# Characterizing and Predicting Good First Issues

Yuekai Huang[1,3], Junjie Wang[1,2,3,*], Song Wang[4], Zhe Liu[1,3], Dandan Wang[1,3], Qing Wang[1,2,3,*]
[1]Laboratory for Internet Software Technologies, [2]State Key Laboratory of Computer Sciences,
Institute of Software Chinese Academy of Sciences, Beijing, China;
[3]University of Chinese Academy of Sciences, Beijing, China; [*]Corresponding author
[4]Lassonde School of Engineering, York University, Canada;
huangyuekai18@mails.ucas.ac.cn,junjie@iscas.ac.cn,wq@iscas.ac.cn

## ABSTRACT

**Background.** *Where to start contributing to a project* is a critical challenge for newcomers of open source projects. To support newcomers, GitHub utilizes the Good First Issue (GFI) label, with which project members can manually tag issues in an open source project that are suitable for the newcomers. However, manually labeling GFIs is time- and effort-consuming given the large number of candidate issues. In addition, project members need to have a close understanding of the project to label GFIs accurately.

**Aims.** This paper aims at providing a thorough understanding of the characteristics of GFIs and an automatic approach in GFIs prediction, to reduce the burden of project members and help newcomers easily onboard.

**Method.** We first define 79 features to characterize the GFIs and further analyze the correlation between each feature and GFIs. We then build machine learning models to predict GFIs with the proposed features.

**Results.** Experiments are conducted with 74,780 issues from 10 open source projects from GitHub. Results show that features related to the semantics, readability, and text richness of issues can be used to effectively characterize GFIs. Our prediction model achieves a median AUC of 0.88. Results from our user study further prove its potential practical value.

**Conclusions.** This paper provides new insights and practical guidelines to facilitate the understanding of GFIs and the automation of GFIs labeling.

## CCS CONCEPTS

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

## KEYWORDS

Machine Learning, Open Source Software, Issue Report, Newcomers

**ACM Reference Format:**
Yuekai Huang[1,3], Junjie Wang[1,2,3,*], Song Wang[4], Zhe Liu[1,3], Dandan Wang[1,3], Qing Wang[1,2,3,*]. 2021. Characterizing and Predicting Good First Issues. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '21), October 11–15, 2021, Bari, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3475716.3475789

## 1 INTRODUCTION

Open source software (OSS) is becoming increasingly popular. People from all over the world can contribute to open source software anytime and anywhere. This enables many open source software to be developed and maintained continuously, coming up with high quality software. There are 60M+ new repositories created in GitHub, one of the world's largest open source platforms, from October 2019 - September 2020 [19].

The continuous improvement of an open source software project depends on the developers who contribute to it, but many developers may not be able to participate in an open source software project for a long term [31]. In order to keep the vitality of open source software, newcomers are highly needed [21]. On the one hand, newcomers also eagerly participate in OSS projects, driven by the factors such as extrinsic benefits (e.g., better jobs) and intrinsic motivations (e.g., enjoyment) [4, 30, 50, 57]. On the other hand, there are many barriers which might hind the newcomers from successful onboarding [14, 31, 36, 41, 43, 44]. For example, the newcomers often face the problem of not knowing where to start contributing to an OSS project, which prevents them from joining the open source community.

To better guide newcomers to contribute to open source projects, project members often label issues that are suitable for newcomers to solve with special labels, e.g., the Good First Issue (short for GFI in this paper) label provided by GitHub, with which newcomers can quickly begin to contribute to the projects.

However, for most projects on GitHub, the percentage of GFIs is quite low, and some projects do not even have GFIs [47]. This may be due to the fact that labeling GFIs often requires developers to have a close understanding of the projects, which is quite time- and effort- consuming. Besides, considering the large number of candidate issues and the rapid increase of issues created in the open source projects, it is unrealistic for the project members to inspect each issue.

There have been many studies on newcomers in OSS projects, e.g., the barriers that newcomers face [36, 43], newcomers' motivations in contributing to an OSS project [4, 16, 40, 50, 57]. This paper aims at providing an automatic labeling approach for GFIs to reduce the burden of developers and in the meanwhile help newcomers easily onboard.

To achieve this goal, we first define 79 features from three dimensions, i.e., clearness of issue description, complexity of changes involved, and skills required, to characterize the GFIs. Specifically, the clearness of an issue description is measured by its reporter's experience, its text richness, and the readability of its textual description. The complexity of a change is measured from the points of influenced scope of the issue and the code complexity of the involved code. And we utilize the information related to the issue's type and the semantics of the issue's description to measure whether it only requires limited skills for solving an issue, i.e., doable for newcomers.

We then experiment with 74,780 issues from 10 OSS GitHub projects. We extract the above 79 features for GFIs and non-GFIs (i.e., issues that are not GFIs) respectively. Our statistical test shows that most of the features demonstrate significant differences between GFIs and non-GFIs, i.e., can be used to characterize the GFIs. Among these 79 features, features related to the semantics, the readability, and text richness of the issue play a larger contribution in GFIs characterization. The results indicate that the GFIs tend to be described with certain words, in better readability, with shorter and clear descriptions, and with more code snippets.

We further build machine learning models with the proposed features to predict GFIs. The results showed that the median AUC of the prediction is 0.88. We also include the effort-aware evaluation and find that one can retrieve a median of 42% total GFIs by inspecting a mere of 5% issues predicted by our model. Besides, we examine the relative contribution of each category of features in predicting GFIs with feature selection algorithms, aiming at providing actionable decision support for developers about choosing features in building GFIs prediction models.

Furthermore, we have conducted a user study on the newly-reported issues in GitHub to evaluate its potential practicability in real-world practice. Specifically, we run our prediction model on 48 newly-reported issues, and we then send the predicted GFIs to developers for confirmation. Among the 16 responses, 10 issues are confirmed as GFIs. Developers also show interest in our automatic GFIs prediction approach. This further proves the potential practical value of this work.

The main contributions of this paper are as follows:

- We define 79 features to characterize GFIs from three dimensions. **To the best of our knowledge this is the first study in GFIs characterization.**
- We experimentally characterize GFIs using the 79 features and demonstrate the feasibility of distinguishing GFIs from non-GFIs by using these features, which help understand GFIs.
- We build machine learning-based models to predict GFIs with the proposed features, and results are promising. In addition, we have further examined feature selection algorithms on the prediction performance to provide actionable decision support in model building.
- We evaluate the practical value of our GFIs prediction approach in real-world practice with affirmative feedback.

## 2 DATASET

Our experimental data are collected from GitHub, one of the world's largest open source software sites. We crawl the projects with more than 10k stars, and randomly select 10 projects with the number of issues more than 3k. We collect all the issues in each project

**Table 1: Dataset used in this paper**

| Repo | Domain | Language | Stars | GFIs | non-GFIs |
|---|---|---|---|---|---|
| babel | Compiler | JavaScript | 37,926 | 103 | 5,333 |
| bitcoin | Client | C++ | 46,395 | 249 | 8,113 |
| jest | Framework | TypeScript | 33,331 | 148 | 8,269 |
| lighthouse | Plugin | JavaScript | 21,257 | 118 | 9,749 |
| skaffold | Tool | Go | 10,635 | 119 | 4,157 |
| packer | Tool | Go | 11,501 | 248 | 8,406 |
| graphql-engine | Sever | Haskell | 19,575 | 127 | 5,100 |
| minikube | Tool | Go | 19,777 | 202 | 5,539 |
| react-admin | Framework | TypeScript | 15,277 | 104 | 3,714 |
| server | Server | PHP | 12,423 | 239 | 14,743 |

from 2010/12/19 to 2020/07/19, all the comments of each issue, the commit history and the source code of each project.

We then identify the GFIs from the issues of each project based on their labels. Although GitHub officially provides the 'good first issue' label to denote the newcomer-friendly issue, each project usually has its customized labels which also indicate the issue is suitable for newcomers, e.g., 'easy', 'beginner', and 'minor bug'. Therefore, we use the labels suggested in [47] to retrieve the GFIs.

For the issues which are not marked as GFIs, we consider them as the candidate non-GFIs. However, we notice that a project might start to adopt the GFIs mechanism at the middle of its evolution period, or stop to mark the GFIs from a certain time. To alleviate the influence of this phenomenon, we discard the issues that were closed before the earliest GFI, as well as the issues that were created after the latest GFI. The remaining non-GFIs are left for further experiments. Table 1 demonstrates the details of the experimental dataset used in this work.

## 3 CHARACTERIZING GFIS

### 3.1 Studied Features

To characterize what makes a GFI, we first collect the features which might exert the difference between GFIs and non-GFIs. Motivated by existing studies [15, 47, 56], we collect the features from three dimensions: 1). ***clearness of issue description***, we assume GFIs should have clear issue description telling where errors occur or what changes need to be done. 2). ***complexity of changes involved***, we assume the scope of changes is an important factor to determine whether issues are suitable for newcomers, and GFIs should involve self-contained change, i.e., only touch a small part of the codebase. 3). ***skills required***, ideal GFIs should only require limited skills for be solved, because newcomers generally do not have rich experience. Based on these three dimensions, we design 79 features to characterize GFIs spanning across 7 categories, listed in Table 2.

*3.1.1* ***Clearness of Issue Description***. Ideally, GFIs should have a clear issue description telling what the problem was and where to make the changes. First, we assume the issue reporter's experience would influence whether the issue has a clear description. Second, the issue with rich textual descriptions is more likely to specify the errors to tackle and changes to make. Third, the readability of the issue also reflects whether a newcomer can easily get the described problem and required changes. Taken in this sense, we design these three categories of features to measure whether the issue has a clear issue description.

**Table 2: Studied features**

| Dimension | Category | Feature | | Description | P-Value & Effect Size | | | |
|---|---|---|---|---|---|---|---|---|
| Clearness of Issue Description (45) | Reporter Experience (4) | Reporter Role (3) | is_member | Whether the reporter of this issue is a project member / contributor / collaborator | # N | | | |
| | | | is_contributor | | # N | | | |
| | | | is_collaborator | | # N | | | |
| | | has_gfi | | Whether the reporter has previously proposed GFIs in this project | *** N | | | |
| | Text Richness (5) | Text Length (4) | length_title | Number of words in issue title / body / all comments / comment average | *** N | | | |
| | | | length_body | | *** N | | | |
| | | | length_all_comments | | *** M | | | |
| | | | length_avg_comments | | *** M | | | |
| | | comments_num | | Number of comments in issue | *** M | | | |
| | Readability (36) | ari_title/body/all_comments/avg_comments | | The readability of title / body / all comments / comments average | # N | # N | *** M | *** M |
| | | cli_title/body/all_comments/avg_comments | | | * N | # N | *** M | *** M |
| | | dcrs_title/body/all_comments/avg_comments | | | # N | *** N | *** M | *** M |
| | | dw_title/body/all_comments/avg_comments | | | *** N | *** N | *** M | *** S |
| | | fkg_title/body/all_comments/avg_comments | | | # N | # N | *** M | *** M |
| | | fre_title/body/all_comments/avg_comments | | | # N | # N | *** M | *** M |
| | | gf_title/body/all_comments/avg_comments | | | * N | *** N | *** S | *** S |
| | | lwf_title/body/all_comments/avg_comments | | | *** N | *** N | *** M | *** S |
| | | smog_title/body/all_comments/avg_comments | | | # N | *** N | *** S | *** S |
| Complexity of Changes Involved (11) | Influenced Scope (3) | url_num | | The number of URL | ** N | | | |
| | | code_num_body | | Number of code snippets in issue body | *** N | | | |
| | | code_num_comments | | Number of code snippets in issue comments | *** S | | | |
| | Code Complexity (8) | modified_files | | The number of modified files | *** S | | | |
| | | file_lines | | The number of lines of code for the most modified file | # N | | | |
| | | file_complexity | | The complexity of code for the most modified file | *** S | | | |
| | | file_gfi | | The number of GFIs related to the most modified file | # N | | | |
| | | change_num | | The modification times of the most modified file | *** N | | | |
| | | inserted_lines | | The number of lines inserted / deleted / modified of the most modified file | # N | | | |
| | | deleted_lines | | | * N | | | |
| | | modified_lines | | | ** N | | | |
| Skills Required (23) | Issue Types (8) | is_bug | | Whether the issue has a special label | # N | | | |
| | | is_documentation | | | # N | | | |
| | | is_duplicate | | | ** N | | | |
| | | is_enhancement | | | *** N | | | |
| | | is_help_wanted | | | # N | | | |
| | | is_invalid | | | # N | | | |
| | | is_question | | | # N | | | |
| | | is_wontfix | | | # N | | | |
| | Semantics (15) | Topic Number (3) | topic_num_title | The number of topics in the issue title / body / comments | *** N | | | |
| | | | topic_num_body | | *** N | | | |
| | | | topic_num_comments | | *** S | | | |
| | | GFI Likelihood (12) | GaussianNB_title/body/comments | Bayes score of issue title / body / comments | *** N | # N | *** S | |
| | | | MultinomialNB__title/body/comments | | *** S | *** M | *** M | |
| | | | BernoulliNB__title/body/comments | | *** S | *** M | *** L | |
| | | | ComplementNB__title/body/comments | | *** S | *** M | *** M | |

***p<0.001, **p<0.01, *p<0.05, #p≥0.05
N:Negligible, S:Small, M:Medium, L:Large

**Reporter Experience:** There are four features in this category to measure the issue reporter's experience. The first three features measure the role of the issue reporter, i.e., whether he/she is the member, the contributor, or the collaborator of the project. The developers with these roles are deeper involved in the development and maintenance of the project, and have a better understanding of the project. Therefore, they have a higher possibility to tell what the problem was and pinpoint the exact code that needed to be changed. Furthermore, the aforementioned different roles have different levels of understanding of the project, e.g., the contributor may be familiar with only certain module while the member would have a certain degree of understanding about the whole project. Therefore, we design them as separate features. The fourth feature measures whether the historical reported issues by the reporter are marked as GFIs in the project. If someone has reported GFIs in the past, he/she would be more likely to write a clear issue description.

**Text Richness:** There are five features in this category, i.e., four of them measuring the textual length and the other measuring the number of comments. We first measure the length of the title and description. We assume these two fields should be with detailed descriptions to contain enough knowledge which can help the newcomers to fully understand the issue. On the other hand, these two fields should also be without redundant information thereby the newcomers can quickly locate the key point. Similarly, we also measure the length of the comments, which are important complements to the description, from two aspects, i.e., the total length and the average length of all comments. In addition, we include the number of comments, supposing there should be a reasonable number of comments to provide supplemental information for the newcomers. For the features involving comments, we retrieve the time when the issue is labeled as a GFI, and only consider the comments posted before that time. This is because the subsequent comments would not help to characterize the GFIs. If an issue is not labeled as a GFI, we use all of its comments.

**Readability:** The features in this category are designed to measure the textual readability of the issues. Readability is mainly calculated based on the number of words, syllables, difficult words and so on in a sentence, which is used to measure the difficulty of understanding the sentence. We assume the textual readability of the issues can affect whether the newcomer can obtain a clear and quick understanding of the issue. Following [24, 60], we employ nine metrics to measure the readability, i.e., *Automated readability index (ari)* [39], *Coleman–Liau index (cli)* [12], *Dale–Chall readability score (dcrs)* [10], *number of difficult words (dw)*, *Flesch-Kincaid Grade Level (fkg)* [27], *Flesch reading ease (fre)* [18], *Gunning fog index (gf)* [20], *Linsear Write (lwf)* [28] and *SMOG Index (smog)* [33], and each of them are retrieved respectively for title, description, all comments, and in terms of the average comments. For example, *ari_title* in Table 2 means the *ari* score calculated for the title of the issue. We used Python package textstat[1] to calculate these features.

### 3.1.2 *Complexity of Changes Involved*.

The scope of the involved changes of the issue should also be considered, and an ideal GFI should only touch a small part of codebase. In this sense, we design two categories of features to measure the self-contained change. First, the potential influenced scope should be as small as possible; and second, the involved code should be as less as possible.

**Influenced Scope:** We first use the involved urls and code snippets in the description and comments to measure the potential influenced scope of the issue. We assume the number of urls and code snippets reflect the potential influenced scope and complexity of the issue, and they also indicate whether the issue contains necessary information (e.g., ways to reproduce or locate the bug) to be a GFI. Three features (e.g., *code_num_body*) are included in this category.

**Code Complexity:** To measure the code complexity, we extract the features from the source code for resolving the issue. We first build the link between the issue and the related code, following the method proposed in [5], i.e., if the commit message contains the related issue id, the issue and commit are linked. We then use the linked data to extract these related features. There are eight features to measure the code complexity. The number of modified files reflects the scope of the required changes for resolving the issue and the possibilities of the issue being GFIs. We also measure the complexity of the most relevant file (among the modified files) with the size of the file (i.e., *file_lines*), the complexity of the file (i.e., *file_complexity*), the change history of the file (e.g., *change_num*, *inserted_lines*), and the number of GFIs of the file (i.e., *file_gfi*).

### 3.1.3 *Skills Required*.

Ideally, the GFIs should only require limited skills so that the newcomers can manage to resolve them. From one point of view, we assume the type of the issue can potentially reflect how much experience is needed. From another point of view, we employ the semantics involved in the textual descriptions of the issue to indicate the required skills.

**Issue Types:** We assume the issues of certain types would be more likely to be GFIs, e.g., newcomers might be more suitable for improving the documentation than fixing a bug. We use the labels of the issue to denote the issue type, and we only consider the labels marked before GFI while ignoring the subsequent labels. We employ 8 features related to issue types, corresponding to the labels provided by GitHub[2].

**Semantics:** The features in this category fall into two parts. The first part involves the number of topics of the textual descriptions, and the second part captures the likelihood of being a GFI learned from the textual descriptions. Intuitively, a GFI should involve a limited number of topics and require limited skills to be understood and resolved. For the likelihood, we assume the GFIs might involve similar technical aspects and we calculate four types of Bayes scores on the textual descriptions of the issue learned from other GFIs, i.e., Gaussian Naive Bayes [58], Multinomial Naive Bayes [34], Bernoulli Naive Bayes [34] and Complement Naive Bayes [38]. All these features are calculated in terms of the title, body, and the comments of the issues.

For the topic number, We use the python package Gensim[3] to build the Latent Dirichlet Allocation (LDA) [8] topic model. It trains the topic model with the textual descriptions of all issues, calculates the topic probability distribution of each issue, and determines the topic through a threshold (default *0.01* in Gensim). For the Bayes

[1]https://github.com/shivam5992/textstat

[2]https://docs.github.com/en/free-pro-team@latest/github/managing-your-work-on-github/managing-labels#about-default-labels

[3]https://radimrehurek.com/gensim/

score, we use the python package Sklearn[4] to build the Bayes models and use them to predict each issue being a GFI, and employ the probability as the Bayes score.

These semantic features need to be calculated based on historical data of GFIs and non-GFIs. To retrieve these features, following [15], we randomly separate the training set into two subsets, train the model in one subset, and calculate the features in the other subset and the entire training set is used to calculate these features of the test set.

## 3.2 Quantitative Analysis

We characterize the GFIs with the hypothesis testing results about whether each of the aforementioned features exerts a difference between GFIs and non-GFIs. If a significant difference is observed, we consider the corresponding feature can be utilized to characterize the GFIs, and vice visa. In detail, we extract all the features from the dataset and perform the Wilcoxon rank-sum test between GFIs and non-GFIs. We include the Bonferroni correction [53] to counteract the impact of multiple hypothesis tests. Besides the *p-value* for signifying the significance of the test, we also present the *Cliff's delta* to demonstrate the effect size of the test. And we use the commonly-used criteria to interpret the effectiveness levels, i.e., Large (above 0.474), Median (0.33-0.474), Small (0.147-0.33), and Negligible (less than 0.147) (see details in [22]). We highlight the cell which has a significant difference between GFIs and non-GFIs, and the larger the effect size, the deeper color of the cell.

The null hypothesis and alternative hypothesis are as follows:

- H0: there is NO difference between the feature values of GFIs and non-GFIs.
- H1: there is a difference between the feature values of GFIs and non-GFIs.

The last column in Table 2 demonstrates the quantitative results, and the next paragraphs present the analysis of these results in terms of each category of features.

***Reporter Experience.*** From Table 2, we can see that reporter's role demonstrates no significant difference between GFIs and non-GFIs, while *has_gfi* (i.e., whether the reporter's submitted issues being marked as GFIs) exerts significant difference (i.e., p-value is smaller than 0.001). This indicates the features related to the reporter's experience have minor contributions in characterizing the GFIs.

***Text Richness.*** All the features in this category exert a significant difference (i.e., p-value is smaller than 0.001) between the GFIs and non-GFIs, and especially for these features obtained in terms of the comments of the issue (i.e., the effect size is median). In detail, the dataset shows that the number of comments and length of the comments are significantly smaller for GFIs than non-GFIs. This might because for the issues with more or lengthy comments (potential more discussion and clarification about the problems and solution), it may indicate they are more difficult and is not suitable for newcomers.

***Readability.*** A large portion of the features related to the textual readability exert a significant difference between GFIs and non-GFIs, and for the readability of the comments, all related features demonstrate substantial difference (i.e., effect size is small to median). Furthermore, the dataset also reveals that if the comments are easy to read (e.g., contain less difficult words, use words with fewer syllables), the corresponding issue is more likely to be a GFI. In addition, some of the readability features obtained in terms of issue title or body also exert significant difference, which further demonstrates the importance of description's readability in distinguishing GFIs from non-GFIs.

[4]https://github.com/scikit-learn/scikit-learn/

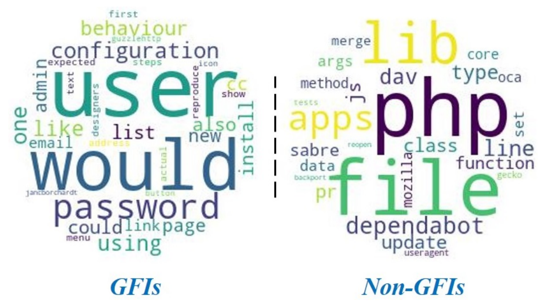


**Figure 1: Wordcloud for GFI and non-GFI of project *server***

***Influenced Scope.*** The features about an issue's influenced scope exert a significant difference between GFIs and non-GFIs. Both the number of code snippets in the issue body and in the comments can indicate whether an issue is a GFI. We further find that GFIs tend to have more code snippets in the issue body to help describe and clarify the issue.

***Code Complexity.*** Few features about code complexity exert a significant difference between GFIs and non-GFIs. Specifically, the number of modified files and the complexity of the modified file demonstrate a substantial difference (i.e., the effect size is small) between GFIs and other issues. Unsurprisingly, we find that, for GFIs, there are fewer modified files and the modified files are less complex, which indicates the less difficulty in resolving the GFIs.

***Issue Types.*** For features about issue types, certain label (i.e., *is_enhancement*) can help distinguish GFIs and non-GFIs. Our observation from the dataset reveals that GFIs are more likely to be enhancement issues. One of the possible reasons for this can be that enhancement labels often indicate feature requests, which involve less modification of existing code. Other issue types, e.g., whether it is a bug, do not exert a significant difference between GFIs and non-GFIs, indicating the GFIs can possess a wide range of types.

***Semantics.*** Most features related to the issue's semantics exert a significant (i.e., p-value is smaller than 0.001) and substantial (i.e., effect size is small to large) difference between GFIs and non-GFIs. The title, the body, and the comments of the issues can contribute to distinguishing the GFIs. Surprisingly, we find that the number of topics of issues' title or body of GFIs is larger than non-GFIs. This might because GFIs tend to be described in detail with more words to provide a clear description, while some non-GFIs might be described roughly with fewer words and fewer number of topics. In addition, the significant difference between GFIs and non-GFIs for the features related to Bayes scores implies that there exist indicative words in GFIs.

Figure 1 demonstrates an example of these indicative words of GFIs and non-GFIs for project *server*, which confirms GFIs and non-GFIs have a different distribution of words in their discussions, and the semantics of issues' textual descriptions is a good source of information for distinguishing these two types of issues.

**To summarize,** most of the investigated features exert a significant and substantial difference between GFIs and non-GFIs, indicating they can be utilized for characterizing GFIs. Specifically, features related to the semantics, readability, and text richness of the issues play a larger contribution in GFIs characterization. In addition, GFIs tend to be described with certain words, in better readability, with shorter and clearer descriptions, and associated with more code snippets in the issue description, to help describe and clarify the issue.

## 4 EXPERIMENT DESIGN

### 4.1 Research Questions

This section aims at exploring to what extent these features proposed in Section 3 can be employed to predict the GFIs, so as to assist the developers in automatically labeling the issues for newcomers. The following questions will be answered.

- **RQ1**: Are the proposed features good at predicting GFIs?
- **RQ2**: What is the contribution of each category of features in predicting GFIs?
- **RQ3**: Can a reduced feature set achieve comparable performance compared to the full feature set?

RQ1 evaluates to what extent the proposed features can predict the GFIs. RQ2 examines the relative contribution of each category of features in predicting GFIs, which aims at understanding these features. RQ3 examines the performance when using feature selection algorithms, which aims at providing actionable decision support for developers when choosing features in building GFIs prediction models.

### 4.2 Experiment Setup

To answer RQ1, following existing studies [6, 9, 29], we conduct the 5-fold cross validation on each project, record each model's performance and treat the average value as the final performance of the project to avoid bias. To reduce the impact of data distribution on model performance, we use stratified k-folds to ensure consistent data distribution between the training data and the testing data. The training set is oversampled using SMOTE [11] to balance the data, while we keep the testing data as it is to maintain its true distribution. Note that, since the features related to `Code Complexity` can only be retrieved after the issue is closed, we omit this category of features and employ other six categories for model building.

We experiment with the following four commonly-used machine learning algorithms to evaluate the GFIs prediction performance, i.e., Logistic Regression (LR) [7], Support Vector Machine (SVM) [13], Random Forest (RF) [23], and Multilayer Perceptor (MLP) [35].

To answer RQ2, we use the same experimental setup as RQ1. We first experiment with each category of features and obtain the performance of GFIs prediction; we then conduct experiments after removing each category of features from the full feature set.

To answer RQ3, we experiment with three commonly-used feature selection algorithms. In detail, SelectFromModel (SFM) is an algorithm based on the estimator's attribute like coefficients [1]. SequentialFeatureSelector (SFS) is a greedy based algorithm that adds the best feature every iteration [2, 17]. Recursive Feature Elimination (RFE) is an algorithm that recursively removes the least important features [3]. Furthermore, we additionally employ other two feature subset choices, i.e., using all the significant features in Table 2 (short for *SIG_FET*) and using all the features whose effect size is not negligible in Table 2 (short for *H_SIG_FET*).

### 4.3 Baselines

To further demonstrate the effectiveness of our approach, we compare it with the state-of-the-art baseline and a random approach.

***TextPredictor* [42]** : is a natural language processing based prediction approach, which automatically identifies the issues that newcomers can resolve. It first vectorizes the textual descriptions in the issue report by term frequency and inverse document frequency (TF-IDF). Together with the features from sentiment analysis and the number of words in the issue, it builds machine learning models for the prediction.

***Random***: simulates the scenario that the developers assign the GFIs label randomly. In detail, we randomly label a subset of issues as GFIs, and calculate the performance of random labeling. The process is repeated 10 times and the average metric values are used to evaluate its performance.

### 4.4 Evaluation Metrics

As shown in Table 1, the number of GFIs are highly imbalanced among all the issues of a project, thus we use AUC, which is widely utilized in measuring the performance of prediction models for imbalanced data [32, 48, 52, 56], for measuring the performance.

**AUC** refers to the area under the Receiver Operating Characteristic (ROC) curve, which measures the overall discrimination ability of the model. The ROC curve is drawn by calculating the true positive rate and false positive rate under various thresholds. AUC for a perfect model would be 1.0, and a model is considered applicable if its AUC is larger than 0.7 [49].

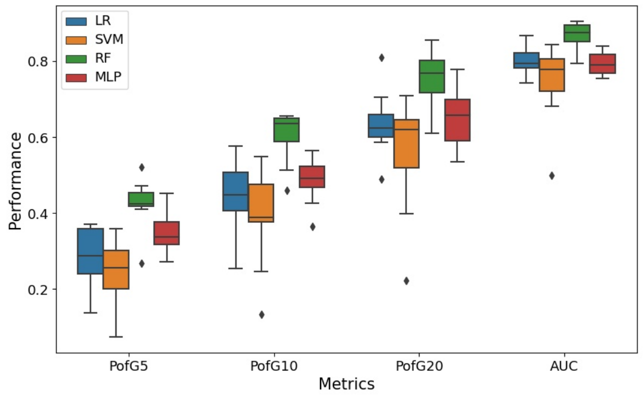
In addition, we also adopt the effort-aware evaluation and use a metric, i.e., PofG{K}, borrowing from defect prediction studies (also imbalanced scenario) [26, 46, 56], to measure the performance.

**PofG{K}** is defined as the percentage of true GFIs identified by inspecting the top *K%* issues predicted by the model. A higher PofG{K} indicates that one can identify more GFIs when inspecting a limited number of total candidates. In our experiments, we set K as 5, 10, and 20 (i.e., PofG5, PofG10, and PofG20) to obtain a thorough view of the prediction performance.

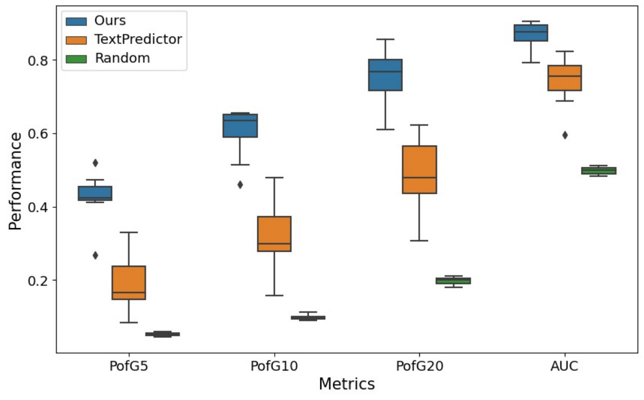
## 5 RESULT ANALYSIS

### 5.1 Answering RQ1

Figure 2(a) presents the PofG5, PofG10, PofG20, and AUC values for each experimental project under different machine learning models. Overall, as we can see, among the four types of machine learning based models, RF can outperform others in all four metrics. For the RF-based model, with the proposed features, it can achieve 0.79 to 0.91 AUC, with a median AUC of 0.88. The high AUC values indicate that our proposed features are good at predicting the GFIs. Furthermore, the median PofG5 of RF-based model is 0.42, and the median PofG20 is 0.77, indicating that by inspecting a mere of 5%



(a) Performance under different machine learning algorithms



(b) Performance under different approaches

**Figure 2: Performance of GFI prediction**

issues predicted by our model, a median of 42% total GFIs can be retrieved, and by inspecting a mere of 20% issues predicted by our model, about 77% total GFIs can be retrieved.

The best model, i.e., RF, is an ensemble learning method, which consists of multiple estimators, and even if some of them obtain wrong predictions, others can correct the results, which makes it perform well in GFIs prediction. MLP, which is a deep learning algorithm, is supposed to perform well. Yet it is inferior to the random forest model. This might because the model is good at utilizing large amount of training parameters to fit the data. But in this scenario, the GFIs only account for a small proportion of all issues, even over-sampling is applied to balance the data, there is less information to learn. Besides, the over-sampling would potentially amplify the noise in the data, and easily mislead the MLP model. Hence, in the following experiment, if not specified, we will use random forest model.

***Comparison with baselines.*** In order to demonstrate the advantages of the features we proposed, we compare our approach with two baselines (see Section 4.3). The results are shown in the Figure 2(b). We can see that the performance of our approach is significantly higher than the two baselines, which further indicates the effectiveness of the proposed features in predicting GFIs.

The random baseline achieves quite lower performance, indicating a well-designed approach is necessary for GFIs prediction. Another baseline, i.e., *TextPredictor*, uses TF-IDF to model the textual descriptions, and consider the sentiment score and number of words in the issue, to jointly predict the GFIs. In comparison, our approach employs the topic model and four Bayes scores to model different aspects of the textual descriptions. We also consider other issue features as the readability and text richness to promote the prediction. This again suggests the necessity and effectiveness of our proposed features.

Our GFIs prediction approach can achieve a median AUC of 0.88 with the proposed features, which significantly outperforms the state-of-the-art baseline.

**Table 3: P-value and effect size between performance of the prediction model with the full feature set and a sub category of features.**

| Sub-Features | PofG5 | PofG10 | PofG20 | AUC |
|---|---|---|---|---|
| `Reporter Experience` | *** L | *** L | *** L | *** L |
| `Text Richness` | *** L | *** L | *** L | *** L |
| `Readability` | *** L | *** L | *** L | *** L |
| `Influenced Scope` | *** L | *** L | *** L | *** L |
| `Issue Types` | *** L | *** L | *** L | *** L |
| `Semantics` | *** L | *** L | *** L | *** L |

$^{***}p<0.001$, $^{**}p<0.01$, $^{*}p<0.05$
N:Negligible, S:Small, M:Medium, L:Large

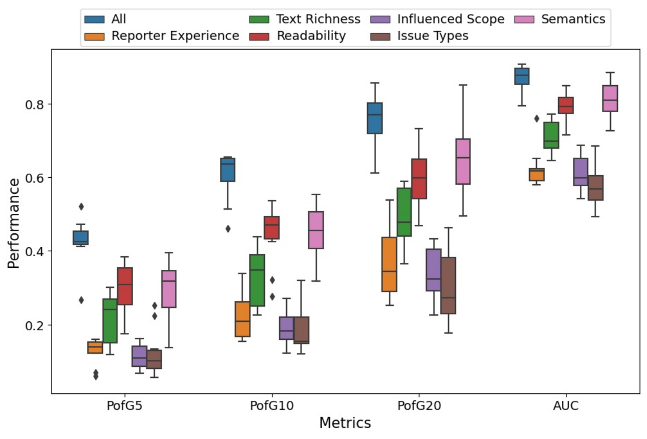
## 5.2 Answering RQ2

Figure 3 (a) demonstrates the GFIs prediction performance with each category of features, i.e., `Reporter Experience`, `Text Richness`, `Readability`, `Influenced Scope`, `Issue Types`, and `Semantics`. We also present the performance using all the proposed features (i.e., *All*) for comparison. Table 3 additionally presents the Wilcoxon signed-rank [54] test results to compare whether the performance with each category of features demonstrates significant difference with the performance using all features.

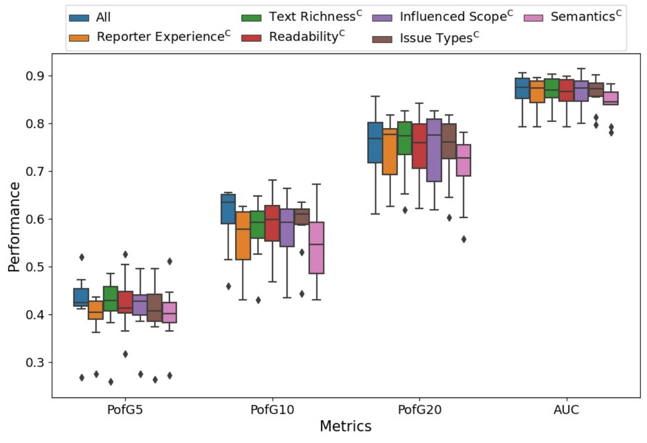
We can see that with any category of features, the GFIs prediction performance is significantly lower than the performance using the whole set of features.

Figure 3 (b) demonstrates the GFIs prediction performance when removing each category of features, as well as using all the proposed features (i.e., *All*). Table 4 additionally presents the Wilcoxon signed-rank test results to compare whether the performance using all features demonstrates significant difference with other settings.

We can see that when removing any category of features, the median PofG10 would decrease notably. Furthermore, when removing each category of features (except `Issue Types`), the prediction performance would undergo a significant decline in at least one of the four evaluation metrics.



(a) Performance of each category of features



(b) Performance of complement sets for each category of features

**Figure 3: Performance of subsets of features**

**Table 4: P-value and effect size between performance with all features and complement sets of features of different categories (the superscript $C$ means the subset features with excluding the specific category)**

| Sub-Features | PofG5 | PofG10 | PofG20 | AUC |
|---|---|---|---|---|
| `Reporter Experience`$^C$ | ** S | *** S | N | * N |
| `Text Richness`$^C$ | N | ** S | N | N |
| `Readability`$^C$ | N | * N | N | * N |
| `Influenced Scope`$^C$ | N | ** N | N | N |
| `Issue Types`$^C$ | N | N | N | N |
| `Semantics`$^C$ | ** S | *** M | *** S | *** S |

***p<0.001, **p<0.01, *p<0.05
N:Negligible, S:Small, M:Medium, L:Large

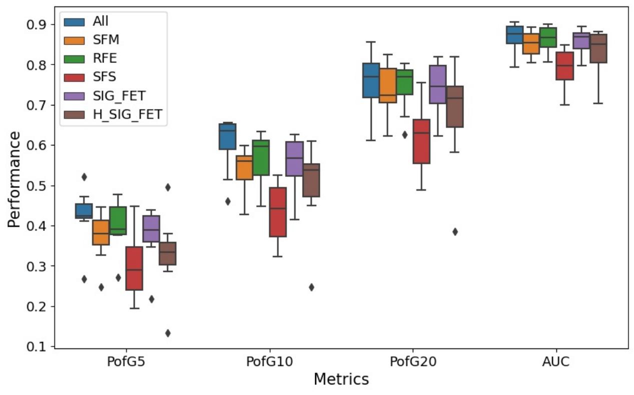


**Figure 4: Performance under different feature selection algorithms**

The experimental results indicate all categories of features can contribute to the GFIs prediction and the developers should try their best to collect all these categories of features in order to achieve satisfying prediction results.

Among the six categories of features, the performance would drop sharply when removing the `Semantics` features. This indicates the importance of this category of features, and is also consistent with the hypothesis testing results in Section 3.2. Besides, we also notice that, when using the `Readability` features for prediction, the performance is relatively high; yet when we remove the `Readability` features, the performance decline is not so significant. This implies that the contribution of the `Readability` features overlap with other features, e.g., the hard words might also potentially be considered in `Semantics` features.

In comparison, we can see that although the `Reporter Experience` has less significant features, it also has a significant impact on performance. When removing features related to `Reporter Experience`, the performance (especially PofG5 and PofG10) undergoes significant decline; yet when only using this category of features, the performance is relatively low. This might indicate, the information characterized by the reporter's experience is relatively unique and has less intersection the information characterized by other features. In other words, these features are less substitutable than other features.

Overall, all the categories of proposed features can contribute to the GFIs prediction and the `Semantics` category of features contribute the most to the prediction performance.

## 5.3 Answering RQ3

Figure 4 presents the performance of GFI prediction under different feature selection algorithms, as well as with all proposed features.

We can see that compared with using all proposed features, no matter what feature selection algorithm is utilized, the performance would decline.

Moreover, the same situation also occurs both in SIF_FAT and H_SIG_FAT, and compared with SIF_FAT, H_SIG_FAT decreases greatly, which shows that although some features have negligible effect size, they also have a big influence on the prediction of GFIs. Therefore, when discarding features is required, careful feature selection is required.

In general, the results indicate all the proposed features are useful in predicting GFIs, and we recommend the practitioners utilize all the features in model building.

## 6 DISCUSSION

### 6.1 User Study

The performance in Section 4 is obtained based on the historical issues with the marked GFI label. This section conducts an experiment on the newly reported issues to further examine whether these proposed features can accurately predict GFIs, and to investigate whether the developers think them useful.

In detail, we use the projects from Good First Issue collection website[5] as the candidate projects for experiment. Since these projects have at least one issue marked as GFIs, we suppose they are utilizing this mechanism. We crawl the issues created after the latest GFI's creation time, run our prediction model on these issues, and obtain the top 2 predicted GFIs of each project. For the issues with low probability (less than 0.5) to be a GFI, we ignore them.

We then submit a comment below the related issue, suggesting the issue can be marked as a GFI together with the primary idea about how we determine it. In total, 16 responses are received among the 48 submitted comments. Of all the responses, 10 issues are confirmed as GFIs, while 6 issues are denied. Due to the space limit, we put the detailed information on our website[6], and present few examples below.

Among the identified issues, some are confirmed directly by the developers, and several developers show interest in this research. Note that, the content within the brackets before the response (e.g., *solidity #10035*) denotes the project name and issue id.

- *(coq #13280) "yes, that seems reasonable, thanks"*
- *(amphtml #30592) "Yes, this could be a GFI. Cheers."*
- *(solidity #10035) "This is very interesting indeed! Could you identify other issues in this repository to be potentially labelled GFI?"*
- *(polar-bookshelf #1556) "I guess it should - but I'm not completely sure of the evaluation criteria of a GFI from the previous message. I found really interesting that an ML is analyzing what gets posted, keep continuing this :)"*

Some issues are considered to be non-GFIs because GFIs have special criteria in their projects. This further indicates the difficulties of GFIs auto-labeling, and the necessity of the exploration in this topic.

- *(oppia #10610) "In this repo, good first issues are tagged if the solution is known and/or have sufficient well-documented examples. This issue does not fall under the "good first issue" category because it requires debugging to figure out the root cause of the issue before it can be fixed."*
- *(joplin #4203) "No, not a good first issue. Spec is not fully developed and implementing this requires a relatively good knowledge of the codebase."*

In addition, developers also post constructive suggestions about using the GFIs mechanism. For example, one developer thinks that the total number of GFIs should not be too high, and the GFIs need to be strictly reviewed.

- *(jabref #6865) "I think, the discussion is also whether a good-first-issue should be something being a quick win for the contributor or whether it should be hard-to-do, but afterwards, the contributor knows nearly everything about the software. We optimize for the former when putting GFI labels. A nearly forgotten property of GFI issues is that the total number of GFI issues should not be too high. A GFI should be well-described and provide the newcomer a good start. This is hard work to do."*

### 6.2 Lower Cross-project Performance and Data Shift Problem

We also have conducted the cross-project GFI prediction, and Table 5 presents the AUC of cross project prediction by treating each project as testing data and every other project as training data in turn, with the worst performance in blue and best cross-project performance in red. Other metrics exert a similar trend, and we did not present them due to the space limit.

We can see that the cross-project prediction achieves lower performance than within project prediction. The average performance decreased by 11%. This coincides with the trend of cross-project report prediction [15]. Different projects can easily demonstrate different distribution of features caused by the developer habits, team organization, programming languages, etc. These can easily cause performance degradation in cross-project prediction.

We can also observe that the best performance (i.e., red value) tends to appear in the largest 5 training projects (projects with more than 8000 issues), while worst performance (i.e., blue value) tends to appear in the smallest 5 training projects (projects with less than 6000 issues), which indicates that enough training data can improve the performance of the model. Besides, we also find a relatively small project *react-admin* can predict project *jest* more accurately than other training projects, which might be due to the fact that they belong to the same domain and use the same programming language. Yet, more exploration is needed to better investigate the cross-project GFIs prediction and developers should carefully choose the training data when building prediction models with the data from other projects.

### 6.3 Threats to Validity

The internal threats are related to the acquisition of available information. During the feature extraction process, some issue features are affected by time. When we capture an issue, we get all the information from issue creation to closure, but for GFIs, some information is not generated when labeling as a GFI, which may result in confusion of information. We make some constraints time to mitigate this threat.

The threats to external validity concern the generality of this study. Our data are crawled from GitHub, one of the largest open

[5]https://goodfirstissue.dev/
[6]https://github.com/20210515/GFI

Table 5: Performance of cross-project prediction (AUC)

| Train / Test | babel | bitcoin | jest | lighthouse | skaffold | packer | graphql-engine | minikube | react-admin | server |
|---|---|---|---|---|---|---|---|---|---|---|
| babel | 0.88 | 0.73 | 0.71 | 0.67 | 0.61 | 0.70 | 0.69 | 0.64 | 0.73 | 0.74 |
| bitcoin | 0.71 | 0.79 | 0.72 | 0.67 | 0.60 | 0.68 | 0.69 | 0.69 | 0.71 | 0.69 |
| jest | 0.69 | 0.68 | 0.87 | 0.75 | 0.71 | 0.73 | 0.63 | 0.73 | 0.79 | 0.74 |
| lighthouse | 0.57 | 0.68 | 0.74 | 0.85 | 0.65 | 0.68 | 0.78 | 0.68 | 0.68 | 0.73 |
| skaffold | 0.74 | 0.74 | 0.77 | 0.83 | 0.90 | 0.80 | 0.72 | 0.83 | 0.77 | 0.84 |
| packer | 0.66 | 0.64 | 0.69 | 0.70 | 0.67 | 0.88 | 0.70 | 0.71 | 0.78 | 0.79 |
| graphql-engine | 0.72 | 0.80 | 0.73 | 0.76 | 0.77 | 0.76 | 0.90 | 0.78 | 0.74 | 0.78 |
| minikube | 0.66 | 0.68 | 0.74 | 0.75 | 0.70 | 0.75 | 0.68 | 0.91 | 0.75 | 0.73 |
| react-admin | 0.78 | 0.74 | 0.77 | 0.79 | 0.72 | 0.80 | 0.67 | 0.76 | 0.87 | 0.81 |
| server | 0.58 | 0.69 | 0.70 | 0.64 | 0.55 | 0.64 | 0.54 | 0.67 | 0.68 | 0.82 |

source software websites, and the selected projects are popular. The results of this study are based on these projects and may not be applicable to all scenarios, but the size of these projects can help reduce this threat.

The threats to construct validity are mainly derived from the metrics selected in our experiment. We use AUC and PofG to evaluate the performance of the model. The selection of these two metrics may have potential threats. However, these measures have been used in previous work [26, 56], so the threat is relatively small.

## 7 RELATED WORK

In recent years, many researchers focus on the motivation of contributors in OSS. Ye et al. [57] theorized that learning is one of the motivational forces for developers to participate in an OSS and discussed the significance of the theory in OSS and software engineering. Von Krogh et al. [50] reviewed the previous research related to the motivation of developers, and proposed a new theoretical framework. Allaho [4] conducted a statistical analysis on the social networks of OSS, and the results show that social relations have a significant impact on contributors. Krishnamurthy [30] pointed out that peripheral developers would be more involved if they are more likely to gain reputation.

There are also studies focusing on the barriers developers may encounter. Shibuya et al. [41] pointed out that the lack of appropriate tasks, up-to-date documents, new tools are the factors that hinder developers from joining the project. Steinmacher et al. [43] has identified 58 possible barriers that newcomers may face. Lee et al. [31] found that most developers just want to fix issues, rather than become long-term maintainers, and discussed four kinds of barriers the contributor faced. Mendez et al. [36] revealed the issues on both tools and infrastructure, and believed that there were six types of newcomer barriers related to tools and infrastructure.

In the face of the barriers, some scholars have conducted research on newcomers in OSS and how to attract more newcomers to join projects. Jensen et al. [25] found that for newcomers, whether they can receive a response in time affects whether they will participate in it. Zhou et al. [59] modeled the willingness and opportunity of the contributors and found out the factors for the newcomers to become long-term contributors. Wang et al. [51] designed a tool extension, which can provide visual bug information for developers and help them search for bugs they are interested in. Wolff-Marting et al. [55] made suggestions on both the procedural environment and the newcomers' confidence to help the newcomers save time in building the environment. Panichella et al. [37] defined several tools which can recommend suitable mentors for newcomers and help them understand the code. Steinmacher et al. [45] believes that some guidelines can help newcomers join the project, and puts forward suggestions in the three aspects of the contribution process, social behavior, and technology.

This paper aims at providing automatic GFIs labeling to help newcomers easily find the appropriate tasks, so as to increase the possibility of becoming long-term contributors.

## 8 CONCLUSION

In this work, we aim to identify the differences between GFIs and non-GFIs by analyzing the various features of issues and recommend candidate GFIs for developers to reduce their labeling burden. To achieve this goal, we first crawl ten large projects from GitHub and extract 79 features from them. Quantitative analysis reveals that most of the features are significantly different between GFIs and non-GFIs. After that, we build the machine learning based model with these features to predict GFIs. The results show that these features can well recommend candidate GFIs for developers.

It should be noted that the features used in model training in this paper are based on the text or attributes in the issue, and do not involve the code of the project. Therefore, in the future, we can extract more features by combining code localization to enhance our current approach.

## ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under grant No.2018YFB1403400, National Natural Science Foundation of China under Grant No. 62072442, No. 62002348.

## REFERENCES

[1] 2020. https://scikit-learn.org/stable/modules/feature_selection.html#select-from-model.

[2] 2020. https://scikit-learn.org/stable/modules/feature_selection.html#sequential-feature-selection.

[3] 2020. https://scikit-learn.org/stable/modules/feature_selection.html#rfe.

[4] Mohammad Y. Allaho and Wang-Chien Lee. 2013. Analyzing the social ties and structure of contributors in open source software community. In *Advances in Social Networks Analysis and Mining 2013, ASONAM '13, Niagara, ON, Canada - August 25 - 29, 2013*, Jon G. Rokne and Christos Faloutsos (Eds.). ACM, 56–60. https://doi.org/10.1145/2492517.2492627

[5] Adrian Bachmann and Abraham Bernstein. 2009. Software process data quality and characteristics: a historical view on open and closed source projects. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, Amsterdam, Netherlands, August 24-28, 2009*, Tom Mens, Kim Mens, and Michel Wermelinger (Eds.). ACM, 119–128. https://doi.org/10.1145/1595808.1595830

[6] Ting Bai, Wayne Xin Zhao, Yulan He, Jian-Yun Nie, and Ji-Rong Wen. 2019. Correction to "Characterizing and Predicting Early Reviewers for Effective Product Marketing on E-Commerce Websites". *IEEE Trans. Knowl. Data Eng.* 31, 4 (2019), 818. https://doi.org/10.1109/TKDE.2019.2894055

[7] Joseph Berkson. 1944. Application of the logistic function to bio-assay. *Journal of the American statistical association* 39, 227 (1944), 357–365.

[8] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (2003), 993–1022. http://jmlr.org/papers/v3/blei03a.html

[9] Leo Breiman and Philip Spector. 1992. Submodel selection and evaluation in regression. The X-random case. *International statistical review/revue internationale de Statistique* (1992), 291–319.

[10] Jeanne Sternlicht Chall and Edgar Dale. 1995. *Readability revisited: The new Dale-Chall readability formula.* Brookline Books.

[11] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[12] Meri Coleman and Ta Lin Liau. 1975. A computer readability formula designed for machine scoring. *Journal of Applied Psychology* 60, 2 (1975), 283.

[13] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[14] Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, and Jacqueline de Vries. 2010. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel (Eds.). ACM, 275–284. https://doi.org/10.1145/1806799.1806842

[15] Yuanrui Fan, Xin Xia, David Lo, and Ahmed E. Hassan. 2020. Chaff from the Wheat: Characterizing and Determining Valid Bug Reports. *IEEE Trans. Software Eng.* 46, 5 (2020), 495–525. https://doi.org/10.1109/TSE.2018.2864217

[16] 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.* 3–21.

[17] Francesc J Ferri, Pavel Pudil, Mohamad Hatef, and Josef Kittler. 1994. Comparative study of techniques for large-scale feature selection. In *Machine Intelligence and Pattern Recognition*. Vol. 16. Elsevier, 403–413.

[18] Rudolph Flesch. 1948. A new readability yardstick. *Journal of applied psychology* 32, 3 (1948), 221.

[19] GitHub. 2020. The 2020 State of the Octoverse. https://octoverse.github.com/#overview.

[20] Robert Gunning et al. 1952. Technique of clear writing. (1952).

[21] Alexander Hars and Shaosong Ou. 2002. Working for Free? Motivations for Participating in Open-Source Projects. *Int. J. Electron. Commer.* 6, 3 (2002), 25–39. https://doi.org/10.1080/10864415.2002.11044241

[22] Melinda R Hess and Jeffrey D Kromrey. 2004. Robust confidence intervals for effect sizes: A comparative study of Cohen'sd and Cliff's delta under non-normality and heterogeneous variances. In *annual meeting of the American Educational Research Association*. 1–30.

[23] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[24] Pieter Hooimeijer and Westley Weimer. 2007. Modeling bug report quality. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer (Eds.). ACM, 34–43. https://doi.org/10.1145/1321631.1321639

[25] 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 *44th Hawaii International International Conference on Systems Science (HICSS-44 2011), Proceedings, 4-7 January 2011, Koloa, Kauai, HI, USA.* IEEE Computer Society, 1–10. https://doi.org/10.1109/HICSS.2011.264

[26] Tian Jiang, Lin Tan, and Sunghun Kim. 2013. Personalized defect prediction. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, Ewen Denney, Tevfik Bultan, and Andreas Zeller (Eds.). IEEE, 279–289. https://doi.org/10.1109/ASE.2013.6693087

[27] J Peter Kincaid, Robert P Fishburne Jr, Richard L Rogers, and Brad S Chissom. 1975. *Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel.* Technical Report. Naval Technical Training Command Millington TN Research Branch.

[28] George R Klare. 1974. Assessing readability. *Reading research quarterly* (1974), 62–102.

[29] Ron Kohavi. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes.* Morgan Kaufmann, 1137–1145. http://ijcai.org/Proceedings/95-2/Papers/016.pdf

[30] Rajiv Krishnamurthy, Varghese S. Jacob, Suresh Radhakrishnan, and Kutsal Dogan. 2016. Peripheral Developer Participation in Open Source Projects: An Empirical Analysis. *ACM Trans. Manag. Inf. Syst.* 6, 4 (2016), 14:1–14:31. https://doi.org/10.1145/2820618

[31] 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, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard (Eds.). IEEE / ACM, 187–197. https://doi.org/10.1109/ICSE.2017.25

[32] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 2008. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Trans. Software Eng.* 34, 4 (2008), 485–496. https://doi.org/10.1109/TSE.2008.35

[33] G Harry Mc Laughlin. 1969. SMOG grading-a new readability formula. *Journal of reading* 12, 8 (1969), 639–646.

[34] Andrew McCallum, Kamal Nigam, et al. 1998. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, Vol. 752. Citeseer, 41–48.

[35] Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5, 4 (1943), 115–133.

[36] Christopher J. Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret M. Burnett. 2018. Open source barriers to entry, revisited: a sociotechnical perspective. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 1004–1015. https://doi.org/10.1145/3180155.3180241

[37] Sebastiano Panichella. 2015. Supporting newcomers in software development projects. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 586–589. https://doi.org/10.1109/ICSM.2015.7332519

[38] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. 2003. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 616–623.

[39] RJ Senter and Edgar A Smith. 1967. *Automated readability index.* Technical Report. CINCINNATI UNIV OH.

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

[41] Bianca Shibuya and Tetsuo Tamai. 2009. Understanding the Process of Participating in Open Source Communities. In *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS '09)*. IEEE Computer Society, USA, 1–6. https://doi.org/10.1109/FLOSS.2009.5071352

[42] 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 2018, Madrid, Spain, September 23-29, 2018.* IEEE Computer Society, 172–182. https://doi.org/10.1109/ICSME.2018.00027

[43] Igor Steinmacher, Ana Paula Chaves, Tayana Uchôa Conte, and Marco Aurélio Gerosa. 2014. Preliminary Empirical Identification of Barriers Faced by Newcomers to Open Source Software Projects. In *2014 Brazilian Symposium on Software Engineering, Maceió, Brazil, September 28 - October 3, 2014.* IEEE Computer Society, 51–60. https://doi.org/10.1109/SBES.2014.9

[44] Igor Steinmacher, Marco Aurélio Graciotto Silva, Marco Aurélio Gerosa, and David F. Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Inf. Softw. Technol.* 59 (2015), 67–85. https://doi.org/10.1016/j.infsof.2014.11.001

[45] Igor Steinmacher, Christoph Treude, and Marco Aurélio Gerosa. 2019. Let Me In: Guidelines for the Successful Onboarding of Newcomers to Open Source Projects. *IEEE Softw.* 36, 4 (2019), 41–49. https://doi.org/10.1109/MS.2018.110162131

[46] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online Defect Prediction for Imbalanced Data. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*, Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). IEEE Computer Society, 99–108. https://doi.org/10.1109/ICSE.2015.139

[47] Xin Tan, Minghui Zhou, and Zeyu Sun. 2020. A first look at good first issues on GitHub. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 398–409. https://doi.org/10.1145/3368089.3409746

[48] Chakkrit Tantithamthavorn and Ahmed E. Hassan. 2018. An experience report on defect modelling in practice: pitfalls and challenges. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Frances Paulisch and Jan Bosch (Eds.). ACM, 286–295. https://doi.org/10.1145/3183519.3183547

[49] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, 321–332. https://doi.org/10.1145/2884781.2884857

[50] 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 quarterly* (2012), 649–676.

[51] 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, CHASE 2011, Waikiki, Honolulu, HI, USA, May 21, 2011*, Marcelo Cataldo, Cleidson R. B. de Souza, Yvonne Dittrich, Rashina Hoda, and Helen Sharp (Eds.). ACM, 76–79. https://doi.org/10.1145/1984642.1984661

[52] Song Wang, Chetan Bansal, Nachiappan Nagappan, and Adithya Abraham Philip. 2019. Leveraging change intents for characterizing and identifying large-review-effort changes. In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*. 46–55.

[53] Eric W Weisstein. 2004. Bonferroni correction. (2004).

[54] F. Wilcoxon. 1945. Individual comparison by ranking methods. *Biometrics* 1 (1945).

[55] Vincent Wolff-Marting, Christoph Hannebauer, and Volker Gruhn. 2013. Patterns for tearing down contribution barriers to FLOSS projects. In *IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques, SoMeT 2013, Budapest, Hungary, September 22-24, 2013*. IEEE, 9–14. https://doi.org/10.1109/SoMeT.2013.6645669

[56] Meng Yan, Xin Xia, David Lo, Ahmed E. Hassan, and Shanping Li. 2019. Characterizing and identifying reverted commits. *Empir. Softw. Eng.* 24, 4 (2019), 2171–2208. https://doi.org/10.1007/s10664-019-09688-8

[57] Yunwen Ye and Kouichi Kishida. 2003. Toward an Understanding of the Motivation of Open Source Software Developers. In *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, Lori A. Clarke, Laurie Dillon, and Walter F. Tichy (Eds.). IEEE Computer Society, 419–429. https://doi.org/10.1109/ICSE.2003.1201220

[58] Harry Zhang. 2004. The Optimality of Naive Bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA*, Valerie Barr and Zdravko Markov (Eds.). AAAI Press, 562–567. http://www.aaai.org/Library/FLAIRS/2004/flairs04-097.php

[59] Minghui Zhou and Audris Mockus. 2012. What make long term contributors: Willingness and opportunity in OSS community. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, Martin Glinz, Gail C. Murphy, and Mauro Pezzè (Eds.). IEEE Computer Society, 518–528. https://doi.org/10.1109/ICSE.2012.6227164

[60] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Trans. Software Eng.* 36, 5 (2010), 618–643. https://doi.org/10.1109/TSE.2010.63