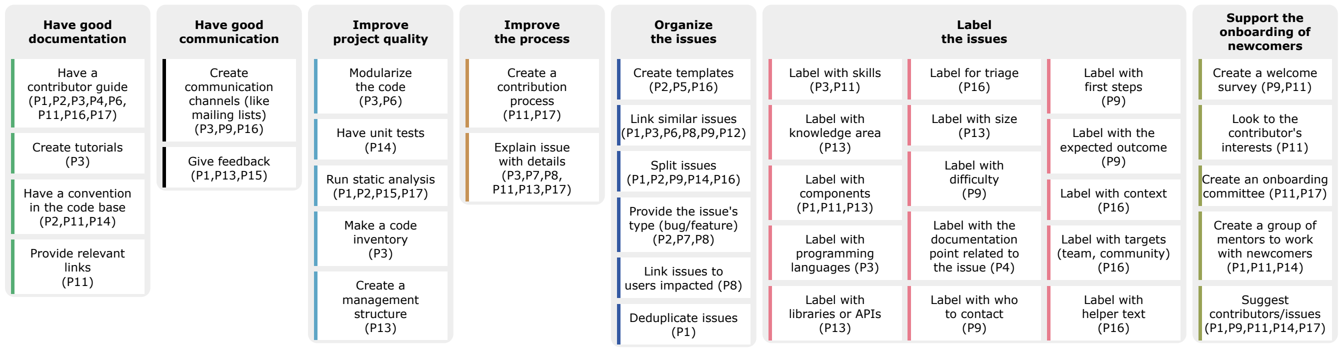
Have good documentation
Have a contributor guide (P1,P2,P3,P4,P6, P11,P16,P17)
Create tutorials (P3)
Have a convention in the code base (P2,P11,P14)
Provide relevant links (P11)
Have good communication
Create communication channels (like mailing lists) (P3,P9,P16)
Give feedback (P1,P13,P15)
Improve project quality
Modularize the code (P3,P6)
Have unit tests (P14)
Run static analysis (P1,P2,P15,P17)
Make a code inventory (P3)
Create a management structure (P13)
Improve the process
Create a contribution process (P11,P17)
Explain issue with details (P3,P7,P8, P11,P13,P17)
Organize the issues
Create templates (P2,P5,P16)
Link similar issues (P1,P3,P6,P8,P9,P12)
Split issues (P1,P2,P9,P14,P16)
Provide the issue's type (bug/feature) (P2,P7,P8)
Link issues to users impacted (P8)
Deduplicate issues (P1)
Label the issues
Label with skills (P3,P11)
Label with knowledge area (P13)
Label with components (P1,P11,P13)
Label with programming languages (P3)
Label with libraries or APIs (P13)
Label for triage (P16)
Label with size (P13)
Label with difficulty (P9)
Label with the documentation point related to the issue (P4)
Label with who to contact (P9)
Label with first steps (P9)
Label with the expected outcome (P9)
Label with context (P16)
Label with targets (team, community) (P16)
Label with helper text (P16)
Support the onboarding of newcomers
Create a welcome survey (P9,P11)
Look to the contributor's interests (P11)
Create an onboarding committee (P11,P17)
Create a group of mentors to work with newcomers (P1,P11,P14)
Suggest contributors/issues (P1,P9,P11,P14,P17)

Figure 5: Community strategies to help newcomers finding a suitable issue.

tons of issues. And they take a lot of time to react to comments... And I think that this engages a lot. I think that you have to give time for people to figure it out. But keeping them weeks or months without an answer usually would be too much"(P13).

Improve project quality. Quality improvement aims to find easier ways to fix and evolve the code. One strategy that helps is by having the code organized clearly and explicitly in a modular way. This is mentioned by P3, who said: *"That is yet convenient here: the code of <software name> is structured by module, and each module has a folder."* This facilitates, among other things, locating the pieces of code related to specific issues and features. Keeping the code covered with unit tests and providing static analysis tools make it convenient for newcomers to understand if their code is following the standards.

Improve the process. Improving the process was mentioned by participants both in terms of (i) creating a contribution process so newcomers can *"learn where they can contribute"* (P1) by *"going through the official onboarding process"* (P11). At a more granular level, the process of creating a task should guarantee that the proponent is going to (ii) explain the issue with details because *"if the domain knowledge is missing, it is a lot harder for someone to join in"* (P3). Besides explaining the issue, details can also include previous solution attempts and results, so a contributor is aware of previous strives and avoids rework (P8).

Organize the issues. Issue organization benefits the newcomers and the overall team by helping the project prioritize and allocate the right resources to the right issues. Regarding the issue itself, (i) creating a template help to standardize details and guide the author of an issue to fill the expected data (P16). The template can include (ii) a link to similar issues, which is a detail that can *"inspire the contributor"* (P9) on how to solve the issue. The strategy to (iii) split issues avoids driving newcomers away due to complexity. The issue can have *smaller sub-issues that can be taken by new people, and subsequently added everything, so that we can close the issue in the end* (P9). When it comes to the issue tracker, (iv) providing the issue's type that could be used on a filter helps newcomers to reduce the number of choices from a long list if they would prefer to work on a specific type of issue (e.g., feature request or backtrack) (P7). Not only the type but knowing the (v) impacted users can entice newcomers to pick a task by *"knowing how many people are facing this issue"*. Although requiring a manual effort, (vi) *deduplicate issues* (P1) is a strategy to avoid rework by having more than one issue to the same task.

Label the issues. Labeling could be part of the issue organization due to its straight relationship. In fact, labels can be a way to provide the issue's type, indicating whether they represent bug reports, feature requests, or other types of tasks. However, due to the great number of ways of labeling identified by the interviewers, we decided to create a category for the labels' strategies. Our participants mentioned they would like to have labels with *"specific skills would be required to solve the issue"* (P11), for example, *"skill: documentation' or skill: ruby"* (P3). Regarding the technical skills, participants brought out the need to have labels with knowledge area (P13), components (P1, P11, P13), programming language (P3), and libraries or APIs (P13).

The status of the issue can be part of a label that shows if an issue is still in triage or even under ongoing work by another contributor (P16)—in that case, a newcomer can decide to join and collaborate. Both size (P13) and difficulty level (P9) were mentioned in terms of effort and complexity. As for assistance in understanding the issue, P4 recommended having a label to the point of documentation related to the issue, so newcomers can have a better picture of the piece of software they will deal with. In case of questions, when having *"labels regarding who to contact if you need help with that issue?"* (P9), a newcomer can feel safe having someone to contact with. When coming to action, a label with first steps and expected outcomes (P9) can also be a helping hand to newcomers on the pathway to solving the issue. P16 points to the necessity of labeling the issues with the context of the project and indicating not only the target (*"if you don't have context, you don't need to know what accessibility end is. But my team needs to know what that means."*), but also including helper texts that describe the labels.

Support the onboarding of newcomers. Supporting the onboarding can include identifying the newcomers' characteristics and potential. It can be done with a survey identifying their skills and interests. An onboarding committee in charge of the integration may define the policies and goals. Knowing the newcomers' potential and interests, the community can recommend a mentor *"to make it easy for a certain person to contribute more"*(P1). The mentor may indicate some easy tasks as *"first issues which will help [the newcomer] to get familiarized with the code base"*(P11).

4.2 RQ2: How do newcomers and existing contributors differ in their opinions of which strategies are important for newcomers?

To compare the relative importance of the strategies from the point of view of different stakeholders, we used the Schulze method [19] to combine the rankings for (i) newcomers and (ii) community strategies. In the following subsections, we present the rankings and how they compare.

4.2.1 Mismatches in newcomers' strategies. Figure 6 presents the ranking of the preferences from the three groups: newcomers, frequent contributors, and maintainers. The numbers in the figure represent the position of each strategy in the combined ranking (it is possible to have ties).

Regarding strategies that newcomers are expected to use to choose a task, frequent contributors and maintainers had very similar values. However, while the Schulze method found a clear sequence of importance for frequent contributors, two ties occurred for maintainers: "Set up the environment" and "Understand the issue" tied in the first position; and "Communicate with the community" and "Understand the context" tied in the last position.

Newcomers also had similar rankings. However, they value "Set up the environment" more than "Understand what needs to be changed". The former appears in the first position, while the latter—which appears in the first position for frequent contributors and maintainers—appears in the third position for newcomers. Finally, the "Communicate with the community" was ranked in the penultimate position by all groups.

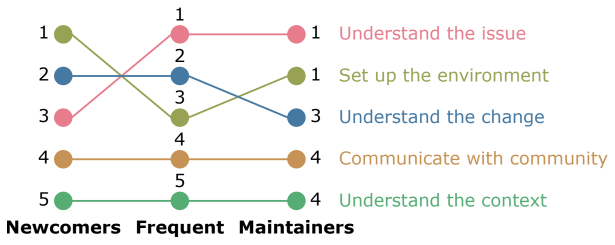

Figure 6: The relative importance of newcomer strategies

4.2.2 Mismatches in maintainers' strategies. Fig. 7 presents the combined rankings for maintainers' strategies, according to each stakeholder. Once again, the perspective of frequent contributors and maintainers are similar, with one standout difference: "Support the onboarding of newcomers".

The "Improve project quality" strategy was ranked as the top-ranked strategy for frequent contributors and maintainers. We have almost an agreement since newcomers ranked it in second place. The most important strategy according to the newcomers was "have good documentation", which is also tied as the second most important strategy for frequent contributors and maintainers. Newcomers and frequent contributors agree that "Label the issues" is the least important strategy. Maintainers also agree with its low importance, ranking it in the sixth position.

We also found some mismatches. For newcomers, "good communication" is only the fifth strategy contrasting with the second place for maintainers (tied with "good documentation") and the fourth for frequent contributors. Another mismatch regards the category "improve the process." It was ranked third according to newcomers, but its ranking dropped significantly for frequent contributors and maintainers (sixth and fifth, respectively). Still, for "Support the onboarding of newcomers," while it was ranked last for the maintainers, it was the third for newcomers and second for frequent contributors. This is surprising since the intuition we had was that maintainers should prioritize the onboarding process to count on human resources to work on the issues.

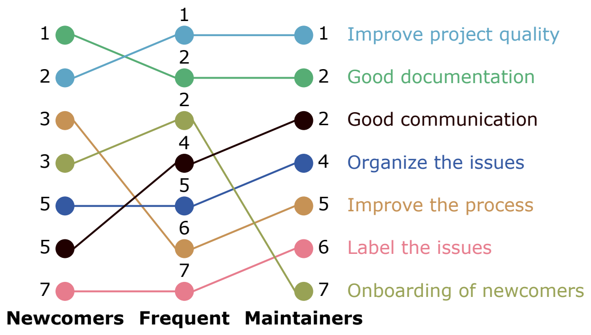

Figure 7: The relative importance of community strategies

5 DISCUSSION

Why does the convergence of relative importance matter? An OSS project is a challenging environment composed of diverse team members with a variety of experience levels and informal relations [24]. Within a dynamic organization, it is difficult to identify the competencies of each member, as people have different styles of development, are physically distant, and lack a structured working relationship [24]. In this environment, knowing the community's interests and concerns and managing to converge them can help better manage the project. For example, Steinmacher et al. [26] report difficulties mentors face in assisting newcomers. Lack of information from maintainers about newcomers denies assistance and makes the prioritization of the onboarding process harder.

In our results, we found a high convergence between frequent contributors and maintainers, both in terms of which strategies newcomers use to choose a task and strategies communities can use to support newcomers in choosing a task. However, newcomers have different interests and concerns. This discrepancy between perspectives might create a gulf of expectations and misunderstandings, making newcomers struggle, and maintainers mismanage the project with ineffective strategies.

Maintainers and contributors fight the same battles from different perspectives. Although contributing to a project is an overarching goal shared by everyone, maintainers and newcomers have different objectives. Maintainers are concerned with keeping the project running smoothly, attending to their customers, and managing the workload. On the other hand, newcomers may be looking for the benefits of the contribution to their career or the project directly. Thus, easy access to technical tips through documentation and project quality verification plays a special role when looking for a task to start with.

A recent study about the shifts in motivation [7] confirms the diversity of reasons that newcomers join and senior developers keep contributing. The first group wants to learn and aims to improve the career (indeed, the learning process may leverage the career–extrinsic motivation [7]). Therefore they are thirsty for projects with good documentation, whereas the experienced contributors want quality over documentation and good communication channels to ask questions. They aim for altruism or ideology (intrinsic motivations) [7]. Since regular contributors believe the onboarding process is a priority, altruism might direct them to help newcomers.

A study for the Linux Kernel OSS project [41] shows the number of files and commits particularly grows in some modules, while the flow of joiners is stable or even drops. Also, the maintainers' effort increased with author churn [41]. This is particularly observed in many OSS projects. As they cannot count on more newcomers and face team churn, investing time in documentation and quality seems to be aligned with our results.

Maintainers have a deep knowledge of their projects and the ideology they implement. Therefore, the main important task is to improve the project quality and understand the issue's content. As high-ranked officers, they know the battlefield. On the other hand, rookies carefully assess the environment before engaging in a project. Therefore, the ability to set up the environment is crucial to the first contribution.

The work of den Besten et al. [5] shows evidence that open-source project allocation is influenced by code characteristics and complexity. One may be able to assess the skills and the complexity level of a task by looking into the documentation, figuring out how to set up the environment, and identifying the complexity of the change. Newcomers like to start with a specific kind of problem, involving a less complex, contained, and low workload [34]. Sarma et al. [18] proposed BugExchange: a tool to help newcomers find a task while pointing to related documentation, recommend issues, and communicate with near-peer mentors. The idea behind the tool is to create a learning environment and to aid newcomers to climb the issues' complexity step by step.

As contributors mature and become frequent contributors, they navigate project issues and find the resources they need. In fact, the results showed a decrease in the priority of the "set up the environment" strategy.

Multi-teaming needs documentation and collaboration support. OSS projects usually are multi-teaming (i.e., projects whose members work on multiple projects simultaneously). Therefore, multi-teaming and OSS research have a common ground. Multi-teaming research corroborates the idea brought by OSS communities. The plurality of members may leverage the knowledge inside the project, but, on the other hand, it can hamper coordination and fragment the team's attention [10]. A proposed solution for multi-teaming is the use of information systems to support collaboration and a central repository (i.e., platforms like GitHub) for knowledge modeling or specific tools like the dashboard proposed by Guizani et al. [9]. The information systems may be seen as a document repository and a collaboration platform to assist team members in addressing the shared cognition problem by enabling information flow [10]. Since newcomers seek knowledge and tasks to which to contribute, it meets our findings for good documentation and a quality project. Maintainers, as project managers, must be aware of contributors' needs and prepare the project's repository to meet contributors' expectations. Our findings suggest that maintainers should invest in well-written documentation, a communication channel for the team, and project quality improvement.

Leveraging related research is perhaps a good way to avoid reinventing the wheel. Multi-teaming research may borrow ideas to address the team management encompassing strategies to integrate newcomers, manage the quality, and prepare the project to use a contribution process suitable to dynamic teams with high churn volume, difficulties of communications, flexible hierarchy, and diverse levels of members commitment [10].

The paradox of choice. This paradox emphasizes the greater the number of options we have, the less satisfaction we will derive from our decisions [21]. When newcomers open an issue tracker list and encounter many open issues, they can struggle to find the most suitable task to contribute to and often give up. When people do not have a strategy to elect viable choices, a decision can become overwhelmed by the options, reducing the likelihood of making a good choice and leading to frustration [21]. Strategic thinking involves planning and thinking. Planning includes analysis and procedures, whereas thinking involves synthesis—encouraging intuitive, innovative, and creative thinking [32]. While the list of issues will continue to exist and, in many cases, as a long list, our results suggest strategic thinking to help a newcomer when choosing a task to contribute. We provide suggestions for both the newcomer (Section 4.1.1) and the community (Section 4.1.2) to mitigate the paradox of choice in the issue tracker list.

Strategies used in practice or suggested by maintainers are not well-documented in the literature. Despite the recent literature covering several of the strategies that maintainers can use to support newcomers, we are surprised that after many papers about the topic, we still found other strategies. For example, as presented in related work (Section 2), GitHub projects employ many labeling strategies, such as "Label with Components".[8] However, we found other approaches. For example, labels for knowledge area, expected outcome, and context were proposed by our interviewees.

We also found some new strategies to address the organization of the issues. For example, it would be interesting to link the issues with stakeholders who would benefit or be impacted (not developers working on it). This would be valuable since the business unit or customers interested in the solution of the issue would be explicit.

The importance of the newcomers' strategy "Setup the environment" is probably due to the increasing complexity of recent applications and the plurality of the configurations. Since OSS projects are not contained in a single company, configuration management (CM) is hard to pursue, creating additional challenges for this strategy [15]. Future work can address specific strategies to handle this complexity, focusing on CM for newcomers.

While the strategies proposed in the literature barely tackle which strategies newcomers should use to communicate with the

[8]https://github.com/JabRef/jabref

community, some communities' strategies may help to increase the confidence of newcomers, such as acting with kindness and putting effort to help newcomers feel part of the team and not afraid of the community [30].

6 THREATS TO VALIDITY

There are some limitations related to our research results.

Generalizability. One can argue that a majority of our interviewees identified as men. Although it is a similar distribution to typical OSS gender demographics [3, 23, 35], we could have found new insights with a more diverse distribution of gender. The strategies uncovered in our study are not meant to be exhaustive, and further research into different types of projects will likely uncover other strategies. Furthermore, we acknowledge that our sample may be biased in unknown ways, and our results are only valid for our respondents. Additionally, the results presented in this paper are related to Open Source communities. Thus, we do not expect that the strategies found in our study will be directly applicable to other software domains. Nevertheless, to allow replication of our study, we carefully describe our research method steps.

Replicability in qualitative research is hard, since human behaviors, feelings, and perceptions change over time. Merriam [14] suggests checking the consistency of the results and inferences. Consistency refers to ensuring that the results consistently follow the data and the data analysis can support all inferences. To increase consistency, we performed data analysis in pairs, which was consistently revised by two experienced researchers. We held weekly meetings to discuss and adjust codes and categories until we reached an agreement. We also performed member checking with four participants, who confirmed our interpretation with minor changes. Moreover, we provide the codebook for traceability and increase comprehensibility and repeatability.

Theoretical saturation. A potential limitation in qualitative studies is not reaching theoretical saturation. The quality, rather than the size, of the sample of participants is essential to increase our confidence in the results. In this study, we interviewed 17 participants with different perspectives and perceptions about the studied phenomenon. Our participants were diverse in terms of the number of years with OSS and roles. Further, these participants represent 26 different OSS projects of different sizes. The number of projects is higher than the number of interviewees, as some of them contribute to more than one project in parallel. The number of interviewed participants was adequate to uncover and understand the core categories in a all-defined cultural domain or study of lived experience [2]. While we cannot claim saturation, our population has helped us uncover a consistent and comprehensive account of the strategies.

Inappropriate participation. As described in Section 3.2.2, we employed several filtering and inspecting strategies to reduce the possibility of fake data; however, it is not possible to claim that our data is completely free of this threat. From the 12 answers indicating no previous contributions only two respondents had no coding experience but have informed projects they work (possible as non-coder). Since non-coder contributions are also valuable, we decided to include these answers. Some participants did not answer the name/number of the projects they contributed as it was not a mandatory answer. To verify the commitment and experience we relied on the questions: How frequently do you contribute to OSS projects? How many years contributing to OSS projects? How many years of programming experience do you have?

7 CONCLUSION

We interviewed maintainers from diverse OSS projects and identified 27 strategies (grouped in five categories) that a newcomer uses to choose a task, and 40 strategies (grouped in seven categories) communities employ to help the newcomers. Following, we surveyed maintainers, newcomers, and frequent contributors to rank the newcomers' and maintainers' strategies. Using a Schulze method, we ranked the relative importance of the strategies to elucidate which ones are seen as more relevant for contributors in different roles (newcomers, frequent contributors, and maintainers), highlighting how they diverge. We found maintainers and newcomers diverge about the importance of the process of onboarding, the improvement of the contribution process, and the team communication. Overall, stakeholders agreed on the priority of project quality, good documentation, correctly reading and understanding the problem, and identifying what changes needs to be made.

Prior works proposed several guidelines, mitigation strategies, and processes to overcome the initial barrier faced by the newcomers. Our ranking might be used to prioritize the management effort in OSS projects or support aid goals to improve the onboarding process. Strategies that converge in serving the various stakeholders can decrease existing gaps in perspectives, therefore, obviating the problem of the expectation gulf.

Future work should reach maintainers to receive feedback about how communities can adopt the strategies and how to automate them. Additional research may also propose ways to improve productivity in OSS communities by analyzing multi-team research that possibly shares problems with OSS projects like team churn and poor team coordination. Reusing mature strategies to create robust contribution processes, collaborations, and support the integration of new team members seem to have liaison with the challenges faced by the OSS communities. Additional research can also uncover the sequence and prerequisites of the strategies. Finally, another interesting future work would be handling the choice paradox by suggesting a step-by-step project-customized process to be followed by newcomers to track progress and avoid getting lost in selecting an issue.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under Grant numbers 1815486, 1815503, 1900903, and 1901031, CNPq grant #313067/2020-1. We also thank the participants who spent their time answering our interviews and survey.

This work is partially supported by CNPq/MCTI/FNDCT (grant #408812/2021-4) and MCTIC/CGI/FAPESP (grant #2021/06662-1).

REFERENCES

[1] Sogol Balali, Umayal Annamalai, Hema Susmita Padala, Bianca Trinkenreich, Marco A Gerosa, Igor Steinmacher, and Anita Sarma. 2020. Recommending tasks to newcomers in oss projects: How do mentors handle it?. In *Proceedings of the 16th International Symposium on Open Collaboration*. ACM, Virtual Conference, Spain, 1–14.

[2] H Russell Bernard. 2017. *Research methods in anthropology: Qualitative and quantitative approaches.* Rowman & Littlefield, Washington, DC.

[3] Bitergia. 2016. Gender-diversity Analysis of the Linux kernel Technical Contributions. Accessed: 2020-10-16. https://blog.bitergia.com/2016/10/11/gender-diversity-analysis-of-the-linux-kernel-technical-contributions.

[4] CRAN. 2018. CRAN Repository Policy. https://cran.r-project.org/web/packages/votesys/index.html

[5] Matthijs den Besten, Jean-Michel Dalle, and Fabrice Galia. 2008. The allocation of collaborative efforts in open-source software. *Information Economics and Policy* 20, 4 (2008), 316–322.

[6] D Garrison, Martha Cleveland-Innes, Marguerite Koole, and James Kappelman. 2006. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The Internet and Higher Education* 9, 1 (2006), 1–8.

[7] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. 2021. The shifting sands of motivation: Revisiting what drives contributors in open source. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* IEEE, IEEE, Madrid, Spain, 1046–1058.

[8] Mariam Guizani, Amreeta Chatterjee, Bianca Trinkenreich, Mary Evelyn May, Geraldine J Noa-Guevara, Liam James Russell, Griselda G Cuevas Zambrano, Daniel Izquierdo-Cortazar, Igor Steinmacher, Marco A Gerosa, et al. 2021. The Long Road Ahead: Ongoing Challenges in Contributing to Large OSS Organizations and What to Do. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–30.

[9] Mariam Guizani, Thomas Zimmermann, Anita Sarma, and Denae Ford. 2022. Attracting and Retaining OSS Contributors with a Maintainer Dashboard. *CoRR* abs/2202.07740 (2022), 5.

[10] Pranav Gupta and Anita Williams Woolley. 2018. Productivity in an era of multi-teaming: The role of information dashboards and shared cognition in team performance. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–18.

[11] Yuekai Huang, Junjie Wang, Song Wang, Zhe Liu, Dandan Wang, and Qing Wang. 2021. Characterizing and Predicting Good First Issues. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* ACM, Bari, Italy, 1–12.

[12] Maliheh Izadi, Kiana Akbari, and Abbas Heydarnoori. 2022. Predicting the objective and priority of issue reports in software repositories. *Empirical Software Engineering* 27, 2 (2022), 1–37.

[13] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket tagger: Machine learning driven issue classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, IEEE, Cleveland, USA, 406–409.

[14] Sharan B Merriam and Elizabeth J Tisdell. 2015. *Qualitative research: A guide to design and implementation.* John Wiley & Sons, Chichester, England.

[15] Stefan Meyer, Philip Healy, Theo Lynn, and John Morrison. 2013. Quality assurance for open source software configuration management. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.* IEEE, IEEE Computer Society, Timisoara, Romania, 454–461.

[16] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. 2013. Creating a shared understanding of testing culture on a social coding site. In *2013 35th International Conference on Software Engineering (ICSE).* IEEE, IEEE Computer Society, San Francisco, USA, 112–121.

[17] Fabio Santos, Igor Wiese, Bianca Trinkenreich, Igor Steinmacher, Anita Sarma, and Marco A Gerosa. 2021. Can I Solve It? Identifying APIs Required to Complete OSS Tasks. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR).* IEEE, IEEE, Madrid, Spain, 346–257.

[18] Anita Sarma, Marco Aurélio Gerosa, Igor Steinmacher, and Rafael Leano. 2016. Training the future workforce through task curation in an OSS ecosystem. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, Seattle, USA, 932–935.

[19] Markus Schulze. 2003. A new monotonic and clone-independent single-winner election method. *Voting matters* 17, 1 (2003), 9–19.

[20] Markus Schulze. 2011. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social choice and Welfare* 36, 2 (2011), 267–303.

[21] Barry Schwartz. 2004. *The paradox of choice: Why more is less.* HarperPerennial, New York, NY.

[22] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.

[23] F. Sharan. 2016. ASF Committer Diversity Survey. Accessed: 2020-10-16. https://cwiki.apache.org/confluence/display/COMDEV/ASF+Committer+Diversity+Survey+-+2016.

[24] Marissa L Shuffler and Matthew A Cronin. 2019. The challenges of working with "real" teams: Challenges, needs, and opportunities. , 211–218 pages.

[25] Christoph Stanik, Lloyd Montgomery, Daniel Martens, Davide Fucci, and Walid Maalej. 2018. A Simple NLP-based Approach to Support Onboarding and Retention in Open Source Communities. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, IEEE Computer Society, Madrid, Spain, 172–182.

[26] Igor Steinmacher, Sogol Balali, Bianca Trinkenreich, Mariam Guizani, Daniel Izquierdo-Cortazar, Griselda G Cuevas Zambrano, Marco Aurelio Gerosa, and Anita Sarma. 2021. Being a Mentor in open source projects. *Journal of Internet Services and Applications* 12, 1 (2021), 1–33.

[27] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing.* ACM, Vancouver, Canada, 1379–1392.

[28] Igor Steinmacher, Tayana Uchôa Conte, and Marco Aurélio Gerosa. 2015. Understanding and supporting the choice of an appropriate task to start with in open source software communities. In *2015 48th Hawaii International Conference on System Sciences.* IEEE, IEEE Computer Society, Kauai, USA, 5299–5308.

[29] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.

[30] Igor Steinmacher, Christoph Treude, and Marco Aurelio Gerosa. 2018. Let me in: Guidelines for the successful onboarding of newcomers to open source projects. *IEEE Software* 36, 4 (2018), 41–49.

[31] Igor Steinmacher, Igor Wiese, Ana Paula Chaves, and Marco Aurélio Gerosa. 2013. Why do newcomers abandon open source software projects?. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE).* IEEE, IEEE Computer Society, San Francisco, USA, 25–32.

[32] Gail Steptoe-Warren, Douglas Howat, and Ian Hume. 2011. Strategic thinking and decision making: literature review. *Journal of Strategy and Management* 4, 3 (2011), 238–250.

[33] Anselm Strauss and Juliet M. Corbin. 2007. *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory* (3rd ed.). SAGE Publications, Thousand Oaks, USA.

[34] Xin Tan, Minghui Zhou, and Zeyu Sun. 2020. A first look at good first issues on github. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* ACM, Virtual Event, USA, 398–409.

[35] Bianca Trinkenreich, Igor Wiese, Anita Sarma, Marco Gerosa, and Igor Steinmacher. 2022. Women's Participation in Open Source Software: A Survey of the Literature. *Transactions on Software Engineering and Methodology (TOSEM)* (2022), 35. https://doi.org/10.1145/3510460

[36] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. 2018. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th International Conference on Software Engineering.* ACM, Gothenburg, Sweden, 511–522.

[37] Jianguo Wang and Anita Sarma. 2011. Which bug should I fix: helping new developers onboard a new project. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering.* ACM, ACM, Waikiki, Honolulu, USA, 76–79.

[38] Jun Wang, Xiaofang Zhang, and Lin Chen. 2021. How well do pre-trained contextual language representations recommend labels for GitHub issues? *Knowledge-Based Systems* 232 (2021), 107476. https://doi.org/10.1016/j.knosys.2021.107476

[39] Claes Wohlin and Aybüke Aurum. 2015. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering* 20, 6 (2015), 1427–1455.

[40] Yang Zhang, Yiwen Wu, Tao Wang, and Huaimin Wang. 2020. iLinker: a novel approach for issue knowledge acquisition in GitHub projects. *World Wide Web* 23, 3 (2020), 1589–1619.

[41] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. 2017. On the scalability of Linux kernel maintainers' work. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* ACM, Paderborn, Germany, 27–37.