An Empirical Study of Cross-Project Pull Request Recommendation in GitHub

Wenyu Xu, Yao Lu*, Xunhui Zhang, Tanghaoran Zhang, Bo Lin, Xinjun Mao
School of Computer Science and Technology
National University of Defense Technology
Changsha, China
{xuwenyu, luyao08, zhangxunhui, zhangthr, linbo19, xjmao}@nudt.edu.cn

***Abstract*—As a core contribution merge mechanism in distributed collaborative development, pull requests contain valuable knowledge of code evolution and issue resolution. With the co-evolution of multiple projects in a software ecosystem, relevant and similar issues can arise across different projects. Leveraging existing solutions in pull requests (PRs) through cross-project pull request recommendation (CPR) can enrich context knowledge and improve the efficiency of issue resolution. However, the characteristics of CPR and its effectiveness in the process of issue resolution still remain unclear. To bridge this gap, we conduct an empirical study of the CPR on GitHub. We first extract 4,445 CPRs from 2,500 open source projects and quantitatively analyze the characteristics of CPR. Then we conduct a qualitative analysis of sampled CPR cases to understand the influence of CPR. We also use a regression model to explore the impact of CPRs on issue resolution. Our main findings are as follows: (1) Experienced contributors in target projects make most of the CPRs and their CPRs are more timely than inexperienced contributors; (2) In CPR dataset, bugs constitute the largest proportion of target issue types, followed by enhancements, features and questions; (3) Nearly half of the CPRs are accepted by issue participants; (4) A greater number of the CPRs contribute indirectly to solving the target issue by offering solutions and contextual information, rather than providing appropriate code that can be directly applied to the issue; (5) Most of CPR-related factors have a significant impact on issue resolution delay. Among these, recommendation latency has the most significant impact, followed by the type of recommender. Our work has important insights into CPR and offers important guidance for developers on recommending cross-project PRs to resolve the mushrooming issues.**

***Index Terms*—pull-request recommendation, software maintenance, software repository mining**

I. Introduction

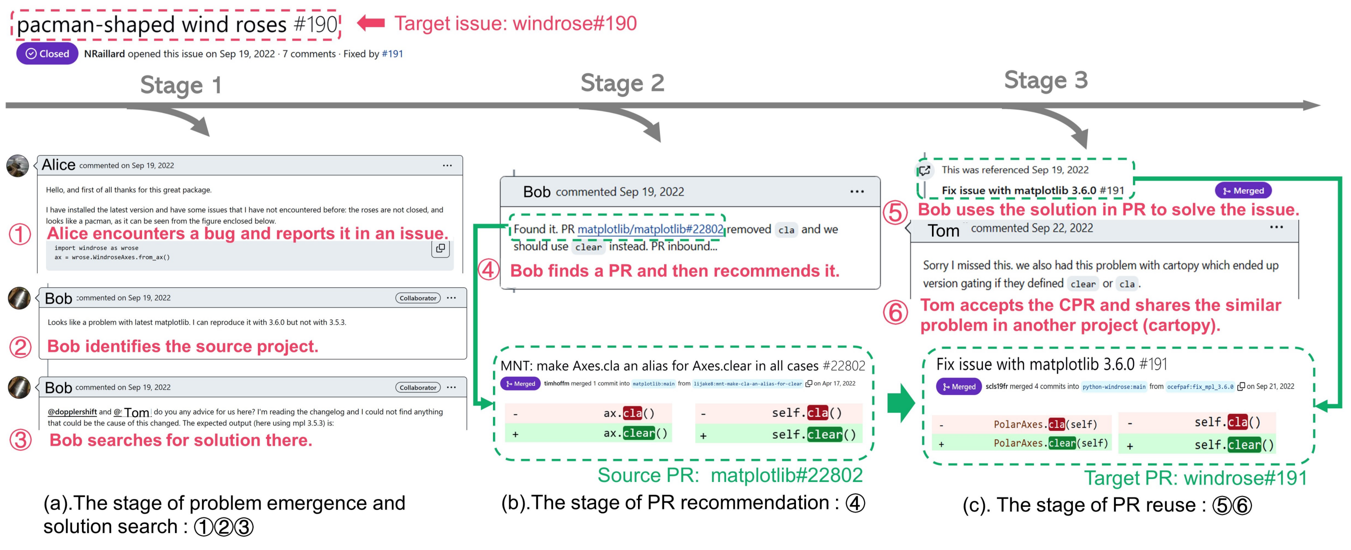
Pull requests (PRs) play a pivotal role in the collaborative development of open-source software (OSS) projects [4] by coupling code repository with issue-tracking [14], [15] system. Within these PRs lies a treasure trove of valuable contributions, such as meticulously-crafted code enhancements and crucial bug fixes. The previous work has focused on improving the PR mechanism to enhance the efficiency of distributed development [16], [18], [24], [27]–[29], [35]. With the co-evolution of multiple projects in a software ecosystem, relevant and similar issues can arise across different projects [2]. On the one hand, existing PRs may contain solutions to the current issue, and reusing them may improve the efficiency and quality of issue resolution. On the other hand, by learning from other teams' solutions to similar issues, developers may avoid repetitive tasks and improve the efficiency of issue resolution and the quality of project maintenance [25]. Therefore, developers are increasingly recommending PRs from other projects, i.e., cross-project pull request recommendation (CPR).

Figure 1(b) represents a concrete example of CPR. In this case, the code changes in the upstream project *matplotlib*[1] lead to the bug in downstream project *windrose*[2]. In the issue#190[3], Bob (For simplicity, we use Alice, Bob and Tom to refer to the different participants.) recommended in his comment the PR#22802[4] from *matplotlib*. Through CPR, Bob introduced solutions from *matplotlib* to *windrose*, attempting to reuse code changes in PR#22802 to fix the bug in *windrose*. If the team had not acquired the knowledge of code changes in CPR, they would have discussed and tested their project extensively to locate the source of the bug and might even give up fixing the bug, as shown in figure 1(c).

This example underscores the importance of cross-project PR reuse mechanisms [30] in software ecosystems, as developers often leverage such references to enhance code quality and development efficiency. Compared to other resources like Stack Overflow, which provides solutions through community-driven discussions and code snippets, existing PRs offer unique advantages. Stack Overflow posts are often isolated solutions without direct integration into the codebase, requiring additional effort to adapt and integrate them into software projects. In contrast, PRs can represent integrated and tested solutions within the actual software development, providing more reliable and appropriate solutions to certain issues. This resource may significantly reduce the effort required to adapt solutions and ensures compatibility with the existing codebase. Therefore, our study focuses on CPR as the main research object, aiming to analyze its characteristics, the role and the impact of CPR in the process of issue resolution.

* Yao Lu is the corresponding author.

[1]https://github.com/matplotlib/matplotlib
[2]https://github.com/python-windrose/windrose
[3]https://github.com/python-windrose/windrose/issues/190
[4]https://github.com/matplotlib/matplotlib/pull/22802


pacman-shaped wind roses #190
Target issue: windrose#190
Closed
Stage 1
Stage 2
Stage 3
① Alice encounters a bug and reports it in an issue.
② Bob identifies the source project.
③ Bob searches for solution there.
④ Bob finds a PR and then recommends it.
⑤ Bob uses the solution in PR to solve the issue.
⑥ Tom accepts the CPR and shares the similar problem in another project (cartopy).
Source PR: matplotlib#22802
Target PR: windrose#191
Merged
Collaborator

NRaillard opened this issue on Sep 19, 2022 · 7 comments · Fixed by #191

Alice commented on Sep 19, 2022

Hello, and first of all thanks for this great package.

I have installed the latest version and have some issues that I have not encountered before: the roses are not closed, and looks like a pacman, as it can be seen from the figure enclosed below.

```
import windrose as wrose
ax = wrose.WindroseAxes.from_ax()
```

Bob commented on Sep 19, 2022

Looks like a problem with latest matplotlib. I can reproduce it with 3.6.0 but not with 3.5.3.

Bob commented on Sep 19, 2022

@dopplershift and @Tom do you any advice for us here? I'm reading the changelog and I could not find anything that could be the cause of this changed. The expected output (here using mpl 3.5.3) is:

Bob commented Sep 19, 2022

Found it. PR matplotlib/matplotlib#22802 removed `cla` and we should use `clear` instead. PR inbound...

MNT: make Axes.cla an alias for Axes.clear in all cases #22802

timhoffm merged 1 commit into `matplotlib:main` from `lijake8:mnt-make-cla-an-alias-for-clear` on Apr 17, 2022

```
-    ax.cla()          -    self.cla()
+    ax.clear()        +    self.clear()
```

This was referenced Sep 19, 2022

Fix issue with matplotlib 3.6.0 #191

Tom commented Sep 22, 2022

Sorry I missed this. we also had this problem with cartopy which ended up version gating if they defined `clear` or `cla`.

Fix issue with matplotlib 3.6.0 #191

scls19fr merged 4 commits into `python-windrose:main` from `ocefpaf:fix_mpl_3.6.0` on Sep 21, 2022

```
-    PolarAxes.cla(self)      -    self.cla()
+    PolarAxes.clear(self)    +    self.clear()
```

(a).The stage of problem emergence and solution search : ①②③

(b).The stage of PR recommendation : ④

(c). The stage of PR reuse : ⑤⑥

Fig. 1. An Example of CPR

However, there remains a notable gap in the literature regarding the exploration of PR recommendation mechanisms, particularly in understanding how PRs from one project are recommended to other projects. Without such knowledge, developers and researchers may fail to fully leverage existing software resources within PRs, thereby resulting in redundant development efforts. Our research contributes to the field by shedding light on the characteristics and effectiveness of **c**ross-project **p**ull request **r**ecommendation (CPR).

With this focus, we will delineate the definitions of key terms integral to the practice of CPR. The term CPR refers to *the act of recommending a PR from an external project during the engagement with GitHub issues*. This PR is designated as source PR, while the project from which it originates is referred to as source project. Conversely, the issue being addressed is termed target issue, and the corresponding project is identified as the target project. Additionally, the developer responsible for making the CPR is called (CPR) recommender.

To gain a deeper understanding of CPR, we utilized the GitHub API and heuristic methods to collect CPR cases from open source projects, followed by a comprehensive data analysis of the dataset. Subsequently, to elucidate the positive roles CPR plays in issue resolution, we manually annotated and analyzed a representative sample of CPR cases. Finally, to investigate the impact of CPR-related factors on the issue resolution process, we developed a multiple linear regression model to examine the relationship between these factors and issue resolution delays. The highlights of our findings are as follows. *Characteristics of CPR:* (1) Experienced contributors contribute most of CPRs and their CPRs are more timely than inexperienced contributors. *Effectiveness of CPR:* (3) Nearly half of CPRs receive acceptance from issue participants; (4) A larger proportion of CPRs help solve the target issues indirectly by providing solutions and contextual information, rather than directly offering applicable code. *Impact of CPR on Issue Resolution:* (5) Prompt recommendations are associated with more rapid problem-solving, whereas overly complex recommended PRs can substantially prolong the time required to resolve issues.

To summarize, we make the following contributions:

- To the best of our knowledge, we construct the first CPR dataset[5], comprising 4,445 CPR cases. Each case in our dataset includes the recommender, recommendation time, and the metadata of both the PR and the related issue, which can facilitate further investigations on this topic.
- Our research reveals the significant contribution of experienced developers in making CPRs. It also demonstrates that CPR more often plays an indirect and supplementary role in problem-solving, rather than directly providing usable code.
- We examine the influence of various CPR-related factors on issue resolution. Our analysis reveals that most of these factors significantly affect resolution delays. Among them, recommendation latency has the greatest impact, followed by the type of recommender.
- This study shows that although developers encountering challenging issues may benefit from the PRs from other projects, efficient CPRs necessitate a comprehensive understanding of the target project.

II. MOTIVATION AND RESEARCH QUESTIONS

As a core contribution mechanism in distributed collaborative development, pull requests store valuable knowledge of code evolution and issue resolution, which can be shared and reused across different projects through CPR. Figure 1 illustrates the CPR process in GitHub. Figure 1 shows the process of CPR in GitHub. This case can be divided into three stages: *problem emergence and solution search*, *PR recommendation* and *PR reuse*.

[5]The full dataset is accessible at https://zenodo.org/records/12789494.

- *Problem emergence*: Firstly, Alice encountered a bug when the latest version of *windrose* was installed and thus she reported the bug in *windrose* to ask for help. After a series of tests, Bob found that simply reducing the version of the *matplotlib* library which *windrose* relies on could eliminate the bug. So he was certain that the bug originated from *matplotlib*, not *windrose* itself and that the bug can't be fixed until the *windrose* team understands the recent code changes in *matplotlib*. Therefore Bob had to search for the cause in the PRs of *matplotlib*, which stores the code changes in need.
- *PR recommendation*. In the PR *matplotlib* #22802, Bob finally found the cause, which was a change in the method name, from *cla* to *clear*. In order for the team to understand the cause, Bob recommended the PR to the target issue and briefly explained the code changes. Through recommending the PR, the necessary development knowledge in *matplotlib* expedites the issue resolution process in *windrose*.
- *PR reuse*. After fully understanding the cause of the bug, Bob created PR *windrose*#191 to fix the bug. The code changes in the fix are similar to those in *matplotlib*, but adapted to the context of the code in *windrose*. PR#191 passed the tests and successfully solved the bug in *windrose*. But the same bug had also appeared in another project *cartopy*, reported by another developer Tom. However, he missed the PR *matplotlib* #22802 and didn't come up with the right solution even if he had identified where the error occurred in *cartopy*. Without that PR, Tom could not solve the similar bug in *cartopy* and had to disable some code features through version gating, which seriously affects the functionality of the software and reduces the code quality.

We can see from the example that the CPR might expedite the process of issue resolution through supplementing key information from other projects. Guided by the motivations above, we realize that behind CPR phenomenon lies a new paradigm for promoting the reuse of code knowledge in the open source community. However, to the best of our knowledge, there is a lack of work on CPR. In order to gain a better understanding of CPR, we propose the following research questions.

RQ1: What are the characteristics of CPR? This question aims to investigate the current practices of CPR and obtain the intuitive understanding of this collaborative phenomenon. We proceed by deconstructing this question into the following sub-questions:

RQ1.1: Who recommend cross-project pull requests in the process of issue resolution?

RQ1.2: When does CPR occur during the issue resolution process?

RQ1.3: What types of issues are typically targeted for CPRs?

RQ2: How do developers response to CPRs in the process of issue resolution?

This research question attempts to obtain insights of CPR from the viewpoint of developer responses, including the distribution of developers' feedbacks on CPR and the roles of CPRs in the process of issue resolution.

RQ3: How do CPR-related factors affect the resolution of the target issues?

Investigating CPR-related factors can reveal key determinants influencing the efficacy of CPRs. Consequently, this research question seeks to examine the impact of CPR-related factors on the resolution of target issues.

III. Research Methodology

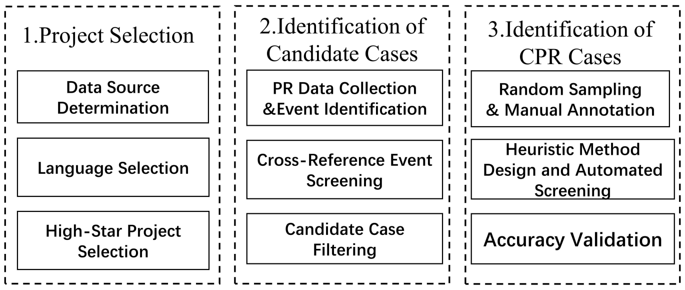

1.Project Selection
Data Source Determination
Language Selection
High-Star Project Selection
2.Identification of Candidate Cases
PR Data Collection &Event Identification
Cross-Reference Event Screening
Candidate Case Filtering
3.Identification of CPR Cases
Random Sampling & Manual Annotation
Heuristic Method Design and Automated Screening
Accuracy Validation

Fig. 2. The Data Preparation Process

In this section, we first outline the construction of the CPR dataset and then we will introduce the qualitative analysis method. Subsequently, we present the linear regression model to examine the impact of CPR on issue resolution. The process of data preparation is illustrated in Figure 2.

A. Data Collection

We introduce how we construct our dataset including the selection of source projects, the identification of candidate cases and the identification of CPR cases.

1) Project Selection: We initiate the project selection process by focusing on source projects to ensure that all the source PRs originate from specific projects. We select source projects based on the continuously updated dataset GHS [5], which contains over 735,000 GitHub repositories written in popular programming languages. To enhance the representativeness of our project collection, we initially concentrate on five programming languages with the largest number of GitHub repositories: JavaScript, Python, C/C++, Java and PHP. Previous research [6] indicates that most of GitHub projects are inactive or not intended for software development. Also, a prior survey [26] reveals that developers consider the number of stars before using or contributing to a GitHub project. Therefore, we select the top-500 most starred repositories for each of those languages as of March 2024.

2) Identification of Candidate Cases: In GitHub, a cross-reference event means an issue or PR is referenced from another issue or PR. So we choose to collect CPR cases by screening for such events. We primarily use the GitHub API to gather information and identify cross-reference events[6]. For

[6]https://docs.github.com/en/rest/using-the-rest-api/issue-event-types?apiVersion=2022-11-28#cross-referenced

a source project, we first use the GitHub API to get all its PRs and their related information (e.g., PR title, PR description). After that, for each PR, we use the cross-reference API to get all cross-reference events. Issue and PR have distinct URLs, allowing us to identify their types. Our algorithm recognize a cross-reference event as a candidate case only when the target type is issue, the source type is PR and the target project differs from the source project. Through this process, we found a total of 32,310 candidate CPR cases from 25,161 PRs in source projects. To confirm issue resolution, we retain only cases where the target issue is closed, resulting in 10,444 candidate CPR cases. Currently, only cases with cross-project PR links have been screened, and further screening is needed to identify genuine CPR cases.

3) Identification of CPR: To identify cases that recommend PRs, we employ a heuristic method based on textual keywords and syntactic patterns. We begin by randomly sampling 384 cases and meticulously annotating each to ascertain if it is a CPR case, resulting in 162 identified cases. The sample size of 384 is determined based on statistical requirements to ensure a 95% confidence level and a 5% margin of error. By systematically analyzing the comments of these identified cases, we identify common keywords and sentence patterns[7] developers utilize to recommend PRs. Using these keywords and sentence patterns, we conduct an automated screening of the entire dataset, identifying 4,555 cases as CPR cases. To verify the accuracy of this automated annotation, we randomly sampled an additional 384 cases and manually annotated them, confirming 363 as CPR cases. This validation process yields an accuracy rate of 95% for the automated annotation. Both rounds of manual annotations were independently completed by two authors, with each annotation round achieving a Cohen's Kappa coefficient of 90%. This identification methodology, integrating initial manual annotation, automated screening, and subsequent validation, ensures the reliability of our CPR dataset.

B. Qualitative Analysis

To investigate how developers respond to CPR and the role that CPR plays in issue resolution, we conducted a thorough examination of random CPR case samples, following the coding method of the prior research [4]. Initially, the first coder analyzed 100 CPR cases to identify developer responses and the specific roles of CPR through reading the comments of target issues, which serves as the boot-strapping sample. Subsequently, a different set of 100 cases was analyzed by all three coders to validate the identified categories, forming the cross-validation sample. After completing cross-validation, the two datasets were merged. Additionally, 200 randomly selected cases were incorporated into the bootstrapping sample. This process resulted in a final dataset of 400 CPR cases for qualitative analysis.

[7]The keywords and sentence patterns is also accessible at https://zenodo.org/records/12789494.

C. Regression Analysis

1) Regression Modeling: In order to discover the relationship between CPR and the timely closing of the pending issues, we propose a multiple linear regression model, *Resolution delay model*. In our models, we applied log transformations to continuous variables to stabilize their variance and reduce heteroscedasticity [3], [7].

The variance inflation factors, which indicate multicollinearity among our predictors, all remained below 3, ensuring their reliability, which follows the standard of prior work [3]. We use the adjusted R^2 statistic to evaluate goodness-of-fit of our model. For each model variable, we report its coefficient, stand error, significance level, and sum of squares (via *ANOVA* analysis). We consider coefficients to be statistically significant if they achieved a significance level of $p < 0.05$ [31].

2) Regression Variables: Because we focus on how long it takes for the target to be resolved after CPR, the dependent variable in our model is the issue resolution delay, denoted as *issueResolutionDelay* and measured in days. It represents the time interval between the CPR and the closure of target issue.

Our independent variables can be divided into two categories: CPR-related variables and other variables. CPR-related variables are as followd:

- ***recommendationLatency***: Time interval between the target issue creation and the recommendation of CPR, in days. When recommendation latency is longer, developers might already be occupied with other issues, or project priorities may have shifted, leading to delayed processing of issue resolution. This latency can increase the time required to resolve the issue. Additionally, later-recommended PRs may not receive the same level of attention or prompt response as earlier ones, further reducing the efficiency of PR reuse. Therefore, we assume that if recommendation latency is longer, the issue resolution delay might be longer.
- ***uMergedPR***: Binary, True if the source PR is merged to the source projects, as a proxy for the quality of the source PR. If the source PR is eventually merged into the source project, the issue resolution delay might be shorter. PRs that are merged into the source project typically undergo rigorous review and testing, suggesting higher quality. Consequently, these PRs are more likely to offer robust solutions, which can expedite the adoption and resolution of the target issue.
- ***nCommentPR***: Total number of the comments (not the reveiw comments) in the source PR, as a proxy for the information in the discussion of source PR. More comments in source PR may correlate with shorter issue resolution delay. Increased comments suggest active engagement and diverse perspectives, enhancing clarity and addressing potential issues, which facilitates quicker resolutions once the PR is recommended.
- ***nAddedLinePR***: Number of the code lines added in the source PR, as a fine proxy for the scope of code changes in PR. More added code lines may potentially prolong the issue resolution delay, as larger code changes often necessitate more extensive review and testing before implementing the

solution to address the target issue.

- ***textLenPR***: Total number of words in title and description text within the source PR. A longer title and description may contribute to a shorter issue resolution delay, as they typically indicate clearer documentation. Consequently, this enhanced clarity can facilitate a better understanding of the proposed solution.
- ***uContributor***: Binary, True if the recommender has contributed code to the target project before the CPR is made. It serves to assess the recommender's previous interactions within the context of the target project. When a recommender has prior contributions, the issue resolution delay may be shorter, as established contributors typically possess a deeper understanding of the project's context. This expertise allows them to identify and recommend the most suitable PRs, resulting in more efficient issue resolution.

Other variables are as followed:

- ***uHasLabelTrue***: Binary, True if the target issue is labeled in the target project, serving as a proxy for the development team's focus. When an issue is labeled, the resolution delay may be shorter, as labeling signifies prioritized attention from the development team. Labeled issues typically receive more focused scrutiny and resources, which can expedite the problem-solving process. Furthermore, this prioritization enhances visibility and encourages timely responses, thereby reducing resolution delays.
- ***nParticipantIssue***: Number of developers that participated in the resolution of the target issue, as a proxy for the developer involvement. A higher number of participants may lead to shorter issue resolution delays, as increased involvement typically fosters collaborative problem-solving. Moreover, greater developer participation brings diverse perspectives and expertise, which facilitates faster identification of solutions in PR. These collaborative efforts streamline both the review and implementation processes, ultimately reducing the time required to resolve the issue.

IV. Research Results

This section introduces the results and findings for each research question.

A. RQ1: Characteristics of CPR

In the next few paragraphs, we will first discuss the role of the CPR recommender (RQ1.1) in the target project, followed by an analysis of the distribution of recommendation latency (RQ1.2). Finally, we will analyze the types of target issues in CPR cases (RQ1.3).

1) RQ1.1 Who recommend cross-project pull requests in the process of issue resolution?: In this sub RQ, we analyze characteristics of CPR from the perspective of the recommenders' role in target projects. Namely, we classify recommenders according to their relationship with the target projects, referred to as author associations, which are detailed in Table I. As mentioned in previous work [32], GitHub defines five author associations: OWNER, MEMBER, CONTRIBUTOR, COLLABORATOR and NONE. We categorize recommenders into two groups based on their author associations with target projects when they make CPRs. First, we classify recommenders who are OWNER, MEMBER, CONTRIBUTOR and COLLABORATOR as experienced contributors because they are either responsible for managing the target project or have contributed code to the target project and are more likely to be familiar the context of the target project. Next, those classified under the NONE association are identified as inexperienced contributors, since their contributions to the target project are limited to creating issues and comments, they are more likely to be unfamiliar with the target project.

TABLE I
AUTHOR ASSOCIATION AND THE DESCRIPTION ON GITHUB

#Author_association	Description
OWNER	The owner of the project
COLLABORATOR	Author who has been invited to collaborate on the project
MEMBER	The member of the organization that owns the project
CONTRIBUTOR	Author who has previously committed to the project
NONE	Author who has no association with the project

TABLE II
THE ROLE OF RECOMMENDERS IN CPR

Role	Proportion
Experienced contributor	74.36%
Inexperienced contributor	25.64%

As shown in Table II, most CPRs are made by experienced contributor (74.36%), highlighting their significant role due to their extensive project experience, technical competence, and deeper understanding of the project's requirements and goals. In contrast, inexperienced contributors recommend fewer CPRs (25.64%) because inexperienced contributors are not familiar with the project context, making it difficult for them to find PRs from other projects that can solve the target issues.

Finding 1: Experienced contributors make more CPRs than inexperienced contributors, accounting for 74.36%

2) RQ1.2: What is the distribution of recommendation latency in CPRs?: With a clear understanding of the recommender's association, our focus shifts towards examining the recommendation latency. In this study, recommendation latency is defined as the interval between issue creation to CPR in days. Since the recommending comments contain links, we refer to the definition of link Latency in previous work [3] when defining recommendation latency.

Figure 3 delineates the histogram of recommendation latency, revealing a long-tail distribution. The latency ranges from several minutes to over 9 years. The median latency for

TABLE III
PRIMARY TYPES OF TARGET ISSUES IN CPR CASES

Type	**Description**	**#Cases**	**Proportion**
Bug	This issue reports an unexpected behavior or error in the software.	852	34.68%
Enhancement	This issue suggests an improvement or addition to the current features of the software.	304	12.37%
Feature	This issue proposes the introduction of a new feature to the software.	167	6.80%
Question	This issue seeks clarification or further information about a particular aspect of the software.	155	6.31%
Help wanted	This issue indicates that assistance is needed to complete a task or resolve a problem.	54	2.20%
Others		925	37.65%
total		2457	100.00%

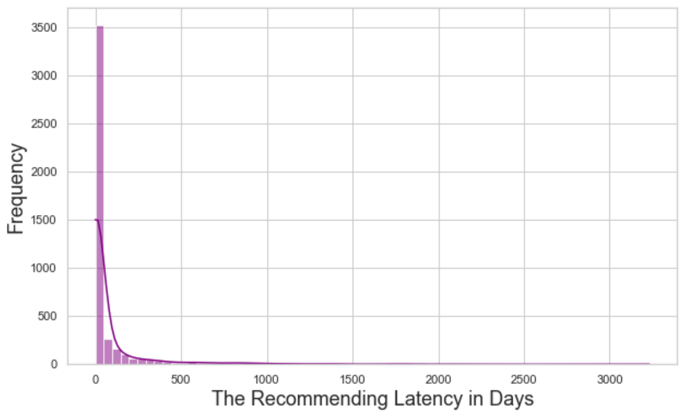


Fig. 3. Distribution of recommendation latency for all CPR cases.

TABLE IV
STATISTICS OF RECOMMENDATION LATENCY IN DAYS

Recommender	25%	50%	75%	Mean
Experienced contributor	0.3	3.6	27.9	70.8
Inexperienced contributor	0.8	7.8	83.6	127.2

the entire dataset is 4.3 days, with a mean latency of 85.3 days. The 25th percentile is recorded at 0.4 days, whereas the 75th percentile stands at 38.8 days. Most CPRs occur within a brief time frame, indicating that the majority of PRs are recommended promptly. However, a small subset of cases experiences longer delays. Such extended latencies may be attributable to the prioritization and complexity of the issues, technical difficulties, or constraints in resources.

Then we further analyzed the recommendation latency across different types of recommenders. As shown in Table IV, the CPRs made by experienced contributors exhibit the lowest 25th percentile, median, and 75th percentile. The results of the WMW tests show that the recommendation latency of experienced contributors is significantly lower than that of inexperienced contributors ($p<7.5e-15$) with a small Cliff's δ: -0.15. These results suggest that experienced contributors tend to make CPRs much more quickly than inexperienced contributors, with shorter latencies at all percentiles and a lower average latency. This likely reflects the benefits of their deeper project knowledge and technical competence.

Finding 2: While a small subset of CPRs face significant delays, most are prompt. The recommendation latency of experienced contributors is apparently shorter than that of inexperienced contributors.

3) RQ1.3: What types of issues are typically targeted for CPRs?: We first analyze the tag list of each target issue and identify that a total of 2,457 issues are tagged. Among these, the five most frequently appearing tags are *bug*, *enhancement*, *question*, *feature*, and *help wanted*. Using the text-based classification from previous work [33], we then screen the target issues based on these tags. Notably, if a target issue contains multiple tags, it is classified under more than one category.

The analysis of target issues in CPRs, as shown in Table III, indicates that *bugs* are the predominant category, comprising 34.68% of the total 2,457 cases. Following this, *enhancements* make up 12.37%, while features account for 6.80%. Additionally, *questions* constitute 6.31%, and *help wanted* represent the smallest proportion at 2.20%. This distribution clearly highlights that *bugs* are the most prevalent type of issues addressed in CPR cases.

Finding 3: In CPR dataset, *bugs* constitute the largest proportion of target issue types at 34.68%, followed by *enhancements* at 12.37%, *features* at 6.80%, *questions* at 6.31%, and *help wanted* at 2.20%.

B. RQ2: Developer Responses to CPR

To understand how developers responded to CPR, we conduct further manual reviews of CPR cases using a sampled dataset, which contains 400 CPR cases. During manual labeling, 20 cases are identified as non-recommended cases and removed, leaving 380 CPR cases. These cases are categorized into **acceptance** and **non-acceptance** based on predefined criteria established during an initial review. Consistent classification rules are applied by two authors to label CPRs as accepted or not. In instances of disagreement, a third author provided the final classification. The classification rules and examples are outlined below:

Accepted CPR. A CPR case is *accepted* if any of the following three conditions occur:

- **Condition 1:** After the CPR is made, a developer contributes a commit or PR for the target issue to solve the issue (checked by the cross-references), where the solution in CPR is mentioned.

TABLE V
THE TAXONOMY OF DEVELOPER RESPONSE IN CPR CASES

Category	Count (%)
Accepted CPRs	167 (43.95%)
Not accepted CPRs	213 (56.05%)
Total	380 (100.00%)

TABLE VI
EFFECT OF PRS IN CPR FOR ISSUE RESOLUTION

Effect	Description	%
Entire Transplantation	The PR can be entirely transplanted into the target project.	8.4%
Solution Provision	The PR provides a solution that can be referenced to resolve the target issue.	27.5%
Context Enhancement	The PR enriches the contextual knowledge of the target issue.	44.3%
Direct Resolution	The PR can directly resolve the target issue.	19.8%

- **Condition 2:** After the CPR is made, subsequent comments of issue participants indicate that they have acquired knowledge (code changes or textual information) from the CPR.
- **Condition 3:** After the CPR is made, subsequent comments from developers indicate that they try to migrate the source PR to the target project.

Not accepted CPR. A CPR case is *not accepted* if none of the above three conditions occurs.

As shown in Table V, among sampled CPRs, 43.95% are accepted, indicating that nearly half of the CPRs may be reused. Not accepted CPRs make up 56.05%, among which only 16 cases are explicitly rejected by the developer and the others receive no response from the developers.

To better understand how CPR contributes to resolving target problems, we conducted a manual review of 167 accepted CPR cases. In this analysis, we manually assessed the role of CPR in issue resolution. We measured the inter-coder agreement, obtaining a Cohen's kappa coefficient of 71%, which indicates substantial agreement [23]. The results of this analysis are presented in the following Table VI:

- **Entire Transplantation**. In this situation, the PR can be entirely transplanted into the target project. In issue #24827[8], for example, after a developer recommended the PR#11442[9], another developer stated that only a copy/paste of code changes in the PR#11442 were needed to solve the target issue and did so in a subsequent PR.
- **Solution Provision**. In this situation, PR contains actionable solutions that can serve as practical guides to resolve the issue. For example, in this CPR case [10], after a developer recommend a PR, the issue creator shows that the required functionality can be achieved with the solution mentioned in that PR.
- **Context Enhancement**. In this condition, PRs furnish supplementary context or background information, thereby augmenting the developers' understanding of the issue and expediting its resolution. For example, in issue#135[11], *macchrome* recommended a related PR[12]. With the help of the context in PR, *graphixillusion* stated that he understood the problem better.
- **Direct Resolution**. In this condition, the underlying cause of the target issue originates in other projects, and the PR directly addresses this root cause. Consequently, the developers are not required to redundantly resolve the issue within the target project. In issue#179[13], for example, *pkuliyi2015* commented that the target issue stemmed from a bug in the upstream project and the PR *gradio#4073*[14] had already fixed it. Therefore there was no need to solve the issues repeatedly.

We then categorize CPRs based on whether the code of PRs can be directly utilized to address the target problem. Specifically, CPRs cases of Entire Transplantation and Direct Resolution are classified as direct contribution, whereas cases of Context Enhancement and Solution Provision are considered to offer indirect assistance. Our analysis reveals that 28.2% of CPRs directly contributed to solving the problem, while 61.8% provided indirect assistance.

Finding 4: In the CPR dataset, nearly half of the CPRs are accepted by developers. Notably, a greater number of the CPRs contribute indirectly to solving the target problem by offering solutions and contextual information, rather than providing code that can be directly applied to the issue.

C. RQ3: Impact of CPR-related Factors

We proposed the *Resolution delay model* in order to investigate the relationship between issue resolution delay and CPR-related factors. Table VII shows the modeling result, where the variables above the dotted line are CPR-related. It shows that this model achieves a fit of $R^2 = 17.40\%$.

Regarding CPR-related factors, we find that *recommendationLatency* exerts a positive influence on *issueResolutionDelay*, implying that longer recommendation latency of CPR correlates with longer resolution delay of the target issue. Conversely, *uContributor* exerts a negative influence, demonstrating that if the recommender has previously contributed code to the target project, the issue resolution delay is shorter. This suggests that recommenders who are well-versed in the context of the target project can make more effective CPRs, thereby reducing issue resolution delays. *nCommentPR* exhibits a positive influence, indicating that a higher number of comments on the PR is associated with a longer delay to

[8]https://github.com/tensorflow/tensorflow/issues/24827#issuecomment-546656874
[9]https://github.com/keras-team/keras/pull/11442
[10]https://github.com/bitsandbytes-foundation/bitsandbytes/issues/40#issuecomment-1414293040
[11]https://github.com/macchrome/winchrome/issues/135
[12]https://github.com/ungoogled-software/ungoogled-chromium/pull/2681
[13]https://github.com/pkuliyi2015/multidiffusion-upscaler-for-automatic1111/issues/179#issuecomment-1538275362
[14]https://github.com/gradio-app/gradio/pull/4073

TABLE VII
RESOLUTION DELAY MODEL. THE RESPONSE IS LN(ISSUERESOLUTIONDELAY+0.5). ADJUSTED $R^2 = 17.40\%$.

Independent Variable	Coeffs (Error)	Sum Sq
Intercept	2.7786 (0.0959)***	
ln(recommendationLatency+0.5)	0.2599 (0.0365)***	1215***
uContributorTrue	-0.3301 (0.0836)***	317***
ln(nCommentPR+0.5)	0.1405(0.0378)***	251***
uMergedPRTrue	-0.2951 (0.0761)***	140***
ln(nAddedLinePR+0.5)	0.1617 (0.0369)***	89***
ln(textLenPR+0.5)	-0.1020 (0.0359)***	36*
ln(nParticipantIssue+0.5)	0.7229 (0.0380)***	1783***
uHasLabelTrue	0.4815 (0.0712)***	530***

$^{***}p < 0.001$,$^{**}p < 0.01$,$^{*}p < 0.05$

close the issue. Additionally, *uMergedPR* shows a negative influence, suggesting that recommending merged PRs results in a shorter delay in issue closure compared to unmerged PRs. Furthermore, *nAddedLinePR* has a positive effect. The more new lines of the code added, the more time is required for review and testing, thereby increasing issue resolution delay. Conversely, *textLenPR* demonstrates a negative effect, indicating that recommending more documented PRs leads to a shorter issue resolution delay.

In terms of other factors, *nParticipantIssue* exerts a positive influence, likely because a higher number of participants necessitates more time for communication and coordination. Lastly, *uHasLabel* has a positive effect, indicating that labeled issues take longer to resolve compared to unlabeled ones.

From the perspective of Sum Sq, *nParticipantIssue* and *recommendationLatency* are the most explanatory variables for issue resolution delay. This means that more participants and delayed recommendations have the most significant impact on the resolution delay. Other factors, such as the contributor's experience (represented by *uContributorTrue*), the number of comments in PR, and whether the PR is merged , also show significant impacts but to a lesser extent. Variables like detailed documentation in PR and added lines of code have relatively weaker but still significant effects. These findings indicate that in the CPRs, the project background of the recommender, the complexity of the PR, and the number of participants are key factors affecting the delays of issue resolution.

Finding 5: Most of CPR-related factors have a significant impact on issue resolution delay. Among these, recommendation latency has the most significant impact, followed by the type of recommender and the number of comments in PR.

V. IMPLICATIONS

We provide insights from our findings for researchers, developers, and tool builders.

A. For Researchers

Our findings indicate that nearly half of CPRs are accepted. This underscores the critical role of CPR for practical problem-solving. Although the vast majority of CPRs are made in a relatively short timeframe, a small proportion of cross-project PRs are recommended to the target problem after a considerable period. This observation underscores the necessity to investigate the factors affecting CPR efficiency. Moreover, our qualitative analysis reveals that CPRs more often contribute indirectly to solving the target problem rather than by directly providing applicable code. Extracting contextual knowledge and solutions from existing PRs thus represents a valuable research question. Additionally, recommending appropriate PRs based on user needs is also of substantial importance. Finally, our regression model results showed that CPR-related factors are associated with the issue resolution delay, which indicates the timeliness of issue closure after CPRs are made. Whether these correlations are causal and how issue resolution is affected by CPR-related factors should be further empirically evaluated, as improving issue resolution efficiency can yield immediate practical benefits and enhance long-term project sustainability [34].

B. For Developers

Our case study illustrates that CPRs can provide critical contextual knowledge and referential solutions for issue resolution, thereby offering significant benefits to developers. We observed that recommending cross-project PRs at an earlier stage can markedly reduce the latency of issue resolution. This reduction is likely due to the diminished attention and delayed response that later CPRs receive. Consequently, it is helpful for developers encountering challenging problems or issues related to other projects to expediently identify and utilize relevant PRs out of their own projects. This strategy may enable them to find reusable PRs in a timely manner. Furthermore, we identified that developers with development experience in target project, whether in code contribution or project management, execute more CPRs and do so more promptly than those lacking target project development experience. This finding suggests that efficient CPRs often necessitate a comprehensive understanding of the target project, encompassing its code modifications and related projects.

C. For Tool Builders

Our study found that CPRs made by experienced contributors are associated with shorter issue resolution delays compared to those by inexperienced contributors. This finding presents opportunities for tool builders to design recommendation systems that assist developers in identifying relevant PRs for target issues, thereby enhancing collaboration and PR reuse within the OSS community [30]. Additionally, the most common types of target issues are bugs, enhancements, questions, features, and help wanted. Recommendation tools can employ different strategies based on the type of target issue to increase the efficiency of CPR and expedite the resolution process for specific issues.

VI. THREATS TO VALIDITY

We discuss our limitations in terms of internal and external threats to validity.

A. Internal Validity

In GitHub, PR can be recommended in the form of attached links to various occasions, including issue, other PR, commit message, etc. In this article, however, we only focus on the occasion of recommending PR to an issue. This is both because GitHub issues are the most common place for developers to report, discuss, and resolve development problems, and because we believe that getting as much knowledge as possible early in the discussion phase (rather than resolving phase) will most likely help the development team. Although the adjusted R^2 of the multiple linear regression model is 17.40%, which is not a perfect fitting degree, what we built is an explanatory model rather than a prediction model, and we just want to study the effect of CPR-related factors and don't explain all the factors influencing the issue resolution.

B. External Validity

We must acknowledge that CPR is a broad open source practice, and a PR in any project is likely to be recommended to other projects. However, given the general situation of open source projects and the tendency of developers to use other projects, our source project collection only considers projects with a high number of stars, which may cause data bias.

VII. RELATED WORK

We will mainly introduce the relevant work from two aspects, pull-based collaborative mechanism and issue resolution on GitHub.

A. Pull-based Collaborative Mechanism

The pull-based model [4], [41] on modern collaborative coding platforms (e.g., GitHub and GitLab) supports an efficient collaboration process [13], by coupling code repository with issue tracking [14], [15]. In order to improve the efficiency of pull-request-based development, a lot of work has been made [16], [18], [28], [29], [35]. Li *et al.* [16] examines the impact and features of duplicate pull requests, which result in a significant waste of human resources. And they proposed an approach [17] combining textual and change similarities to automatically detect duplicate contributions in the pull-based model at submission time. Except pull request duplication, pull request abandonment [18] also caught the attention of researchers. They carried out empirical work to study the impacts of , reasons for and solutions to pull request abandonment. Li *et al.* [20] foucused on the repeatedly revised PRs and identified 15 code review practices for requesting changes to PRs. What is more, the impact of Pull request template [19] and the referencing patterns in pull request [36] have been reported. In addition to exploring the PR process, many tools and methods have been proposed to improve the efficiency of PR submission [37] and review [38]. Although there have been in-depth studies on pull-based collaborative mechanism, there is a lack of empirical studies on cross-project pull request recommendations, which is the focus of this paper.

B. Issue Resolution on GitHub

Issue resolution process is usually considered a challenging, persistent, and central process in the development of OSS project which involves various communities and thus poses particular coordination problems [8]. The primary understanding of bug fixing primarily stems from analyses of several individual projects, focusing on aiding developers in efficiently addressing within-project bug reports [9]. However, these understanding may not be sufficient as the open source community evolves and grows because bug fixing can be propagated from one project to another due to code reuse or information sharing about related bugs in different projects [1]. Boisselle *et al.* [10] found that developers lose a median of 47 days of potential collaboration and users lose 38 days waiting for fixes already made in other contributions. Zampetti *et al.* [11] investigated how developers document pull requests in GitHub with external references (Q&A forums, code examples, etc.). They found that even though external resources can be useful to learn something new or to solve specific problems, they are still rarely referred. Jiang *et al.* [39] extracted concise bug-fixing patches from human-written patches in commits. Furthermore, in a recent paper, Ren *et al.* [12] have proposed issue recommendation model to help developers effectively address cross-project correlated issues (CPCIs), i.e., find related issues for CPCIs. While considerable researches have focused on automatic issue resolution and issue duplication, there remains a significant gap in understanding how PRs can be used to address similar issues across different projects. This study addresses this gap by examining how CPRs can facilitate issue resolution, thereby extending the applicability of PRs beyond individual projects.

VIII. CONCLUSION

To deeply understand characteristics and effectiveness of CPR, we conduct an empirical study in this paper. We find that experienced contributors recommend most of the CPRs, and their CPRs are more prompt than those from inexperienced contributors. Moreover, rather than directly providing code, most CPRs help solve target issues indirectly by offering solutions and contextual information. We also examine how CPR-related factors influence issue resolution. Our regression analysis reveals that prompt recommendations are associated with quicker problem-solving, while overly complex recommended PRs can extend the resolution time.

In the future, we intend to develop an automated CPR tool drawing from our findings. This tool will be designed for integration into GitHub, aligning seamlessly with developers' workflow during PR reuse. And its practical value will be assessed through developers' application in real-world scenarios.

Acknowledgement. This work was supported by National Natural Science Foundation of China (No.62172426).

REFERENCES

[1] M. Gharehyazie, B. Ray, and V. Filkov, "Some from here, some from there: Cross-project code reuse in github," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 291–301.

[2] W. Ma, L. Chen, X. Zhang, Y. Zhou, and B. Xu, "How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 381–392.

[3] Y. Zhang, Y. Yu, H. Wang, B. Vasilescu, and V. Filkov, "Within-ecosystem issue linking: a large-scale study of rails," in *Proceedings of the 7th international workshop on software mining*, 2018, pp. 12–19.

[4] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 345–355.

[5] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*. IEEE, 2021, pp. 560–564.

[6] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, pp. 2035–2071, 2016.

[7] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[8] K. Crowston and B. Scozzi, "Bug fixing practices within free/libre open source software development teams," *Journal of Database Management (JDM)*, vol. 19, no. 2, pp. 1–30, 2008.

[9] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 111–120.

[10] V. Boisselle and B. Adams, "The impact of cross-distribution bug duplicates, empirical study on debian and ubuntu," in *2015 IEEE 15th International working conference on source code analysis and manipulation (SCAM)*. IEEE, 2015, pp. 131–140.

[11] F. Zampetti, L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, and M. Lanza, "How developers document pull requests with external references," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 23–33.

[12] H. Ren, M. Ma, X. Zhang, Y. Cao, and C. Nie, "Effective recommendation of cross-project correlated issues based on issue metrics," in *Proceedings of the 14th Asia-Pacific Symposium on Internetware*, 2023, pp. 1–1.

[13] J. Zhu, M. Zhou, and A. Mockus, "Effectiveness of code contribution: From patch-based to pull-request-based tools," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 871–882.

[14] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE access*, vol. 5, pp. 3909–3943, 2017.

[15] Y. Zhang, B. Vasilescu, H. Wang, and V. Filkov, "One size does not fit all: an empirical study of containerized continuous deployment workflows," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 295–306.

[16] Z. Li, Y. Yu, M. Zhou, T. Wang, G. Yin, L. Lan, and H. Wang, "Redundancy, context, and preference: An empirical study of duplicate pull requests in oss projects," *IEEE Transactions on Software Engineering*, vol. 48, no. 4, pp. 1309–1335, 2020.

[17] Z.-X. Li, Y. Yu, T. Wang, G. Yin, X.-J. Mao, and H.-M. Wang, "Detecting duplicate contributions in pull-based model combining textual and change similarities," *Journal of Computer Science and Technology*, vol. 36, pp. 191–206, 2021.

[18] Z. Li, Y. Yu, T. Wang, G. Yin, S. Li, and H. Wang, "Are you still working on this? an empirical study on pull request abandonment," *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 2173–2188, 2021.

[19] Z. Li, Y. Yu, T. Wang, Y. Lei, Y. Wang, and H. Wang, "To follow or not to follow: Understanding issue/pull-request templates on github," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2530–2544, 2022.

[20] Z. Li, Y. Yu, T. Wang, S. Li, and H. Wang, "Opportunities and challenges in repeated revisions to pull-requests: An empirical study," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, pp. 1–35, 2022.

[21] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in oss community," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 518–528.

[22] X. Tan, M. Zhou, and Z. Sun, "A first look at good first issues on github," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 398–409.

[23] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[24] L. Li, Z. Ren, X. Li, W. Zou, and H. Jiang, "How are issue units linked? empirical study on the linking behavior in github," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 386–395.

[25] H. Hata, R. G. Kula, T. Ishio, and C. Treude, "Same file, different changes: the potential of meta-maintenance on github," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 773–784.

[26] H. Borges and M. T. Valente, "What's in a github star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, vol. 146, pp. 112–129, 2018.

[27] H. Jiang, Y. Li, S. Guo, X. Li, T. Zhang, H. Li, and R. Chen, "Duphunter: detecting duplicate pull requests in fork-based development," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2920–2940, 2023.

[28] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwonka, "Code reviewing in the trenches: Challenges and best practices," *IEEE Software*, vol. 35, no. 4, pp. 34–42, 2017.

[29] J. Liu, A. Deng, Q. Xie, and G. Yue, "A code reviewer recommendation approach based on attentive neighbor embedding propagation," *Electronics*, vol. 12, no. 9, p. 2113, 2023.

[30] G. Melo, T. Oliveira, P. Alencar, and D. Cowan, "Knowledge reuse in software projects: Retrieving software development q&a posts based on project task similarity," *Plos one*, vol. 15, no. 12, p. e0243852, 2020.

[31] Z. Zhang, X. Mao, Y. Lu, S. Wang, and J. Lu, "An empirical study on the influence of social interactions for the acceptance of answers in stack overflow," in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2020, pp. 425–434.

[32] H. Huang, Y. Lu, and X. Mao, "Gathering github oss requirements from q&a community: an empirical study," in *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2020, pp. 145–155.

[33] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?" in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 121–130.

[34] W. Xiao, H. He, W. Xu, Y. Zhang, and M. Zhou, "How early participation determines long-term sustained activity in github projects?" in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 29–41.

[35] S. Khatoonabadi, D. E. Costa, R. Abdalkareem, and E. Shihab, "On wasted contributions: understanding the dynamics of contributor-abandoned pull requests–a mixed-methods study of 10 large open-source projects," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, pp. 1–39, 2023.

[36] A. Chopra, M. Mo, S. Dodson, I. Beschastnikh, S. S. Fels, and D. Yoon, ""@ alex, this fixes# 9": Analysis of referencing patterns in pull request discussions," *Proceedings of the ACM on human-computer interaction*, vol. 5, no. CSCW2, pp. 1–25, 2021.

[37] I. C. Irsan, T. Zhang, F. Thung, D. Lo, and L. Jiang, "Autoprtitle: A tool for automatic pull request title generation," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 454–458.

[38] X. Ye, Y. Zheng, W. Aljedaani, and M. W. Mkaouer, "Recommending pull request reviewers based on code changes," *Soft Computing*, vol. 25, pp. 5619–5632, 2021.

[39] Y. Jiang, H. Liu, N. Niu, L. Zhang, and Y. Hu, "Extracting concise bug-fixing patches from human-written patches in version control systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 686–698.

[40] T. Xiao, S. Baltes, H. Hata, C. Treude, R. G. Kula, T. Ishio, and K. Matsumoto, "18 million links in commit messages: purpose, evolution, and decay," *Empirical Software Engineering*, vol. 28, no. 4, p. 91, 2023.

[41] X. Zhang, Y. Yu, G. Gousios, and A. Rastogi, "Pull request decisions explained: An empirical overview," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 849–871, 2022.