A First Look at Good First Issues on GitHub

Xin Tan
Department of Computer Science and Technology, EECS, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
tanxin16@pku.edu.cn

Minghui Zhou*
Department of Computer Science and Technology, EECS, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
zhmh@pku.edu.cn

Zeyu Sun
Department of Computer Science and Technology, EECS, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
szy_@pku.edu.cn

ABSTRACT

Keeping a good influx of newcomers is critical for open source software projects' survival, while newcomers face many barriers to contributing to a project for the first time. To support newcomers onboarding, GitHub encourages projects to apply labels such as *good first issue* (GFI) to tag issues suitable for newcomers. However, many newcomers still fail to contribute even after many attempts, which not only reduces the enthusiasm of newcomers to contribute but makes the efforts of project members in vain. To better support the onboarding of newcomers, this paper reports a preliminary study on this mechanism from its application status, effect, problems, and best practices. By analyzing 9,368 GFIs from 816 popular GitHub projects and conducting email surveys with newcomers and project members, we obtain the following results. We find that more and more projects are applying this mechanism in the past decade, especially the popular projects. Compared to common issues, GFIs usually need more days to be solved. While some newcomers really join the projects through GFIs, almost half of GFIs are not solved by newcomers. We also discover a series of problems covering mechanism (e.g., inappropriate GFIs), project (e.g., insufficient GFIs) and newcomer (e.g., uneven skills) that makes this mechanism ineffective. We discover the practices that may address the problems, including identifying GFIs that have informative description and available support, and require limited scope and skill, etc. Newcomer onboarding is an important but challenging question in open source projects and our work enables a better understanding of GFI mechanism and its problems, as well as highlights ways in improving them.

CCS CONCEPTS

• **Software and its engineering** → *Collaboration in software development*; **Programming teams**.

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7043-1/20/11.
https://doi.org/10.1145/3368089.3409746

KEYWORDS

Newcomers, Onborading, Good first issues, Open Source software

ACM Reference Format:
Xin Tan, Minghui Zhou, and Zeyu Sun. 2020. A First Look at Good First Issues on GitHub. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3368089.3409746

1 INTRODUCTION

Over the past decades, Open Source Software (OSS) has risen to great prominence due to its unique advantages, e.g., low cost and high quality [38]. Different from traditional software development, OSS development generally depends on contributions from volunteers from all over the world [5]. For these projects, keeping a good influx of new developers is critical for their survival, long-term success, and continuity [1]. Moreover, new developers are a source of innovation for new ideas and work procedures that the group needs [17]. On the other hand, newcomers also eagerly participate in OSS projects, driven by the factors such as extrinsic benefits (e.g., better jobs) and intrinsic motivations [8]. However, successful onboarding in OSS projects is difficult, especially for developers who are new to open source [19, 41]. Many projects have multiple years of development history, and therefore their scale and complexity may be too complicated for newcomers to master, let alone modifying the code. Even if it is a new project, if the experiences of the newcomers do not match, it is not easy to contribute successfully. These obstacles led many potential developers to fail to make their first contribution. After multiple attempts, they may gradually lose motivation, or even give up on contributing, which is a waste for both projects and newcomers.

In order to make projects more beginner-friendly, various strategies have been explored and attempted. For example, GitHub, as the world's largest community of developers [15], recommends that projects' maintainers add files such as *README.md*, *CONTRIBUTING.md*, etc., to make projects appeal to newcomers.[1] These strategies indeed can help newcomers to become more familiar with the projects. However, finding beginner-friendly tasks is still a formidable barrier for their onboarding [32]. A feasible way is to suggest suitable tasks for newcomers, e.g., tasks that can be accomplished without in-depth knowledge of the project. This idea is also applied in practice—GitHub encourages projects to tag the issues that are suitable for newcomers with labels such as *good*

[1] https://help.github.com/en/github/building-a-strong-community

first issue.[2] Although this mechanism has been applied for many years, there are still many newcomers who spend a lot of time but fail to solve such issues, see, e.g., scikit-learn: #11554[3]. It wastes the efforts of both newcomers and project members. Investigating the practices of GFIs can help us truly understand the obstacles that newcomers face when seeking tasks and the difficulties in this guiding relationship, and therefore better support newcomers' onboarding.

Previous studies related to newcomers onboarding examined the motivations of people to become an OSS member, roadmaps to becoming a core developer, or indicators of potential long-term commitment [13, 14, 43]. Some others explored various barriers for newcomers [23, 32]. However, few studies have been devoted to understanding how exactly beginner-friendly tasks are identified and what problems there are in resolving GFIs for newcomers. In this paper, we investigate good first issues (GFIs) in GitHub.[4] We start with understanding the application of this mechanism, and then analyze its effect, reveal its problems and causes, and eventually seek the good practices. In particular, the paper addresses the following research questions:

RQ1: How common do projects report GFIs? We carefully select 816 popular GitHub projects and identify the GFIs in each project. We analyze the use of this mechanism over time, the distribution of GFIs in each project, and the relationship between its application and project popularity. We find that GFIs are reported by more and more projects, especially the popular projects. However, GFIs account for a small proportion among the projects' issues.

RQ2: How are GFIs solved? We answer this question from two aspects: the resolution process of GFIs and the effect for helping newcomers onboard. We utilize six metrics to compare the resolution process of GFIs with that of other issues, and manually analyze 200 GFIs to understand the participation of newcomers. We find that GFIs usually need more days to be solved. Although some newcomers contribute successfully through GFIs, there are still almost half of the GFIs not solved by newcomers and those newcomers who successfully solved GFIs are hard to retain.

RQ3: What factors and problems are related to the effectiveness of GFIs? We establish four sets of metrics and fit a logistic regression model to evaluate their correlation with whether a GFI is solved by a newcomer. We find that several factors, e.g., project popularity, significantly correlate to the effectiveness of GFIs. To complement the above results, we conduct email surveys with the newcomers and the project members to understand their problems and challenges. A thematic analysis on the responses reveals eight reasons for failed contributions.

RQ4: How to identify appropriate GFIs? We conduct email surveys with project members and summarize their practices on labelling GFIs. We also compare the descriptions between the GFIs solved by newcomers and the GFIs not solved by newcomers. Eventually, we extract the key information in GFIs (one of criteria to identify GFIs) that may help newcomers more likely onboard.

This paper makes the following contributions:

- We make a comprehensive understanding of the mechanism of GFIs from its application status, effect, problems, and good practices.
- We obtain various reasons for unsuccessful first contributions, which can help us better understand the barriers facing by newcomers and the difficulties in this guiding process.
- We discover the criteria to identify GFIs that may make GFIs more likely to be solved by newcomers.

We organize the remainder of the paper as follows. Section 2 presents related work; Section 3 introduces our dataset. Section 4 to Section 7 present methods and answers to each of the four research questions. Section 8 discusses the implications of findings for practitioners and researchers. Section 9 presents threats to validity and Section 10 concludes the paper.

2 RELATED WORK

We discuss the related work from two aspects: 1) motivations and barriers for contributing to OSS projects; 2) theories and strategies helping newcomers onboard. Developers participate in OSS projects for a variety of motivations, which has been thoroughly studied. Prior work has examined the motivations of core developers [2, 8, 29, 40] and of peripheral developers [18, 19]. A rather comprehensive survey conducted by Von Krogh et al. reviewed the research on developer motivations over ten years [37]. It summarized that developers are generally motivated by intrinsic (i.e., ideology, altruism, kinship, and fun), internalized extrinsic (i.e., reputation, reciprocity, learning, and own-use), and extrinsic motivations (i.e., career and pay). On the one hand, many newcomers want to join OSS projects, and on the other hand, a continuous influx of newcomers is essential for the survival, long-term success, and continuity of OSS projects [1, 8]. However, newcomers face many challenges in their initiative activities in OSS projects due to lack of the necessary domain knowledge and programming skills [19, 30]. Besides, some non-technical factors also affect newcomers onboarding, e.g., communication [35]. Researchers found that there are 58 barriers, grouped into six different categories: cultural differences, newcomers' characteristics, reception issues, orientation, technical hurdles, and documentation problems [32]. A recent research conducted by Mendez [23] took a new perspective of barriers from tools and infrastructure and found that most of these barriers are tied to newcomers' barriers that have been established in the literature.

Theories and strategies have been put forward to help newcomers onboard. Zhou and Mockus [43] found that the probability for newcomers to become long-term contributors is associated with the capability they present through number of tasks, the effort they devote into issue reports, and the amount of attention they receive from the project. By investigating newcomers' first interactions on mailing lists, researchers found that receiving timely responses, especially within 48 hours, was positively correlated with future participation [13]. Besides, many studies aim to propose supportive strategies for newcomers. Based on the interview and analysis of the existing literature, Steinmacher et al. [34] propose guidelines helping newcomers onboard including contribution process guidelines, social behavior guidelines, and technical guidelines. In order to provide personalized technical support, some studies focus on

[2] https://help.github.com/en/github/building-a-strong-community/encouraging-helpful-contributions-to-your-project-with-labels
[3] https://github.com/scikit-learn/scikit-learn/issues/11554
[4] In this paper, we refer all the issues intended for newcomers as GFIs.

recommending appropriate mentors for newcomers [26]. Task recommendation has also been found important for newcomers. By conducting interviews with the newcomers and the experienced members, researchers found that finding a task to start is difficult for newcomers. To help newcomers find their tasks, they provided a set of strategies, e.g., listing easy tasks by difficulty [33]. Although this study noticed the importance of task recommendation for newcomers, it did not take into account the fact that many projects currently have been labelling suitable tasks (GFIs) for newcomers. A recent study [4] investigated the relationship of newcomers' supportive strategies in GitHub with the level of project success and found a strong positive relationship with task labelling for newcomers. However, many problems still arise in practice concerning the effect of GFIs, e.g., how exactly GFIs are identified and what problems exist. Different from earlier studies, we make a comprehensive understanding of the mechanism of GFIs, which is necessary for understanding newcomers' barriers in finding tasks and the difficulties in this guiding process, and further help newcomers to join the OSS projects.

3 DATASET

We introduce how we construct our dataset, including the standard we follow to select projects and the approach to identify GFIs.

3.1 Project Selection

We conduct this study based on GHTorrent database [10]. To ensure query efficiency, GHTorrent can be accessed over Google Cloud services, which provides an up to date import of the latest GHTorrent MySQL dump.[5] Its lasted version is "2018-04-01" at the time of data collection. We use this dataset as a source to select the most appropriate projects for this study. According to GHTorrent, GitHub hosted 83,624,114 OSS repositories (including forks) in April, 2018. However, many are inactive, toy projects, or duplicates. In our study, we are interested in state-of-the-art software systems that have a relative large real-world user base, meaning that they may have chance to attract newcomers' attention. To retrieve a sensible sample of projects from GitHub, we follow the established project selection instructions [15, 24]. We focus on six programming languages with the largest number of GitHub repositories: JavaScript, Python, Ruby, C/C++, Java, and PHP. We select the top-500 most starred repositories (excluding forks to avoid including the same project multiple times) for each of those languages at the moment of analysis.

To safeguard the quality of the dataset, we excluded the following repositories from the resulting collection of 3,000 repositories: (a) the projects[6] that do not report any issues or the number of reported issues is small (< 50), (b) the projects that do not have the files of "CONTRIBUTING.md" (a default community health file[7]), which may indicate they are not willing to attract newcomers, and (c) the projects that are not software units or are explicitly labeled as unmaintained or are not in English. To apply the last filter, we manually inspected the project descriptions and excluded 240 projects (containing education, storage, and technology code sample, etc.). The final dataset is composed of 816 (= 3000 - 443 - 1501 - 240) projects.

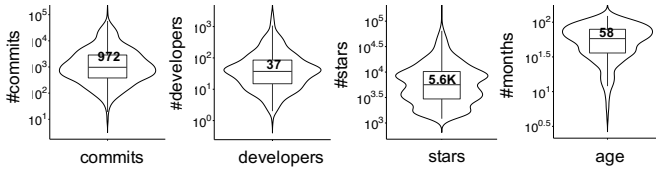

Figure 1: Distribution of the number of commits, developers, stars, and ages of the projects

Figure 1 shows violin plots with the distribution of the number of commits, developers, stars, and age per project in logarithmic scale (note the logarithmic scale of Y axis). The median values are indicated inside the violin plots. It seems that the constructed dataset typically includes large projects, both in number of commits and developers, and that the projects also more likely attract newcomers' attention (number of stars) and have a relatively long history.

3.2 Identification of Good First Issues

GitHub gives projects' members right to organize and prioritize their work.[8] They can apply labels to issues to signify priority, category, or any other useful information. Although *good first issue* is the default label that GitHub encourages projects to tag for issues suitable for newcomers, many projects prefer to use other labels, e.g., *"beginner", "easy", "first timer only"*, and some projects even use multiple labels of this kind.[9] Therefore, we consider two types of labels as our study objects: labels indicating suitability for newcomers (e.g., newbie); labels indicating simplicity of tasks (e.g., easy, minor bug/feature, typo). We include the second type because easy tasks are often considered suitable for newcomers [33].

In order to identify all the issues intended for newcomers, the first two authors independently read and analyze the labels of all the issues in the projects we collect. There are 1,471,541 issues with 7,924 different labels. Because in most cases, the labels indicating issues intended for newcomers are obvious, two authors have the same judgement on these labels. For few labels that we are not certain, such as *"help wanted"*, we decide to delete them to ensure the quality of data (inter-coder reliability: 1-8/7,924=99.90%). We find 113 labels that are used to indicate issues suitable for newcomers. Eventually, we obtain 9,368 GFIs including 9,110 closed GFIs.

Figure 2 presents the ratios of top 20 commonly used labels for GFIs. Although these labels are various, many of them are similar by nature. The label of GFI takes the largest proportion (14.2%) and we use it to represent all the labels intended for newcomers.

4 RQ1: HOW COMMON DO PROJECTS REPORT GFIS?

We aim to understand the application of the mechanism of GFIs in the GitHub. All the analyses are conducted quantitatively.

[5] http://ghtorrent.org/gcloud.html
[6] we use repository and project interchangeably in the paper.
[7] https://help.github.com/en/github/building-a-strong-community/creating-a-default-community-health-file
[8] https://help.github.com/en/github/managing-your-work-on-github/labeling-issues-and-pull-requests
[9] https://github.com/MunGell/awesome-for-beginners

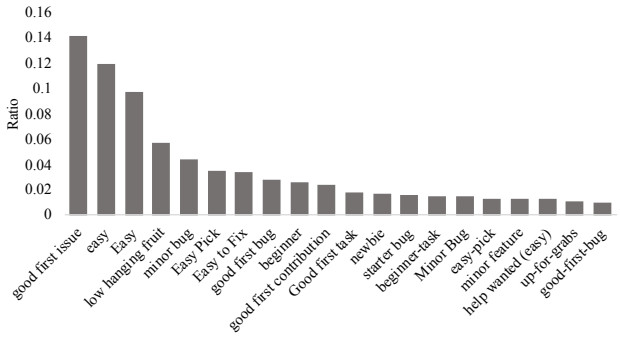

Figure 2: Top 20 commonly used issue labels for newcomers

4.1 Methodology

We answer this RQ from three aspects: 1) the trend of the application of this mechanism; 2) the distribution of GFIs in a project; 3) the relationship between the application of this mechanism and the project popularity. The analysis is based on the projects and GFIs described in Section 3. To decide whether a project applied this mechanism in a certain time period, we consider whether it reported GFIs in that time period.

4.2 Results

Trend of application. Figure 3 shows the number of projects that applied GFI labels from 2009 to 2017.[10] We can see that more and more (state-of-the-art) projects applied this mechanism over time. Especially in recent years, it shows a significant increasing trend. In 2017, nearly 20 percent of the projects that reported issues reported GFIs. It suggests that labelling GFIs has become an increasingly popular way to help newcomers to participate into projects.

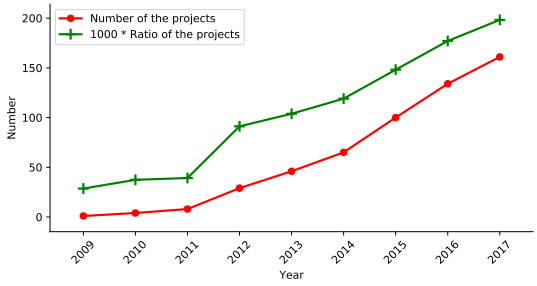

Figure 3: Trend of the application of the mechanism of GFIs

Distribution of GFIs. Figure 4 shows violin plots with the distribution of the number of GFIs and the ratio of GFIs per project (note the logarithmic scale of Y axis). We can see that the number of GFIs varies largely in different projects—the project *scikit-learn* reported 789 GFIs (41.1% of all the issues), whereas the project *savonrb* reported only one (0.2%) GFI. However, GFIs are few overall: the median number of GFIs reported by a project is 11 and the median ratio is only 0.04.

Relationship between project popularity and application of this mechanism. As shown in Figure 5, 52.5% of the top 40 popular projects (sorted by #stars) had ever reported GFIs, whereas this number is 22.5% for the bottom 40 projects. On the whole, the

[10]We do not draw the data of 2018 because we only have four months data for that year.

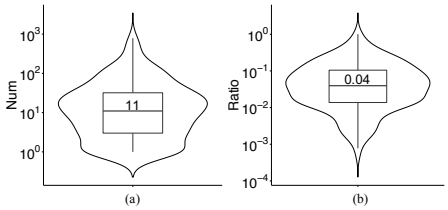

Figure 4: Distribution of GFIs per projects

ratios show a downward trend. It seems that a project's popularity is correlated with whether it labels issues for newcomers (note: it does not suggest causality). To validate this assumption, we use a Spearman correlation to evaluate whether a project has ever reported GFIs is related to the number of stars it has. However, the results (p-value = 4.51e-08, rho = 0.19) suggest a quite weak positive relationship, which shows that the factors related to a project's popularity are complex.

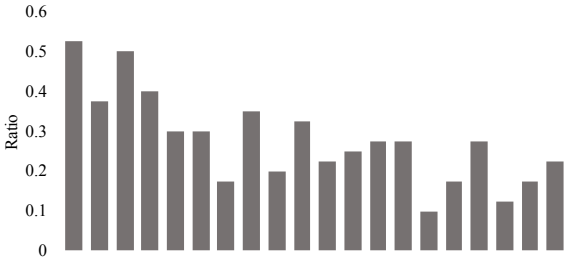

Figure 5: Ratio of the projects reporting GFIs. Each bar represents 40 projects. All the projects are sorted in descending order of popularity (#stars).

> *Summary:* In GitHub, more and more projects (state-of-the-art software systems) are helping newcomers contribute by labeling GFIs, especially for extremely popular projects. However, for most projects, this type of issues account for a small proportion among the projects' issues.

5 RQ2: HOW ARE GFIS SOLVED?

We intend to explore the resolution process and effect of GFIs. Effect means to what extent GFIs can help newcomers onboard.

5.1 Methodology

Targeting the above two goals, our method includes two parts: analysis of the resolution process and the effect of this mechanism.

5.1.1 Analysis of the Resolution Process. We compare the resolution process of the GFIs with that of other issues. Considering there are 9,110 closed GFIs in our dataset, we randomly select the same number of other issues to make a comparison (confidence level: 95%, margin of error: 1.01% [3]). The labels used by other issues are different from GFIs. For example, the top three most used labels are: "*bug*", "*question*", and "*enhancement*". Taking into account the existing studies related to issue resolution process [16] and the availability of GitHub API, we calculate the following metrics for these two groups of issues. **#Days_Resolution**: number of days of resolution, which starts from the day the issue is reported until

it is resolved; **#Actors_Subscription**: number of actors who subscribe to receive notifications for an issue; **#@mentioned**: number of times that developers are @mentioned in an issue body; **#Times_Reference**: number of times that an issue is referenced in a commit body; **#Commenter**: number of developers who comment on an issue; **#Comments**: number of comments on an issue. To determine whether GFIs and other issues are different on these metrics, we utilize the Mann-Whitney U test to compare the two independent groups [25].

5.1.2 Analysis of the Effect. We demonstrate the effect of GFIs from three aspects: (1) among GFIs, how many were solved by newcomers; (2) among newcomers who attempted to solve GFIs, how many of them successfully contributed; (3) among newcomers who successfully solved GFIs, how many of them may retain. The first two aspects focus on the effect of attracting newcomers, while the last aspect focuses on contributor retention. Retention helps to accumulate expertise and developer fluency, thus reducing development costs due to improved productivity [42].

To this end, we randomly choose 200 GFIs from 9,110 closed GFIs for manual analysis. For each of them, we analyze their comments to identify who successfully solved it and who attempted to solve it but eventually failed. We determine whether a developer was willing to solve the issue by the comments such as, "*I'll do this one...*", "*I'd be glad to take a stab at putting a PR in for this*", and etc. To determine who successfully solved the issues, we check the reasons why the issues were closed, e.g.,"*jnothman closed this in #15138 on 30 Oct 2019*".[11] Among 200 GFIs, 36 GFIs are not suitable for our study, such as the issues were closed because the requirements were changed, the issues were duplicate, or they had already been fixed before being reported. Therefore, our analysis is based on 164 GFIs (confidence level: 95%, margin of error: 7.58%). These GFIs involve a total of 141 newcomers attempting to contribute.

Previous research usually treats a newcomer as a developer trying to place their first code contributions into the project [1]. However, when manually analyzing GFIs, we observe that in many cases, developers who had already successfully made a contribution still asked for solving a GFI because they thought they are not capable enough to solve other issues. Therefore, in this study, we treat a developer who have previously contributed less than three commits to a project as the newcomer to this project.[12]

5.2 Results

5.2.1 Resolution Process of GFIs. Figure 6 shows the differences between GFIs and other issues in their resolution process, calculated by the metrics defined in Section 5.1. We can see that almost all the box plots have long upper whiskers and short lower whiskers. It suggests that there are larger variances among the greater values, but the smaller values are concentrated. That is to say, except for some issues with extremely long resolution process (#Days_Resolution) and extensive participation (other metrics), many issues can be solved relatively quickly with limited participation. We can also see that GFIs and other issues distribute differently on this set of metrics: the box plots for GFIs seem to be higher than the equivalent plots for other issues.

[11] https://github.com/scikit-learn/scikit-learn/issues/15076

[12] We use "less than three commits" because the distribution of #previous_commits of developers who solved GFIs shows "sparse in the middle while dense at both ends" and two (commits) is a perfect dividing point.

Table 1: Statistical difference of resolution process for GFIs and other issues

	Median		Mean		r
	GFIs	Others	GFIs	Others	
#Days_Resolution	33	3	143.97	75.55	0.32*
#Actors_Subscription	1	0	1.40	1.13	0.06*
#@mentioned	1	0	1.85	1.14	0.16*
#Times_Reference	0	0	1.49	0.48	0.24*
#Commenter	2	2	2.88	2.43	0.12*
#Comments	4	3	5.60	4.63	0.11*

r represents the effect size ($r = z/\sqrt{N}$). According to [9], $abs(r) \geq 0.1$, $abs(r) \geq 0.3$, and $abs(r) \geq 0.5$ represent small, medium, and large effect sizes, respectively.
* means p-value < 0.001.

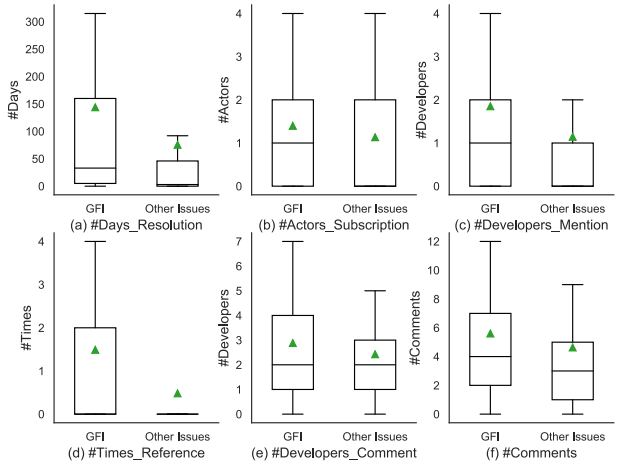

Figure 6: Comparison of resolution process between GFIs and other issues. The outliers are removed.

Table 1 shows the statistical difference of GFIs and other issues on the defined metrics. Consistent with the above analysis, both median and mean values of GFIs are greater than or equal to those of other issues. The results of the Mann-Whitney U test show that although the differences of these two groups on all the metrics are statistically significant, most metrics (except #Day_Resolution) show small effect size, i.e., the differences are not necessarily meaningful in practice.

> *Summary:* For the resolution process, GFIs often take longer in terms of resolution time. However, for other aspects (e.g., the numbers of actors who subscribe, the number of comments and commenters), there is no obvious actual difference between GFIs and other issues.

5.2.2 Effect of GFIs. Figure 7 shows the resolution of GFIs. Less than half (45.1%) of the GFIs were successfully solved by the newcomers who had not contributed any commits yet. A small number

(14.0%) of GFIs were solved by newcomers with little experiences (less than three commits). However, 40.9% of GFIs were not solved by newcomers, among which 29.3% did not attract newcomers' attention at all.

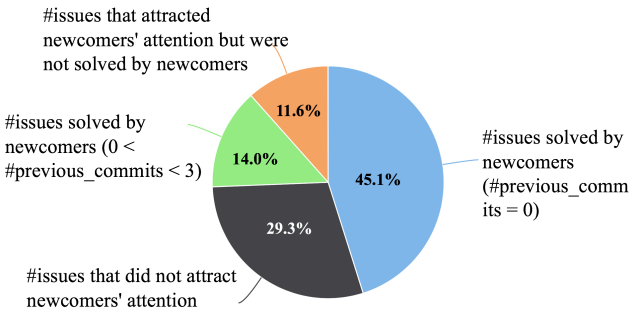

Figure 7: Resolution of GFIs

While the above analysis takes the perspective of GFIs, Figure 8 shows the effect of this mechanism from the newcomers' perspective. Among the 141 newcomers, 68.8% contributed successfully, which implies that some newcomers did contribute successfully through GFIs. However, 31.2% of newcomers did not succeed in contribution and these GFIs were eventually solved by other newcomers or experienced developers.[13] When analyzing the comments, we noticed that while some newcomers were trying to solve the GFIs, some other newcomers suddenly submitted the correct patches (e.g., material-ui: #10609[14]), which leads to the futility of some newcomers' efforts, thus reducing their enthusiasm for contribution. Also, there are cases where newcomers have tried for a long time but still did not submit the right patches. Eventually, the issues were solved by the experts, for example, eslint: #3218[15]. It implies that these GFIs may be difficult and thus are not suitable for newcomers.

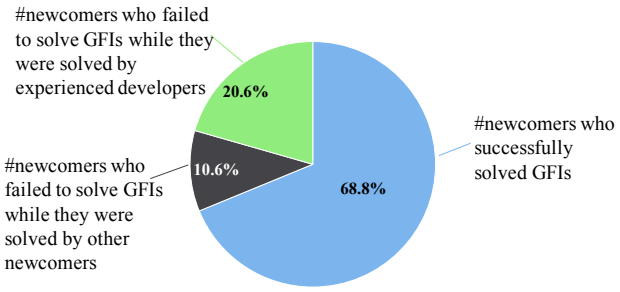

Figure 8: Count of Newcomers who tried to solve GFIs

Figure 9 shows the retention of newcomers who successfully contributed through this mechanism, which is represented by the number of commits they contributed later. Among the 97 newcomers, most (58.4%) are one time contributors meaning that they left the projects after their first contribution. Only three contributors contributed more than ten commits. It suggests that this mechanism may be hard to attract long-term contributors.

[13] In this paper, experienced developers refer to non-newcomers.
[14] https://github.com/mui-org/material-ui/issues/10609
[15] https://github.com/eslint/eslint/issues/3218

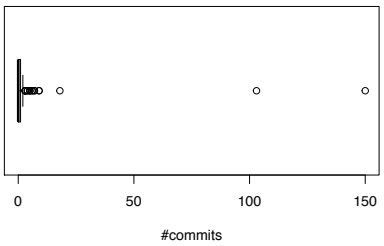

Figure 9: Retention of newcomers

Summary: Some newcomers did join the projects through GFIs but there are still 40.9% of GFIs that were not solved by newcomers and 31.2% of newcomers who failed to solve the GFIs after several attempts. A serious problem seems to be that GFIs are hard to attract long-term contributors, which makes GFIs less effective from the projects' perspectives.

6 RQ3: WHAT FACTORS AND PROBLEMS ARE RELATED TO THE EFFECTIVENESS OF GFIS?

We aim to identify the factors related to whether GFIs is solved by newcomers and the problems of this mechanism.

6.1 Methodology

We first quantitatively explore the factors that may affect newcomers to solve GFIs. Then, we conduct email surveys with newcomers and project members to provide more comprehensive findings.

6.1.1 Quantitative Analysis. We construct a logistic regression model to examine the relationship between the resolution of GFIs and four sets of metrics. These metrics are proposed based on the hypothesises taking into account the factors that may affect newcomers to solve issues and the accessibility of data. They are also generally used in the previous studies. The dependent variable Y of the model is whether a GFI is solved by a newcomer. We code the dependent variable as Y = 1 if a GFI is solved by a newcomer and Y = 0 otherwise. The independent variables use metrics from the following four factors.

(1) **Textual Factors** are extracted directly from the textual data (i.e., the description and title of a GFI), borrowed from a study related to issue reports [12].

- **Length of the Title** is defined as the number of words contained in the title of a GFI. A longer title is more likely to provide sufficient information, and thus newcomers may easily solve it.
- **Length of the Description** is the number of words in the description. Similar to the title, a longer description is likely to provide more elaborate information about the issue.
– **Number of the URLs** is the number of URLs in the description. URLs usually provide valuable information of GFI, e.g., ways to reproduce the bug. More URLs may help newcomers solve the GFI.
- **Number of the Code Snippets** is the number of code snippets in the description. Similar to URLs, code snippets usually provide valuable information (e.g., the location of bugs) about the GFI.
- **Readability** is the Coleman-Liau index [21] of the GFI description, which has been applied in the evaluation of issue reports [11]. It

reveals how difficult to understand the text. A higher readability is more likely to make the GFI easier to understand.

(2) **Technical Factors** aim to evaluate the difficulty and risk of solving a GFI. Instead of directly measuring GFI, they measure the commits corresponding to a GFI. The following metrics are borrowed from a study evaluating software changes [31].

– **Number of the Lines Added** calculates the number of lines of code added when solving a GFI. Adding more lines may mean that the GFI is more difficult and requires more effort to solve.

– **Number of the Lines Deleted** calculates the number of lines of code deleted when solving a GFI. Deleting more lines may introduce more risks, so it is less possible for newcomers to solve.

– **Number of the Files Modified** calculates the number of code files modified when solving a GFI. Modifying more files may mean that the GFI is more risky, so it is not easy for newcomers.

(3) **Project Popularity** is measured by the number of stars of a project that the GFI exists in. More popular the project is, more likely the GFI can attract newcomers' attention [36].

(4) **Reputation of the Labeler** calculates the contributing history of project members who labeled GFIs. Experienced project members are more likely to accurately identify GFIs. We consider the experience of a developer in the certain project and design two metrics: **Number of Days Previously Contributed** and **Number of Commits Previously Contributed** in a project.

To construct this model, for a GFI, we need to know who successfully solved it, but there is no such field in GitHub and because there are 9,110 closed GFIs, manual inspection seems impossible. Fortunately, GitHub supports developers to use keywords (e.g., close, fix, resolve) in pull request descriptions, as well as commit messages, to automatically close issues.[16] Issues will be automatically closed only when commits are merged into the main branch, which can be obtained through GitHub API. Therefore, we identify who successfully solved GFIs and the corresponding commits by analyzing these connections. Eventually, we obtain 1,429 GFIs using this mechanism to close issues, including 684 cases solved by newcomers and 745 cases solved by experienced developers (confidence level: 95%, margin of error: 2.38%). Based on this data, we firstly conduct metric selection because the existing highly correlated factors can lead to overfitting of a model [7]. Then, we calculate the Spearman rank correlation for our metrics to test whether there is any highly correlated pair of metrics. We find that only one high correlation (correlation coefficient $\rho >= 0.7$ [22]) exists: the pair of **Number of Days Previously Contributed** and **Number of Commits Previously Contributed**. We decide to keep the latter.

6.1.2 Email Surveys. In Section 5.2.2, we have identified 44 newcomers who failed to solve GFIs and 39 project members who were associated with 67 ineffective GFIs—meaning that the GFIs labeled by them were failed to be solved by newcomers. We conduct email surveys with them to obtain their views and experiences. Specifically, we ask newcomers *why they failed to solve this GFI*. For project members, we ask *what reasons they thought caused the GFI failed to be solved by newcomers*. We also ask the newcomers and project members their views for *the common challenges and problems of this mechanism*. Eventually, we obtain nine responses from newcomers (response rate: 20.5%) and 11 responses from project members (response rate: 28.2%). We analyze these answers using thematic analysis [6], a common method for analyzing qualitative data. It involves the following steps: (1) initial reading of the answers, (2) generating the initial codes for each answer, (3) searching for themes among the proposed codes, (4) reviewing the themes to find opportunities for merging, and (5) defining the final themes aiming to identifying the "essence" of what each theme is about. Steps (1) to (4) are performed independently by the first two authors. After this, a sequence of meetings is held to resolve conflicts and to assign the final themes (step 5). The final inter-rater reliability is 94.29%.

[16] https://help.github.com/en/enterprise/2.16/user/github/managing-your-work-on-github/closing-issues-using-keywords

Table 2: Results of the logistic regression model. Only significant predictors are shown

Metrics	Est.	Std. Err.	z	Pr(>|z|)
(Intercept)	-6.61	0.85	-7.75	9.1e-15
length of the description	0.17	0.06	2.85	0.00436
#lines deleted	-0.16	0.06	-2.82	0.00487
#commits previously contributed	-0.11	0.03	-3.63	0.00031
#stars of the project	0.58	0.07	8.02	1.1e-15

All the metrics were log-transformed. H-L: $X^2 = 10.70$, $df = 8$, $p = 0.22$

6.2 Results

6.2.1 Results for Logistic Regression Model. Table 2 shows the results of the model. The inferential goodness-of-fit test is the Hosmer–Lemeshow (H–L) test that yield a X^2 of 10.70 and is insignificant ($p > 0.05$), suggesting that the model is fit to the data well [27]. We find that four predictors are significant (at < 0.005 level). After removing those non-significant predictors from the model, we obtain more significant results (at < 0.001 level). We observe that the most significant factor is the **Popularity of the Project**. It is reasonable because if a project can not attract the attention of newcomers, let alone expecting them successfully onboard through GFIs. The results show that the **Reputation of the Labeler** is also an important factor. Different from what we assume, we find that if a GFI is labeled by a project member with less experiences (#commits s/he made), it may have more chance to be solved by a newcomer. A project member in the survey (Section 7.1.1) expressed the similar views, he said, "*I usually treat the issues which I know how to do myself without any substantial investigation as GFIs, but as I have become more familiar with the code, this criterion is less useful*". This may reveal a mismatch between experts and novices, a well understood "zone of proximal development" [39], which describes the case where experts are not usually effective at training or teaching novices. We find that the **Number of the Lines Deleted** shows a negative relationship, which suggests that newcomers are unlikely to solve a GFI involving bulk deletions. After all, this is a high-risk operation. It is worth to note that **Number of the Lines Added** is not correlated, because the GFIs solved by newcomers vary in this regard, including document modifications, enhancement (e.g., adding comments and renaming functions/classes, which generally involve a large number of additions), and minor fixes. In addition, we observe that **Length of a GFI's Description**, i.e., level of detail, is positively related to whether it is resolved by newcomers.

Table 3: Problems of the Mechanism of GFIs/ Why did newcomers fail to solve GFIs? (■: Newcomer ■: Project member)

Problems	Categories	Responses
Insufficient GFIs in the project	Project	11
Inappropriate GFIs	Mechanism	10
Lacking of motivations to retain	Newcomer	7
Hard to start on a new project	Newcomer	6
Few newcomers in the project	Project	5
Uneven level of newcomers' skill	Newcomer	5
Getting snapped up soon	Mechanism	3
Lacking of mentors	Mechanism	2

6.2.2 Results for Email Surveys. We obtain more comprehensive results by email surveys, as shown in Table 3. The thematic analysis on the email responses reveals eight problems classified into three categories. The results of the earlier regression model only reflect two: *inappropriate GFIs*—reflected by length of the description, #lines deleted, and #commits previously contributed (reputation of the labeler), and *few newcomers in the project*—reflected by #stars of the project. The category **Mechanism** contains the problems of the mechanism itself. Eight newcomers and two project members report that certain GFIs are *inappropriate for newcomers*, e.g., it requires a lot of time to figure out where the issue exists. Some developers report that *GFIs getting snapped up soon* and *lacking mentors when getting stuck* are the main reasons for unsuccessful contributions. For example, a newcomer complains that "*when I were trying to solve this issue, another developer submitted the right solution soon*". It indicates that projects need to establish an allocation mechanism to limit who can pick GFIs. The category **Newcomer** contains the problems mainly related to newcomers themselves, such as *lack of motivations to contribute back* and *hard to start on a new project*. For example, a project member says "*they just looking for a way to put a few PRs on their resume, and often aren't willing to make more involved contributions*." It can explain why the solvers of GFIs are almost one time contributors, as illustrated in Figure 9. The last category is **Project** that contains two problems. One is *insufficient GFIs*, which is consistent with the findings in Figure 4. The other is *few newcomers*. Consistent with results of the logistic regression model, it is mainly due to the low visibility of the projects.

> *Summary:* The factors and problems affecting the effectiveness of GFIs are complicated and various. We find that several factors are correlated with the resolution of GFIs, including project popularity, reputation of the labeler, etc. The problems revealed by newcomers and project members cover the categories of project (e.g., insufficient GFIs), mechanism (e.g., inappropriate GFIs) and newcomer (e.g., uneven skill).

7 RQ4: HOW TO IDENTIFY APPROPRIATE GFIS?

We conduct email surveys with project members to discover the criteria for identifying GFIs. We further complement the key information needed in GFI descriptions by comparing the descriptions of solved GFIs and unsolved GFIs. For brevity, in this RQ, "solved GFIs" represents the GFIs that were solved by newcomers; "unsolved GFIs" represents the GFIs that were solved by experienced developers.

7.1 Methodology

7.1.1 Extraction of the Criteria for Identifying GFIs. We extract the common characteristics of GFIs by summarizing the practices of project members in identifying GFIs, which can provide insights for understanding beginner-friendly tasks and further optimizing this mechanism. We conduct email surveys asking project members who ever labelled GFIs how they identify GFIs. Specifically, we choose three GFIs that they most recently labeled and ask: *(1) For each of the above issues, why did you think it was suitable for newcomers?* To obtain more general criteria, we asked the second question: *(2) In general, what are your standards when judging whether an issue is suitable for newcomers?* We also provide some tips based on the former analysis, e.g., the required technology, type of tasks, and descriptions. There are 191 developers who recently (after January, 2018) had identified GFIs. Among them, 153 developers disclosed their email information on GitHub. We send them questionnaire. After a period of 20 days, we obtain 24 responses and seven mails undelivered, resulting in a response rate of 16.4%. To preserve the respondents' anonymity, we use labels D1 to D24 to identify them. We analyze these answers using thematic analysis that is similar to the process in Section 6.1.2. The first two authors conduct this process and the final inter-rater reliability is 95.18%.

7.1.2 Extraction of Key Information in GFI Descriptions. Description is the most important indicator of an issue, as confirmed by the analysis described in Section 7.1.1. Therefore, we extract the key information needed by GFI through comparing the descriptions of solved GFIs and that of unsolved GFIs. The analysis is based on the 1,429 GFIs referred in Section 6.1.1.

The descriptions of GFIs may contain many redundant and trivial words, as well as non-text information (e.g., screenshots and code snippets). Therefore, we first perform a format analysis on the descriptions, leaving only the plain texts. Then, we employ a term frequency-inverse document frequency (TF-IDF) algorithm [28] to mine the keywords. TF-IDF determines the relative frequency of words in a given GFI description compared to the inverse proportion of that word over a set of GFI descriptions. Intuitively, this algorithm determines how relevant a given word is in a given GFI. Formally, for a set of GFI descriptions S and a given word w in a GFI description d ($d \in S$), the word relevance i_w is computed by

$$i_r = f_{w,d} \log(|S|/f_{w,S}), \quad (1)$$

where $f_{w,d}$ denotes the number of times the word w appears in d, $|S|$ denotes the size of the set S, and $f_{w,S}$ denotes the number of times the word w appears in S. Based on this algorithm, we obtain a set of relevant words of each GFI's description. In particular, to avoid recognizing different inflected forms (e.g., "return" and "returns") as different words, we use lemmatisation to refine each GFI via the NLTK tools [20]. We select the top 20 relevant words as keywords of GFI. To find the key information that may make GFIs easier to be solved by newcomers, we collect and compare the keywords of the solved GFIs and unsolved GFIs.

7.2 Results

7.2.1 Criteria for Identifying GFIs. To understand how project members identify GFIs, we conduct email surveys and obtain 24 responses. We present the criteria for judging GFIs with examples of answers associated to them. Because the answers to the two questions in the surveys are similar, we do not make a distinction.

Clear Issue Description: Sixteen developers report a GFI should have a clear description telling where errors occur or what changes need to be done, as in the following answers:

We wrote up exactly what the problem was and pinpointed the exact code that needed to be changed. (D1)

The issue to be addressed was described clearly in the issue. (D10)

Clearly written issue which identifies specifically what needs to change in the behaviour of the software or application. (D19)

Self-Contained Change: Almost all the developers (20 out of 24 responses) consider the scope of changes is an important factor to evaluate whether issues are suitable for newcomers, for example:

The change is self-contained: (1) it touches a small part of the codebase, ideally a single function; (2) it doesn't change wide-spread APIs that require chasing callers; (3) ideally, it's unit-testable. (D4)

Its a short, self contained task that should only touch very few files and the approach to take was already outlined in the issue. (D20)

Limited Skills Needed. Fifteen developers report that an ideal GFI only requires limited skills for solving, because newcomers generally do not have rich experiences, for example:

Issues which won't require any substantial knowledge of the specific programming language or the overall application. (D5)

It doesn't require inside out familiarity with the whole codebase, doesn't imply deep knowledge of RFCs or technical standards, and doesn't require guru-level coding skills. (D4)

"Good first issues" should not tackle any large architectural or behavioural changes. Documentation changes, test fixes, small non-urgent bug-fixes and internal refactoring are good candidates. (D13)

Less Workload: Fourteen developers report that an ideal GFI should take less time for newcomers to solve. As examples,

As for what we use to judge if it's suitable for newcomers, usually it's the amount of time it would take to implement it (usually < 3h). (D5)

Most times I consider issue to be beginner friendly when it only requires few lines of changes which should be more or less trivial to do. (D6)

Available Support: Eight developers consider whether maintainers can provide timely and effective support is essential, e.g.,

I can support newcomers if they get stuck and lead the decisions. (D4)

Is someone on the maintainer team willing and available to support someone new to the project to get the context they'd need to solve this by themselves? (D18)

Motivating Newcomers: Three developers express this view, which brings us a new perspective for identifying GFIs, for example:

It has a visible final impact on users. This is a meta-property of the issue, and I try to take that into consideration to avoid assigning boring tasks. Also, newcomers want to "see" the end result. (D8)

Issues with medium-high value: it can provide a chance to learn about and practice with a new feature. (D12)

Low Urgency. There are three developers mention that GFIs should be low urgency. As an example:

Table 4: Comparison of keywords in GFIs solved by newcomers and GFIs solved by others. (■: Solved ■: Unsolved)

Keywords	Comparison	Keywords	Comparison
version	107	map	37
documentation / doc	89	platform	36
return	49	subsystem	34

We usually treat issues that are low urgency as good first issues because newcomers need more time to get familiar with projects. (D12)

There are other interesting findings that seem inconsistent with the above opinions and are not the mainstream view. For example: a project member (D9) considers GFIs is a way to attract outside experts. He says, "*a good first issue isn't necessarily an easy issue. In many cases these are actually quite difficult. Sometimes the 'good first issue' label is more like an invitation for experts in other subject areas who have knowledge about a related external piece of software to contribute*". Project member D11 also expresses the similar opinion: "*the newcomers that don't find them easy are not qualified: these 'good first issues' are effectively a test to see who should be encouraged further*".

Additionally, three of the respondents express their doubts. It implies that correct identification of GFIs is not an easy task. For example, D3 says, "*unfortunately, these boundaries are also murky for me and I abstain from applying these labels myself*". Similarly, D7 says, "*I don't always agree that issues other people have marked as 'good first issues' are actually good for newcomers*".

> *Summary:* The following criteria are generally used by project members to identify GFIs: whether it has clear descriptions, whether the changes are self-contained, how much effort and skill it requires, whether it motivates newcomers, whether support is available, and the urgency.

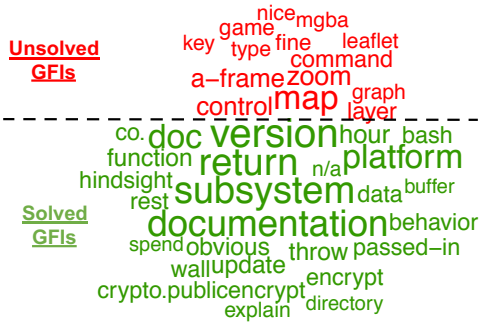

Figure 10: Comparison of solved GFIs and unsolved GFIs. The top 50 keywords with the highest frequency are shown.

7.2.2 Key Information in GFI Descriptions. We extract the most relevant keywords in the GFIs' descriptions that we believe may distinguish suitable GFIs from inappropriate ones. The results are shown in Figure 10 with two clouds (R function: *comparison.cloud*), where we use the keywords of 1,429 GFIs in total. We select the top 50 frequent keywords in total obtained by TF-IDF and make a

comparison of the GFIs that are solved by newcomers (green) and the GFIs that are not solved by newcomers (red). As shown, the keywords of GFIs solved by newcomers are richer than those of GFIs not solved by newcomers. However, the real status is just the opposite: the descriptions of solved GFIs are relatively similar (high keyword frequency), while the descriptions of unsolved GFIs are more diverse (low keyword frequency). Furthermore, the meaning of these two sets of keywords are quite different.

We notice that there are six most frequent keywords. Their frequencies are shown in Table 4.[17] It appears the GFIs that contain keywords "*version*" (66.4%), "*return*" (81.6%), "*subsystem*" (94.1%), "*documentation* / *doc*" (80.9%), and "*platform*" (86.1%) in the descriptions are more likely to be solved by newcomers. For more details, we check the occurrence of these words. The words "*version*", "*subsystem*", and "*platform*" mostly appear when describing the environment in which issues occur (e.g., "version: master, platform: UNIX, subsystem: test"). The word "*return*" mostly appears when describing the return it should be (e.g., it should return a not implemented error.) or the actual return (e.g., statusbar.current height returns undefined instead of real height). The word "*documentation* / *doc*" usually appears when the document is not detailed enough and needs to be edited or added (e.g., "I think the documentation needs to be updated to clarify this behavior"). In fact, such information clarifies the steps for reproducing issues and the form of expected output, which is critical for newcomers. These findings detail the results of earlier survey that show "clear description" is a criterion for GFIs but it is unclear how to quantify "clear" in the descriptions.

> *Summary:* The following information in descriptions may make GFIs more likely to be solved by newcomers: 1) the detailed environment of the version, the subsystem, and the platform where the issue occurs; 2) the actual return of the issue; 3) the expected return of the issue. The issues about documentation are more likely to be solved by newcomers.

8 IMPLICATIONS

We offer practical implications of our findings for newcomers, project members, and researchers.

8.1 Newcomers

The results for RQ2 (Figure 8) shows that 68.8% of newcomers successfully contributed to OSS projects by solving GFIs. Therefore, for newcomers, it is often a good idea to start with GFIs, which should be noticed. In RQ3, we identify the problems and challenges that newcomers may meet when solving GFIs. Clarifying these issues can help newcomers join OSS projects more easily. For example, we find that some newcomers complained that it is hard to start on a new project and get timely help from mentors. Therefore, when newcomers look for a project to contribute to, they may check first whether they can get decent documentations and available support from the project. The newcomers can also learn from the criteria of identifying GFIs, e.g., they may evaluate whether the GFI's description indicates where the issue is, and whether it fits their scope and skill. From another perspective, these findings can help projects build a more friendly environment for newcomers (e.g., identify appropriate GFIs), which in turn can reduce the barriers for newcomers to join a project to some extent.

[17] We merged "*documentation*" and "*doc*" because they are the same.

8.2 Project Members

Our findings provide practical implications for optimizing onboarding process. The results of RQ1 and RQ2 show that more and more projects are helping newcomers onboard through GFIs, which actually play a role. Therefore, other OSS projects that want to bring in newcomers can also try this mechanism. However, many practical problems and difficulties still exist. We expect to help project members optimize this mechanism by identifying these issues and discussing the possible strategies.

1) *Set clear criteria for GFIs.* The results for RQ3 indicate that many newcomers perceived that their failed contribution was due to inappropriate GFIs. It means that to solve these GFIs, they need deep knowledge of the projects and spend a lot of time, which is beyond the capabilities of most newcomers. Worse still, some project members are unclear about the criteria of GFIs. For example, a project member, a respondent to our survey for RQ4, says, "*these boundaries are also murky for me.*" The seven general rules extracted to identify GFIs (RQ4) can provide valuable references for practice. In particular, whether issue descriptions have the relevant information (e.g., *version* and *platform*) is a critical indicator for GFI.

2) *Adopt a strict but flexible GFIs allocation process.* Despite the small number of GFIs in a project (as shown in Figure 4 and Table 3), 40.9% of GFIs were not solved by newcomers (see Figure 7). Part of reason is that the GFIs often get snapped up quickly by experienced developers, which makes GFIs fail to achieve their desired effect. Therefore, project members could put a limit on who can pick the GFIs. There is also a chaotic allocation of GFIs among newcomers: a GFI is picked up by multiple newcomers simultaneously. It makes newcomers frustrated by wasting their efforts. Introducing tags like "taken"/ "not taken" and at least one maintainer/collaborator is watching a GFI should be helpful.

3) *Build friendly environment for newcomers.* Table 3 shows that newcomers have difficulty starting on a new project and lack the timely guidance of mentors. Therefore, projects could prepare clear documentations introducing how to set up local environment, fold structures, code style, and development process etc. GFIs should either be assigned to mentors, or come with specific instructions. Be patient to newcomers whenever they meet difficulties. Encourage and praise newcomers when they make a little progress. As reported, attention helps cultivate long-term contributors [43].

8.3 Researchers

Researchers can build on top of our results and open research questions to deeply understand and improve onboarding process for newcomers. For example, based on the criteria for identifying GFIs obtained in RQ4, future research can explore how to measure these criteria and then automatically identify GFIs. The results for RQ3 show that the factors affecting the effect of GFIs are complicated and various. In fact, we have not conducted in-depth research on some factors. For example, we find that *insufficient GFIs* are identified as a factor associated with ineffective mechanism, but it is unclear

whether this is because some appropriate issues are not marked as GFIs or whether such issues are actually rare in projects. Future work can explore the reasons behind these factors and find more valuable factors and solutions. The essential reason for the failure of this mechanism is the mismatch between project members and newcomers. The study of cognitive differences between experts and novices should also be an interesting question in the future.

9 THREATS TO VALIDITY

Construct Validity concerns the relationship between the treatment and the outcome. Threat comes from the rationality of the questions asked. We are interested in assessing GFIs in GitHub. To achieve this goal we focus on its application status, effect, problems, and practices. We believe that these questions have high potential to provide unique insights and value for practitioners and researchers.

Internal Validity concerns the threats to how we perform our study. The first threat relates to the identification of GFIs (Section 3.2). To identify GFIs, we only consider the labels that obviously indicate friendliness to newcomers. This may have missed some GFIs (e.g., some projects recommend newcomers to start with "help wanted" issues, while others do not) but ensured the quality of the dataset. The second threat relates to the identification of newcomers (Section 5.1.1). To identify newcomers, we focus on the number of commits that developers contribute to a certain project, which may misrepresent of their actual contributions (e.g, report issues) and omit their contributions to other projects. Besides, we use "less than three commits" to identify newcomers, which may bring risks in some cases, e.g., just after solving a GFI, newcomers are capable of solving other issues. To mitigate this risk, we also observe the contributors who have not contributed any commit (commonly used in the literature) and obtain the similar results, e.g., only 45.1% GFIs were solved by newcomers. The third threat relates to the selection of GFIs when answering RQ2 (Section 5.2.2) and RQ3 (Section 6.1.1). Although we select a sample of GFIs, the margins of error are acceptable [3]. The forth threat comes from the data we use to extract the criteria for identifying GFIs (RQ4: Section 7.1.1) because we can not ensure all the project members hold the right judgement for GFIs. To mitigate this threat, we only consider the standards mentioned by multiple project members as the final results. The last threat relates to the themes extracted by thematic analysis (RQ3: Section 6.1.2, RQ4: Section 7.1.1). We acknowledge that the choice of these themes is to some extent subjective. For example, it is possible that different researchers reach a different set of criteria, than the ones proposed in Section 7.2.1. To mitigate this threat, the initial selection of themes is performed independently by the two authors of this paper. After this initial proposal, we compare our list of codes and themes, and develop a coding guide with definitions and examples for each identified theme. Each author then use the coding guide to independently analyze the complete set of data. Based on a discussion of the second round of analysis, the coding guide is further refined, and the data are independently analyzed for a third and final time.

External Validity concerns the threats to generalize our findings. When conducting this study, we only focus on the popular open source projects on GitHub (Section 3.1). While onboarding newcomers is important for all the projects, the GFI mechanism is rarely applied to less popular projects as discovered in this study. This is probably because newcomers tend to work for more visible projects, and on the other hand, senior developers in the less popular projects do not have much time to train newcomers [43]. However, the findings of this study, particularly the criteria to label GFIs should benefit any project that attempts to help newcomers onboard. Whether or not newcomers are willing to get onboarding in a particular project is a different story.

10 CONCLUSION

Numerous OSS projects rely on the contributions of volunteers from all over the world. Therefore, to remain sustainable, these projects need a constant influx of new volunteers, or newcomers. However, newcomers face many barriers during their onboarding process. The primary challenge faced by newcomers is finding appropriate initial tasks. To alleviate this problem, the direct solution is to recommend tasks (GFIs) to newcomers, which is also advocated by GitHub. However, many newcomers still fail to contribute through GFIs.

Therefore, we conduct a preliminary study on this mechanism from its application status, effect, problems, and best practices. We find that this mechanism has been increasingly adopted by the projects in GitHub. We also find that GFIs usually require more time to be solved, whereas for the other aspects of the resolution process (e.g., the number of comments), GFIs do not present significant difference compared to other issues. To analyze the effect of this mechanism, we find that although some newcomers successfully solved GFIs, most of them are one-time contributors. We analyze the factors and problems related to the effectiveness of this mechanism and identify various problems, e.g., insufficient and inappropriate GFIs. Eventually, in view of these problems, we discover the criteria for identifying appropriate GFIs and discuss the corresponding strategies to help newcomers onboarding.

Helping newcomers contribute to OSS projects has always been an important concern of practitioners and researchers, while assisting them to find the right task is the first step to their success. Our work investigates this mechanism from multiple aspects, which can help to better understand the barriers in newcomers onboarding and the difficulties in the guiding process, and thus finding ways to avoid them.

To facilitate replications or other types of future work, we provide the data, scripts, and detailed coding guidelines used in this study online: https://github.com/SunflowerPKU/FSE20Dataset.

ACKNOWLEDGEMENT

This work is supported by the National key R&D Program of China Grant 2018YFB1004201, the National Natural Science Foundation of China Grant 61825201. We would also like to thank Peng Cheng Laboratory for its support of computing resources.

REFERENCES

[1] Shaosong Ou Alexander Hars. 2002. Working for free? Motivations for participating in open-source projects. *International journal of electronic commerce* 6, 3 (2002), 25–39.

[2] Mohammad Y. Allaho and Wang-Chien Lee. 2013. Analyzing the Social Ties and Structure of Contributors in Open Source Software Community. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks*

Analysis and Mining. Association for Computing Machinery, New York, NY, USA, 56–60.

[3] J. Barlett. 2001. Organizational research: determining appropriate sample size in survey research. *Information Technology, Learning, and Performance Journal* 19 (01 2001), 43–50.

[4] Shahab Bayati. 2019. Effect of Newcomers' Supportive Strategies on Open Source Projects Socio-Technical Activities. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, USA, 49–50.

[5] Hoda Baytiyeh and Jay Pfaffman. 2010. Volunteers in Wikipedia: Why the community matters. *Journal of Educational Technology & Society* 13, 2 (2010), 128–140.

[6] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, USA, 275–284.

[7] Donald Farrar and Robert Glauber. 1967. Multicollinearity in Regression Analysis: The Problem Revisited. *The Review of Economics and Statistics* 49 (02 1967), 92–107. https://doi.org/10.2307/1937887

[8] J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. huff. 2007. *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects*. MITP, USA, 3–21. https://ieeexplore.ieee.org/document/6277090

[9] Catherine O Fritz, Peter E Morris, and Jennifer J Richler. 2012. Effect size estimates: current use, calculations, and interpretation. *Journal of experimental psychology: General* 141, 1 (2012), 2.

[10] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, Piscataway, NJ, USA, 233–236.

[11] Pieter Hooimeijer and Westley Weimer. 2007. Modeling Bug Report Quality. In *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*. Association for Computing Machinery, New York, NY, USA, 34–43.

[12] Yonghui Huang, Daniel Costa, Feng Zhang, and Ying Zou. 2018. An empirical study on the issue reports with questions raised during the issue resolving process. *Empirical Software Engineering* 24 (08 2018). https://doi.org/10.1007/s10664-018-9636-3

[13] Carlos Jensen, Scott King, and Victor Kuechler. 2011. Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists. In *2011 44th Hawaii international conference on system sciences*. IEEE, USA, 1–10.

[14] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. 2011. The onion patch: migration in open source ecosystems. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, USA, 70–80.

[15] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th working conference on mining software repositories*. ACM, USA, 92–101.

[16] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2016. Using dynamic and contextual features to predict issue lifetime in GitHub projects. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, USA, 291–302.

[17] Robert E Kraut and Paul Resnick. 2012. *Building successful online communities: Evidence-based social design*. MIT Press, USA.

[18] Rajiv Krishnamurthy, Varghese Jacob, Suresh Radhakrishnan, and Kutsal Dogan. 2016. Peripheral Developer Participation in Open Source Projects: An Empirical Analysis. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article Article 14 (Jan. 2016), 31 pages. https://doi.org/10.1145/2820618

[19] Amanda Lee, Jeffrey C. Carver, and Amiangshu Bosu. 2017. Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, USA, 187–197.

[20] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. ACM, USA, 63–70.

[21] Douglas R McCallum and James L Peterson. 1982. Computer-based readability indexes. In *Proceedings of the ACM'82 Conference*. ACM, USA, 44–48.

[22] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (01 Oct 2016), 2146–2189. https://doi.org/10.1007/s10664-015-9381-9

[23] Christopher Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret Burnett. 2018. Open source barriers to entry, revisited: A sociotechnical perspective. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, USA, 1004–1015.

[24] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empirical Software Engineering* 22, 6 (2017), 3219–3253.

[25] Nadim Nachar et al. 2008. The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology* 4, 1 (2008), 13–20.

[26] Sebastiano Panichella. 2015. Supporting newcomers in software development projects. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, USA, 586–589. https://doi.org/10.1109/ICSM.2015.7332519

[27] Chao Ying Joanne Peng, Kuk Lida Lee, and Gary M. Ingersoll. 2002. An Introduction to Logistic Regression Analysis and Reporting. *Journal of Educational Research* 96, 1 (2002), 3–14.

[28] Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. Piscataway, NJ, USA, 133–142.

[29] Sonali K. Shah. 2006. Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Manage. Sci.* 52, 7 (July 2006), 1000–1014. https://doi.org/10.1287/mnsc.1060.0553

[30] B. Shibuya and T. Tamai. 2009. Understanding the process of participating in open source communities. In *2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. ACM, USA, 1–6. https://doi.org/10.1109/FLOSS.2009.5071352

[31] Emad Shihab, Ahmed E Hassan, Bram Adams, and Zhen Ming Jiang. 2012. An industrial study on the risk of software changes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, USA, 1–11.

[32] Igor Steinmacher, Ana Paula Chaves, Tayana Uchoa Conte, and Marco Aurelio Gerosa. 2014. Preliminary empirical identification of barriers faced by newcomers to Open Source Software projects. In *2014 Brazilian Symposium on Software Engineering*. IEEE, USA, 51–60.

[33] Igor Steinmacher and Marco Aurélio Gerosa. 2015. Understanding and Supporting the Choice of an Appropriate Task to Start With In Open Source Software Communities. In *48th Hawaii International Conference on System Sciences (HICSS-48)*. IEEE, USA, 5299–5308.

[34] 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* PP (01 2018), 1–1. https://doi.org/10.1109/MS.2018.110162131

[35] Xin Tan and Minghui Zhou. 2019. How to Communicate When Submitting Patches: An Empirical Study of the Linux Kernel. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 108 (Nov. 2019), 26 pages. https://doi.org/10.1145/3359210

[36] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*. Association for Computing Machinery, New York, NY, USA, 356–366.

[37] Georg Von Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin W. Wallin. 2012. Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS Q.* 36, 2 (June 2012), 649–676.

[38] Georg Von Krogh and Eric Von Hippel. 2003. Special issue on open source software development.

[39] Lev Vygotsky. 1978. Interaction between learning and development. *Readings on the development of children* 23, 3 (1978), 34–41.

[40] Yunwen Ye and Kouichi Kishida. 2003. Toward an understanding of the motivation of open source software developers. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE Computer Society, USA, 419–429.

[41] Minghui Zhou. 2019. Onboarding and Retaining of Contributors in FLOSS Ecosystem. In *Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability*. Springer, USA, 107–117.

[42] Minghui Zhou and Audris Mockus. 2010. Developer fluency: Achieving true mastery in software projects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, USA, 137–146.

[43] Minghui Zhou and Audris Mockus. 2012. What make long term contributors: Willingness and opportunity in OSS community. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, USA, 518–528.