To Follow or Not to Follow: Understanding *Issue/Pull-Request Templates* on GitHub

Zhixing Li, Yue Yu*, Tao Wang, Yan Lei, Ying Wang, and Huaimin Wang

Abstract—For most Open Source Software (OSS) projects, issues and Pull-requests (PR) are the primary means by which stakeholders of a project report and discuss software problems and code changes, and their descriptions are important for people to understand them. To help ensure the informational quality of issue/PR descriptions, GitHub introduced the *issue/PR template* feature, which pre-populates the description for anyone trying to open a new issue/PR. To better understand this feature, we report on a large-scale, mixed-methods empirical study of templates that explores contents, impacts, and perceptions. Our results show that templates typically contain elements to greet contributors, explain project guidelines, and collect relevant information. After template adoption, the monthly volume of incoming issues and PRs decreases, and issues have fewer monthly discussion comments and longer resolution duration. Although both contributors and maintainers positively rated the usefulness of templates from various aspects, they also reported challenges in using templates (*e.g.*, excessive and irrelevant information request) and suggested potential improvements of the template feature (*e.g.*, better user interaction and advanced automation). This work contributes to the informed use and targeted improvement of templates to enhance OSS practitioners' collaboration and interaction.

Index Terms—Issue Template, Pull-Request Template, GitHub, Open Source Software

1 Introduction

Open source software (OSS) projects are generally developed and maintained in a distributed collaborative manner [43], [46], [55]. The survival and sustainability of many OSS projects relies on community contributors to submit issues [10], [11] and Pull-requests (PRs) [28], [65]. Typically, issues are used for reporting unexpected software behaviors, and PRs are used for committing code changes. With OSS participants globally dispersed, effective communication is essential for project maintainers and contributors to collaborate on the submitted issues/PRs [59], [63].

In most OSS projects, both issue discussion and PR evaluation is conducted primarily through text-based communication [31]. Contributors commonly provide a textual description of issues/PRs and their act of submission initiates the communication. Maintainers read issue/PR descriptions and join the communication by posting comments on issues/PRs. After back-and-forth comments between maintainers and contributors, maintainers ultimately make a decision on issues/PRs [38], [63]. As the starting point of communication, issue/PR descriptions can significantly affect OSS collaboration efficiency. For example, well-described issues/PRs would reduce project maintainers' cognitive load to gain a quick and correct understanding, thus reducing the cost of communication and facilitating the collaboration between maintainers and contributors.

However, data entry in issue/PR submission can be difficult and issue/PR descriptions widely differ in their quality [11], [59]. In many cases, issue/PR descriptions provide inadequate or inaccurate information [11], [22], which could happen not only to OSS newcomers [57], but also to veterans who contribute to a new project that has different policies [29]. The consequence is that maintainers' information needs can not be properly fulfilled by issue/PR descriptions at the very beginning [9], [14], [18]. They have to request more details in additional iterations of interaction with the contributors, leading to unnecessary communication rounds. Moreover, it takes time for maintainers to wait for contributors' responses [14], [30], [50], which can potentially slow down the processing of an issue/PR.

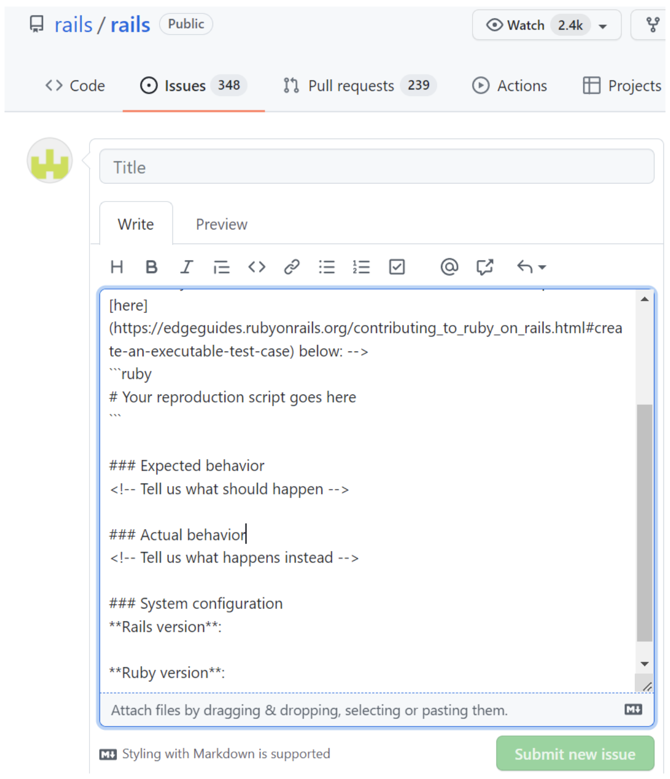
To ensure the informational quality of issues/PRs, many OSS projects suggest contributors first read the contribution guidelines before opening an issue/PR [24], [52], [71]. Additionally, OSS platforms have been providing better tool support for issue/PR submission. For example, GitHub introduced the *issue/PR template* feature [2], which allows project maintainers to customize the information contributors are expected to include when they open new issues and PRs (see Figure 1). Although this feature has been widely used by popular projects for years on GitHub, little is known about its usage in practice and how practitioners perceive it.

In this work, to achieve a better understanding of the template feature, we conduct a large-scale, mixed-methods empirical study of issue/PR templates, by answering the following three research questions:

RQ*$_1$*: What are the contents of templates? We identify templates in 524 popular GitHub projects. We first qualitatively classify the elements contained in template files and then quantitatively explore the frequency of inclusion of each

- *Zhixing Li, Yue Yu, Tao Wang, and Huaimin Wang are with College of Computer, National University of Defense Technology, Changsha, China. E-mail: {lizhixing15, yuyue, taowang2005, hmwang}@nudt. edu.cn*
- *Yan Lei is with School of Big Data and Software Engineering, Chongqing University, China. E-mail: yanlei@cqu.edu.cn*
- *Ying Wang is with the Software College, Northeasthern University, China. E-mail: wangying@swc.neu.edu.cn.*

**Corresponding author: Yue Yu, yuyue@nudt.edu.cn*



~~~
[here]
(https://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html#create-an-executable-test-case) below: -->
```ruby
# Your reproduction script goes here
~~~

Expected behavior
<!-- Tell us what should happen -->

Actual behavior
<!-- Tell us what happens instead -->

System configuration
Rails version:

Ruby version:
```

Attach files by dragging & dropping, selecting or pasting them.
Styling with Markdown is supported
Submit new issue

Fig. 1: GitHub *Issue/PR Template* example.

element among the projects. We find that templates typically contain elements to greet contributors, explain project guidelines, and collect relevant information.

RQ*$_2$*: *What are the impacts of templates?* We examine the git commit history of each template file and identify the adoption time of templates in each project. Then we use regression discontinuity design analyses to evaluate longitudinal effects of adopting templates on the submission, discussion, and resolution of issues/PRs. We observe that after the template adoption, there are fewer monthly submissions of issues and PRs, and issues have fewer monthly discussion comments and longer monthly resolution duration.

RQ*$_3$*: *How do practitioners perceive templates?* We conduct surveys with both contributors and maintainers of the projects using templates to explore their perceptions of templates. Most survey respondents positively rate the usefulness of templates from various aspects. However, they also reported 19 challenges in using templates, such as requiring too many fields, and proposed 9 suggestions to improve the template feature, such as improving user interaction.

In summary, this paper makes the following contributions:

- We present a comprehensive overview of the categories and distribution of the elements contained in templates, which provides a quick reference for project maintainers when creating a template.
- We characterize how project activities change after the adoption of templates, which reveals the longitudinal effects of templates in practice.
- We elucidate practitioners' perceptions of the template feature, which provides evidence for the usefulness of templates and inspires tool designers to improve current support.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes the dataset used in this study. Sections 4, 5, and 6 report the methods and results with respect to our three research questions. Section 7 provides additional discussions. Section 8 discusses the threats to the validity of the study. Finally, we draw conclusions in Section 9.

2 Related Work

We present the related work from two aspects: *informational problems in OSS communication* and *community documentation in OSS projects*.

2.1 Informational problems in OSS communication

Due to the distributed and asynchronous nature of OSS collaboration, text-based communication is the main way to communicate in OSS communities. Many previous studies have investigated OSS communication that happened in mailing lists [31], [32], traditional bug trackers (*e.g.*, Bugzilla) [22], and lightweight issues tracking systems (*e.g.*, GitHub issues) [29], [30]. They showed that OSS developers face extraordinary informational quality problems in communication around both issues and code changes.

For communication around issues, Bettenburg *et al.* [11] conducted a survey with developers from three famous OSS projects to understand how developers use the information in bug reports and what problems they face. They presented the items considered to be most important to developers (*e.g.*, steps to reproduce and stack traces) and revealed a mismatch between what developers need and what bug reporters supply. Davies and Roper [22] manually analyzed 1,600 bugs reports to explore what information users had provided. They observed many incomplete or inaccurate bug reports which often failed to fulfill developers' information needs. Breu *et al.* [14] examined the bug reports collected from the projects Eclipse and Mozilla, and identified a catalogue of questions asked in bug reports. Their quantitative findings showed that many questions were posted to request the missing information for better understanding of the bugs. Kavaler *et al.* [35] studied the relationship between perceived language complexity of issue description and issue resolution latency. They reported that lack of conformity to the project language norm makes an issue less understandable to the community, and thus increases the resolution time of the issue. More specifically, Chen *et al.* [19] studied the usefulness of the logs in bug reports based on 1,500 bug reports of 10 OSS projects. They found that logs helped indicate the location of bugs and inaccurate or insufficient logs extended bug resolution time. Song *et al.* [56] quantitatively analyzed the frequency of observed behavior (OB), expected behavior (EB), and steps to reproduce (S2R) in a large set of bug reports collected from eight OSS projects and one proprietary project. Their results showed that while most bug reports contain OB, only 35.2% and 51.4% contain EB and S2R, respectively. Interestingly, maintainers of more than thousands of OSS projects wrote an open letter to GitHub in 2016 [6], which
```

complained about their experience in dealing with issues missing crucial information and called for improvements to GitHub (the template feature was one of GitHub's responses to this letter). To ensure the quality of issue descriptions, several previous researchers proposed automatic methods to detect [18], [56] and complete [44] the missing information of interest in issues.

For communication around code changes, a study conducted by Ram *et al.* [53] showed that the description significantly affected the reviewability of code changes, *e.g.*, including a motivation in the description was commonly considered to be essential by practitioners. Pascarella *et al.* [50] analyzed 900 code review discussion threads to investigate the information needed by reviewers for proper code review. They constructed a taxonomy of reviewers' information needs, including rationale, code context, and necessity. However, Bacchelli *et al.* [9] reported that reviewers suffered from inferior description of code changes which hindered their understanding of the changes. Ebert *et al.* [23] surveyed code review participants and analyzed code review comments to understand the causes of confusion in code reviews. They found that missing rationale in the description of code changes was one the most prevalent reasons resulting in confusion. Tan *et al.* [59] performed a survey with Linux kernel developers about the communication when submitting patches. Their results showed that low-quality patch descriptions significantly wasted reviewers' effort and most reviewers had rejected patches because of bad descriptions. Moreover, Tan *et al.* [59] also reported that more than half of the survey respondents considered it difficult to write a good patch description. Gousios *et al.* [29] reported a similar finding in a survey with contributors to GitHub projects, which presented that submitting well-described PRs is one of the main challenges faced by contributors. To facilitate PR submission, Liu *et al.* [41] designed an approach to automatically generate PR summaries based on the descriptions of code changes of finer granularity including commit messages and new comments on source code.

We can see that, both generating a good description and achieving a correct understanding of the issues and code changes have always been important but challenging. Templates as a new feature used to ensure the informational quality of issues/PRs deserve a in-depth investigation in terms of its usage in practice and opportunities of improvement.

### 2.2 Community documentations in OSS projects

Contributors need to learn the technical and social aspects of an OSS project to participate. Therefore, project maintainers usually provide documentations establishing standards of behavior for their community and share them in the project's repository. Prior studies have investigated the usage of various community documentations in practice. For example, Prana *et al.* [52] analyzed the contents of README files in OSS projects and found that the most frequently included sections are about specifying what the project does and how to use the project. Elazhary *et al.* [24] inspected the contributing guideline files in dozens of active projects that used continuous integration (CI) tools. They observed five categories of themes, including project orientation and PR acceptance criteria. However, they also found that the documented guidelines were not consistent with the actual practices in some projects. To particularly understand the process of adopting new feature requests, Zhang *et al.* [71] examined the contributing guidelines collected from the most popular GitHub projects. They identified four kinds of information on handling feature requests and highlighted that more than half of the studied projects had a low level of openness with respect to accepting feature requests from the community.

Tourani *et al.* [60] investigated the code of conduct files in OSS projects and reported their popularity and content. They found eleven commonly used code of conduct which generally included five components to create a friendly community. Their investigation also revealed that the adoption of code of conduct can be affected by various factors, such as similarities among communities. By analyzing the discussions about code of conduct, Li *et al.* [37] further studied how code of conduct was used to moderate behavior in practice. Recently, Zhang *et al.* [68] conducted a preliminary study on PR templates particularly in terms of impacts, project characteristics, and non-adoption reasons. Nevertheless, templates' contents deserve a more fine-grained investigation and templates' impacts should be further examined from a longitudinal perspective. Moreover, practitioners' perceptions of templates, especially in terms of application challenges and expected improvements, remain to be investigated.

Our work complement previous studies on OSS community documentation by providing a comprehensive understanding of issue/PR templates.

## 3 Dataset

In this section, we describe how we constructed the dataset [7], including *project selection* and *template identification*.

### 3.1 Project Selection

We began with the top 1,000 most popular projects on GitHub (by September 2020). As done in previous studies [58], [66], the number of stars was used as the proxy for project popularity [12], [45]. We obtained the basic information of these projects via GitHub API [3] and selected projects based on the following criteria:

- *Projects should be for software development:* Inspired by recent work [74], we first removed projects whose programming language was labeled as `Null` by GitHub. Then, we selected projects whose description contains resource-related keywords like *"list"* and *"collection"*. We manually examined the selected projects and removed those for resource sharing or learning.
- *Projects should not be dead.* We are interested in projects that are still maintained, so we excluded projects which did not receive any commit in the past three months.
- *Projects should use GitHub issue/PR.* We removed projects of which the total number of received issues and PRs is 0 at the examination time (*i.e.*, projects used neither GitHub issue nor PR).

Finally, we selected 802 projects involving 40 different programming languages. Figure 2 further presents the distributions of the number of issues, the number of PRs, and age for these projects.

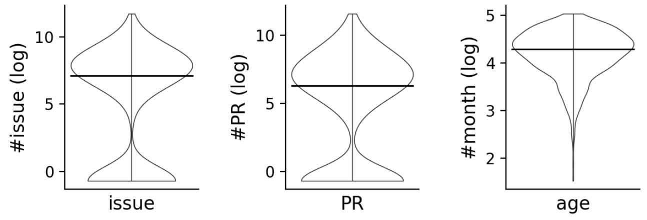


Fig. 2: Distributions of the number of issues, the number of PRs, and age for the studied projects.

## 3.2 Template Identification

For each project, we cloned its source code repository to our local machine. And then we identified the template files and their adoption time in the repository, as follows:

***Identification of existence.*** We searched for the existence of template files from the latest snapshot of the source code repository. To add a template on GitHub, users should create a markdown file named "ISSUE_TEMPLATE/PULL_REQUEST_TEMPLATE" (not case-sensitive) and place it in one of three locations: the root directory, `docs`, and `.github`. Users can also create a subdirectory named "ISSUE_TEMPLATE/PULL_ REQUEST_TEMPLATE" within any of the three directories and then add multiple template files with customized names in the subdirectory. According to such conventions, we identified 1,211 issue template files and 315 PR template files by string matching. These template files involve 524 projects. More specifically, as shown in Figure 3, 277 projects used both issue templates and PR templates, 228 projects used only issue templates, and 19 projects used only PR templates. In comparison, issue templates were used more prevalently than PR templates in the studied projects (505 *vs.* 296).

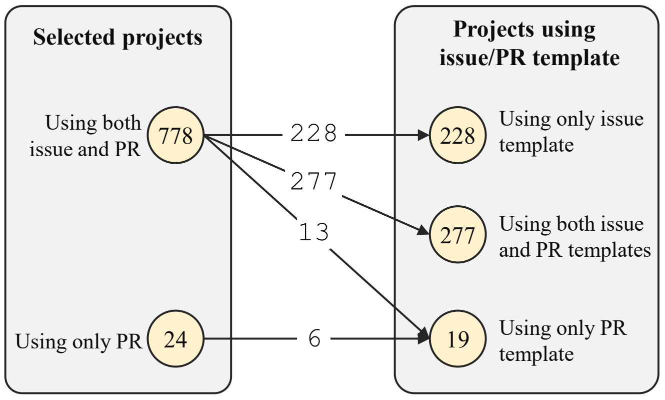


Fig. 3: Distribution of the projects using issue/PR templates.

***Identification of adoption.*** For each template file, we performed the `git log` [17] command to extract all historical commits (*i.e.*, changes) to the file. The first commit which represents the creation of the file was actually the template adoption time in a project. However, in cases where a template file is replaced by multiple template files (or vice versa), the `git log` command on the new file can not recursively extract the change history of the original file, which can cause a bias in the determination of the template adoption time. To tackle this problem, we manually checked the first commit message of each template file. If the commit message contained update-related keywords (*e.g.*, *"update"* and *"move"*) rather than creation-related keywords (*e.g.*, *"add"* and *"create"*), we retrieved the parent commit and verified whether the corresponding snapshot contained an older template file. If it did, we continued to check the first commit message of that old template file. This process was repeated until no older template files were found, and the creation time of the last checked template file was treated as the template adoption time.

## 4 RQ$_1$: Contents of templates

In this research question, we aim to identify the elements contained in issue/PR templates. In the following sections, we present the approach and the results

### 4.1 Methodology

To investigate what a typical issue/PR template looks like, we applied a grounded theory methodology to analyze the content of template files. As we had no predefined categories, we used an open coding approach [39], [75]. Two authors jointly coded a random sample of template files. For each file, they read the content and classified the sentences into meaningful categories. When they disagreed on the classification of a sentence, they discussed to reach an agreement [15]. The coding process was stopped when saturation of findings [26] was reached after analyzing 23 issue template files and 21 PR template files, respectively. Then, all categories are further grouped into higher-level categories, and all authors of the paper review and agree on the final taxonomy. Afterward, according to the defined taxonomy of template elements, the two coders further annotated all template files to quantitatively investigate the distribution of each element across all projects.

### 4.2 Results

Table 1 presents the taxonomy and distribution of elements included in issue and PR templates. These elements can be grounded into three meaningful categories: *Greeting contributors*, *Explaining project guidelines*, and *Collecting relevant information*. The remainder of this section examines the popular and interesting elements in each category in more detail.

#### *4.2.1 Greeting contributors*

At the beginning of some templates, several sentences may be included to greet contributors. For example, some maintainers first ***express gratitude***, especially in PR templates (*e.g.*, *"Thank you for sending a PR. We appreciate you spending time to help improve the Libra project"*), and then ***introduce the template*** in terms of whether some content can be erased (*e.g.*, *"Before submitting your issue, don't hesitate to remove the above introductory text, possible empty sections (e.g. References),*

TABLE 1: The taxonomy and distribution of elements included in issue/PR templates

| **Category** | **Element** | **Distribution** issue templates | PR templates |
|---|---|---|---|
| Greeting contributors | Introduce the template | 33.33% | 31.51% |
| | Express gratitude | 22.16% | 32.53% |
| | Redirect submissions to the right place | 43.71% | 5.14% |
| | Mention OSS culture | 5.59% | 2.74% |
| | Ask for donations | 2.00% | 0.00% |
| Explaining project guidelines | Read community documentation | 22.95% | 38.01% |
| | Add tests for the changes | 0.20% | 53.42% |
| | Update documentation | 0.00% | 39.73% |
| | Format the submission | 5.99% | 33.56% |
| | Search first before submission | 33.93% | 5.48% |
| | Not ask questions | 36.13% | 0.34% |
| | Run the existing tests | 0.00% | 31.16% |
| | Use the latest version | 15.77% | 2.05% |
| | Target the right branch | 2.00% | 13.70% |
| | Discuss first before submission | 2.20% | 12.67% |
| | Sign CLA | 0.00% | 11.64% |
| | Not disclose vulnerabilities | 5.79% | 1.03% |
| | Make atomic contributions | 0.40% | 5.14% |
| | Rebase the commits | 0.00% | 4.11% |
| | Provide comments | 0.00% | 2.74% |
| | List all contributors | 0.00% | 0.68% |
| | First review others' PRs | 0.00% | 0.34% |
| Collecting relevant information | Summary | 56.29% | 59.93% |
| | Type | 64.47% | 26.71% |
| | Environment | 84.83% | 4.11% |
| | Steps to reproduce | 82.44% | 2.74% |
| | Expected/Actual behavior | 73.25% | 7.88% |
| | Related issues | 4.39% | 61.30% |
| | Additional context | 37.72% | 14.38% |
| | Motivation | 19.16% | 27.40% |
| | Suggested solutions | 38.72% | 0.68% |
| | Screenshot | 23.75% | 8.22% |
| | Label | 28.14% | 1.03% |
| | Log/debugging | 15.97% | 0.68% |
| | Stack trace | 15.77% | 0.34% |
| | Side effect | 1.00% | 13.01% |
| | Location | 6.39% | 3.77% |
| | Willingness | 5.59% | 0.34% |
| | Assignee | 2.79% | 2.40% |
| | List of main changes | 0.20% | 4.79% |
| | Status | 0.00% | 4.45% |
| | Severity | 0.40% | 0.00% |
| | Knowledge level | 0.40% | 0.00% |

*and this tip"*) and how some content should be edited (*e.g.*, *"Please replace each `[ ]` by `[X]` when the step is complete, and replace `__` with appropriate data"*). If a project has multiple sub-repositories or moves to a new repository, templates can serve as a good reminder when contributors are submitting an issue/PR and ***redirect submissions to the right place*** (*e.g.*, *"AR.js repository has been moved, please check it out at: https://github.com/AR-js-org/AR.js"*). The redirection facility can also be used to prevent issues/PRs from being mistakenly used (*e.g.*, *"Generally, questions about using Redis should be directed to the [community](https://redis.io/community)"*).

Interestingly, we also found that some projects ***ask for donations*** in their issue templates (*e.g.*, *"Love JHipster? Please consider supporting our collective https://opencollective. com/generator-jhipster/donate"*). Considering that most projects used third-party donation platforms like OpenCollective [48], GitHub can upgrade its sponsor feature [69] to allow users to financially support not only developers but also projects.

### 4.2.2 Explaining project guidelines

To state project conventions and guide contributors to submit issues and PRs, OSS projects usually create a set of community documentation, such as the README file [52] and contributing guideline file [24]. Both issue and PR templates may be used to remind contributors to first ***read community documentation*** to learn the contribution criteria and processes before submission (*e.g.*, *"Before opening: - Read the [contributing guidelines]"* and *"Before filling this issue, please read the manual"*). More specifically, some important contribution practices might be highlighted in templates. Compared with issue templates, PR templates mention more practices regarding submission correctness and quality. For example, contributors are requested to ***run the existing tests*** to make sure that the code changes successfully pass the tests and do not break the project. Contributors also need to ***add tests for the changes*** to prove the fix or new feature works (*e.g., "all new code requires tests to ensure against regressions"* and *"write necessary unit-test to verify your logic correction"*). Besides, some projects suggest contributors ***discuss first before submission***, which can avoid wasting contributors' time and effort on undesired work, especially for new features and significant changes (*e.g., "Requests for new features should first be discussed on the developer forum"*).

As previously reported, PR quality has a broad meaning beyond code [30]. We have found that many projects expected contributors to ***update documentation*** (*e.g., "Please make sure to document all user-facing changes in the `CHANGELOG.md` file"* and *"(If changing the API or CLI) I've documented the changes I've made (in the `docs` directory)"*). Moreover, contributors may be requested to ***format the submission*** in terms of code style (*e.g., "I've followed the _fastlane_ code style and run `bundle exec rubocop -a` to ensure the code style is valid"*), commit message (*e.g., "Commit message has a short title & references relevant issues"*), and PR title (*e.g., "Name the pull request in the form `[#issue] [component] Title of the pull request`"*).

In issue templates, more elements are included to guarantee that the reported issue is valid. A frequently mentioned practice is that contributors should ***not ask questions*** using issues (*e.g., "Please **do not** use the issue tracker for personal support requests"*). Instead, the commonly recommended channels for asking questions about using a project include Stack Overflow [8], projects' forum, Slack [4], *etc.* (*e.g., "Do not use this bug tracker for questions. We have a forum (https://discuss.codemirror.net) for those"* and *"For general questions, please use the pug tag on stack overflow"*). Because the same bug might be encountered by multiple contributors [40], [47], project maintainers generally encourage contributors to ***search first before submission*** to avoid duplicates (*e.g., "Look for similar issues already posted (including closed ones)"* and *"For bugs, do a quick search and make sure the bug has not yet been reported"*).

Additionally, because OSS projects usually have a rapid release cycle [55], the problem found in an old version might be already fixed in a subsequent version. Hence, contributors are recommended to ***use the latest version*** to verify whether the issue is still valid after updating. Maybe due to the limited energy, some projects explicitly stated that they do not deal with issues for outdated versions (*e.g., "Please do not report an issue for a version of PHPUnit that is no longer supported"*).

### 4.2.3 Collecting relevant information

To comprehensively understand the submitted issues and PRs, project maintainers need to collect a variety of relevant information. Generally, contributors are requested to first provide a ***summary*** to briefly describe the issue/PR (*e.g., "A clear and concise description of what the bug is"*). Issues/PRs submitted to achieve different kinds of tasks tend to be processed with different priorities, *e.g.*, bugs usually have higher priority than new features [30]. To help maintainers reduce the effort required to classify a high volume of submissions, contributors are expected to specify the ***type*** of their issues/PRs. In practice, we found two ways used to collect the type information. The first approach is adding multiple template files, *e.g., "ISSUE_TEMPLATE/bug_report.md"* for reporting bugs and *"ISSUE_TEMPLATE/feature_request.md"* for requesting new features. When contributors click to create a new issue, they will have the option to choose which type of issue they intend to submit. The second approach is adding a field in the template to ask contributors to freely describe the type of their submission or to choose from a set of predefined types. Besides, to quickly identify why an issue/PR is needed, especially for feature requests, some projects hope contributors to report the ***motivation*** behind the submission (*e.g., "Explain why this is a bug or a new feature for you"* and *"Any relevant use-cases that you see"*).

In addition to the above information which is commonly gathered in both kinds of templates, project maintainers collect more other information in issue templates. The most frequently collected information is ***environment***, including platform, project version, configuration, *etc.* (*e.g., "Which versions of Redux, and which browser and OS are affected by this issue?"*). Interestingly, we observed that some projects provided contributors with scripts for easily collecting the environment information (*e.g., "Run `gatsby info –clipboard` in your project directory and paste the output here"* and *"run `ng version` and paste output below"*).

To figure out how the problem arose, project maintainers usually ask contributors to clearly describe the ***steps to reproduce*** the reported issue. The reproduction information can be provided as a step-by-step textual description (*e.g., "1. First Step 2. Second Step 3. and so on..."* and *"1. Go to `....` 2. Click on `....` 3. Scroll down to `....` 4. See error"*), code snippets (*e.g., "**Code snippet to reproduce** ```rust # Your code goes here # Please make sure it does not require any external dependencies```"*), or demo applications (*e.g., "we recommend creating a small repo that illustrates the problem"*). Additionally, the ***expected/actual behaviors*** illustrate what contributors expect to happen and what actually happens, which enable maintainers to identify exactly what the problem is. For GUI-related issues particularly, a ***screenshot*** is useful to demonstrate the problematic behavior. If there is an error, contributors can include the ***stack trace*** or ***log/debugging*** information to help maintainers diagnose the problem (*e.g., "If you're reporting a crash, please copy the stack trace below"*).

To facilitate the resolution of the submitted issues, some projects asked whether there is any ***suggested solution*** (*e.g., "Do you have any ideas on how we could fix this"*) or whether contributors have the ***willingness*** to work on the issues (*e.g., "Would you be willing to resolve this issue by submitting a Pull Request?"* and *"Is this a feature you're prepared to implement, with support from us?"*). Interestingly, a few maintainers wish to know contributors' ***knowledge level*** so that they can *"formulate the response in an appropriate manner"*.

Since abundant information has been collected in issues, maintainers generally request PR contributors to link the ***related issues***, if any (*e.g., "If your pull request relates to any existing issues, please reference them by using the issue number prefixed with #"*). To thoroughly evaluate a PR, especially when it introduces new features, some maintainers were concerned about whether it introduced any ***side effect***, such as compatibility problems (*e.g., "If this PR introduces a breaking change, please describe the impact and a migration path for existing applications"*). In some projects, developers are allowed to submit work that is still in progress. In order to be quickly informed of such submissions, maintainers ask contributors to submit a draft PR [1] (*e.g., "If you intend to work on PR over several days, please, create draft pull requests"*) or include a specific mark in PR title (*e.g., "If the PR is work in progress, please add the prefix `WIP:` to the beginning of the title"*) to clearly indicate the ***status*** of PRs.

***RQ*$_1$*:*** Templates contain a wide variety of elements to greet contributors (*e.g., expressing gratitude*), explain project guidelines (*e.g., not asking questions* and *adding tests for the changes*), and collect relevant information (*e.g., steps to reproduce* and *related issues*).

## 5 RQ$_2$: Impacts of templates

The purpose of this research question is to investigate whether several project activity indicators change after the adoption of templates. We first present the method and then report the results.

### 5.1 Methodology

To investigate whether the template adoption has impacts on project activities, we conducted a longitudinal analysis. We computed monthly project-level data for a period of 24 months centered around the adoption date of templates, and employed Regression Discontinuity Design (RDD) [21], [33] to model the impacts of template adoption along the following three dimensions:

***n_submissions***: the number of new issues/PRs submitted during a month.

***n_comments*** : the number of discussion comments on issues/PRs.

***close_time*** : the time spent from issue/PR submission to close.

For the last two response variables, we first computed the value at the level of individual issues/PRs, and then computed the median over all issues/PRs per time window. Figure 4 visualizes the trends in the three response variables 12 months before (month index ranging from `-12` to `-1` ) and after (month index ranging from `+1` to `+12`) the adoption of templates.

To model the effects of template adoption, we followed the prior studies [61], [72] and used three variables:

***time***: the index of time window from the start to the end of the observation period, ranging from `1` to `24`.

***intervention***: a binary value indicating whether a time window is before (*intervention*=0) or after (*intervention*=1) the template adoption.

***time_after_intervention***: the index of time window post to the template adoption, ranging from `1` to `12`.

Bases on the previous work [65], [67], we also controlled for the following known variables:

***proj_age***: the time span from project creation to the current month, measured in months.

***n_commits***: the number of commits the project has received until the current month.

***n_stars***: the number of stars of the project until the current month.

***n_forks***: the number of forks of the project until the current month.

***team_size***: the number of core team members actively participating in issue/PR discussions in the current month.

***experience***: the median number of prior issues/PRs submitted by the contributor computed over all issues/PRs per month.

***n_open_tasks***: the median number of issues and PRs still open at current issue/PR creation time computed over all issues/PRs per month.

***desc_length***: the median length of the issue/PR description computed over all issue/PRs per month.

We also used project name and programming language to model random effects. To prevent multicollinearity from affecting our coefficient estimates, we followed the recommended criterion [20] and excluded variables for which the VIF score is higher than 5. When modeling issue/PR submissions, we excluded the variables *team_size*, *experience*, and *desc_length*, as they are computed after issue/PR submission.

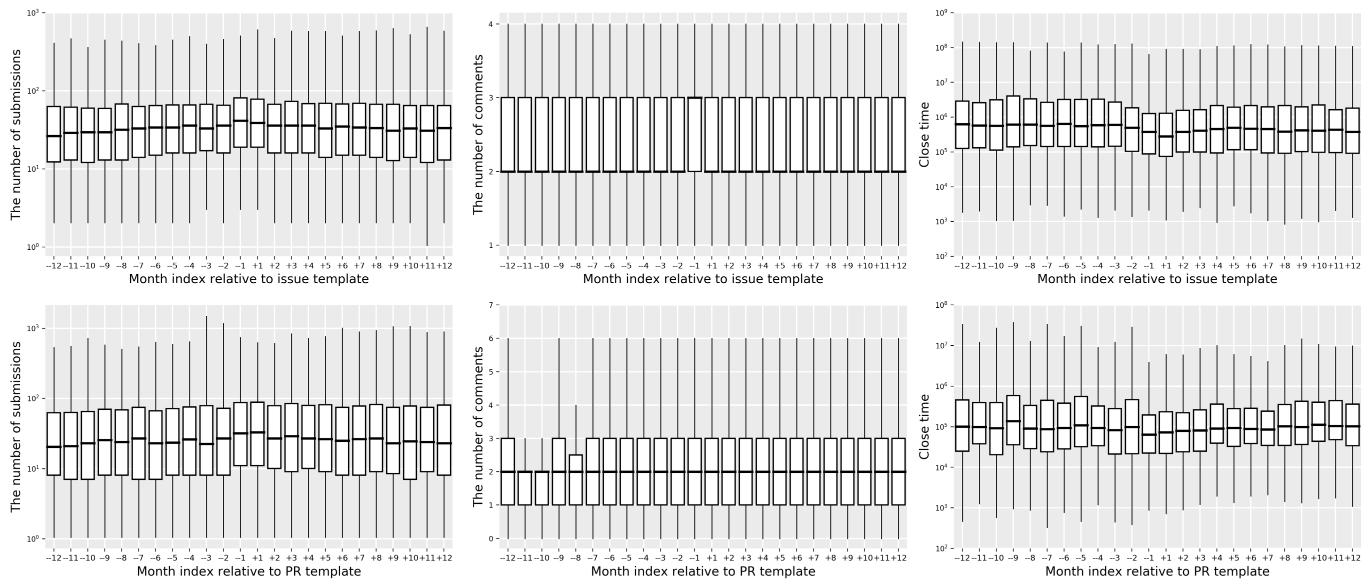
### 5.2 Results

Tables 2$\sim$4 present the results of the fitted RDD models. In addition to the coefficient, standard error, significance level (indicated by stars), and the sum of squares, each table also reports the marginal R-squared ($R^2$) and conditional $R^2$ to quantify the goodness-of-fit of models. We can see that the models can explain more variability in the data when considering both the fixed and random effects (conditional $R^2 >$ marginal $R^2$). In the following, we discuss the models for each response variable.

#### *5.2.1 Submission*

Regarding the model of issue submissions (the columns belonging to *Issue* in Table 2), we observe that *n_stars* explains the largest amount of variability. Its positive coefficient indicates that the more GitHub users starring a project, the more issues the project will receive. This is expected as a larger *n_stars* indicates that a project is being used more widely [13] and more users may submit issues to report the experienced software problems or expected features. As for the three template-related variables, the coefficient for *intervention* is significant, indicating that the model detects a discontinuity at the adoption time of templates. The significant, negative coefficient for *time_after_intervention* suggests that the number of submitted issues presents a slight decreasing trend after adoption.

For the model of PR submissions (the columns belonging to *PR* in Table 2), we see that *commits* explains the largest amount of variability, indicating that the more commits a project has, the more PRs the project will receive. This is in line with the previous finding that larger projects tend to receive more PRs [65]. As for template-related variables, the model does not detect any discontinuity at the adoption time of templates as the coefficient for *intervention* is not significant. However, the positive trend in the number of PR



(a) Submissions (b) Discussion comments (c) Close time

Fig. 4: Trends in response variables before and after the adoption of issue and PR templates, top and bottom respectively (the horizontal line in each box is the median value across all projects).

TABLE 2: Issue/PR submission models. The response is log(*n_submissions*) per month.

| | Issue | | PR | |
|---|---|---|---|---|
| | Coeffs (Err.) | Sum Sq. | Coeffs (Err.) | Sum Sq. |
| (Intercept) | 1.996 (0.291) *** | | 2.273 (0.489) *** | |
| log(prj_age) | -0.548 (0.074) *** | 10.1 | -0.443 (0.125) *** | 3.5 |
| log(n_stars) | 0.260 (0.014) *** | 61.7 | 0.104 (0.021) *** | 6.6 |
| log(commits) | 0.106 (0.009) *** | 23.4 | 0.175 (0.018) *** | 26.7 |
| log(n_open_tasks) | 0.113 (0.009) *** | 32.0 | 0.135 (0.017) *** | 18.4 |
| time | 0.001 (0.002) | 0.1 | 0.008 (0.004) * | 1.2 |
| interventionTRUE | -0.040 (0.019) * | 0.8 | -0.006 (0.033) | 0.0 |
| time_after_intervention | -0.035 (0.003) *** | 28.3 | -0.029 (0.005) *** | 10.1 |
| Marginal $R^2$ | 0.169 | | 0.114 | |
| Conditional $R^2$ | 0.800 | | 0.800 | |

*** $p<0.001$, ** $p<0.01$, * $p<0.05$

submissions before the template adoption (the coefficient for *time* is positive) is reversed, toward a decrease after adoption (the coefficient for *time_after_intervention* is negative).

Overall, we observe a slight decreasing trend in both the number of issue and PR submissions after the template adoption. We speculate that it may be because templates prevent contributors from submitting invalid issues/PRs, as discussed in Section 4 (*e.g.*, *not ask questions* and *search first before submission*). Another possible reason is that some contributors have challenges in filling out the templates and finally abandon submitting issues/PRs, as reported in Section 6.2.2. However, since there is a chance for a software issue to be duplicately reported or solved by multiple contributors [40], [47], it seems that the advantages of templates in avoiding undesired contributions outweigh the disadvantages in causing contribution abandonment.

### 5.2.2 Discussion

As for the model of issue discussion (the columns belonging to *Issue* in Table 3), we find that *desc_length* is strongly associated with the number of issue comments. Its positive coefficient indicates that issues with a longer description are likely to receive more comments. This is expected because issues described through longer texts usually have higher complexity [67], and thus tend to be highly discussed. All template-related variables present significant effects. The negative coefficient for *time* means that the number of issue comments is slightly reduced as time passes. And the decreasing trend is accelerated by the template adoption, as indicated by the negative coefficients for both *intervention* and *time_after_intervention*.

TABLE 3: Issue/PR discussion models. The response is log(median *n_comments*) per month.

| | Issue | | PR | |
|---|---|---|---|---|
| | Coeffs (Err.) | Sum Sq. | Coeffs (Err.) | Sum Sq. |
| (Intercept) | 0.845 (0.117) *** | | -0.780 (0.252) ** | |
| log(prj_age) | -0.079 (0.026) ** | 0.8 | 0.027 (0.063) | 0.0 |
| log(n_stars) | -0.046 (0.009) *** | 2.2 | 0.040 (0.016) * | 1.0 |
| log(commits) | 0.024 (0.006) *** | 1.4 | 0.038 (0.013) ** | 1.3 |
| log(team_size) | 0.100 (0.009) *** | 10.4 | 0.148 (0.021) *** | 7.9 |
| log(n_open_tasks) | -0.005 (0.005) | 0.1 | 0.053 (0.012) *** | 2.9 |
| log(experience) | -0.030 (0.005) *** | 3.6 | -0.055 (0.005) *** | 18.9 |
| log(desc_length) | 0.111 (0.009) *** | 12.8 | 0.071 (0.007) *** | 18.5 |
| time | -0.004 (0.001) * | 0.6 | 0.003 (0.003) | 0.2 |
| interventionTRUE | -0.043 (0.014) ** | 0.9 | -0.092 (0.026) *** | 2.0 |
| time_after_intervention | -0.005 (0.002) * | 0.5 | 0.000 (0.004) | 0.0 |
| Marginal $R^2$ | 0.118 | | 0.150 | |
| Conditional $R^2$ | 0.522 | | 0.649 | |

*** $p<0.001$, ** $p<0.01$, * $p<0.05$

For the model of PR discussions (the columns belonging to *PR* in Table 3), we find that *experience* and *desc_length* are strongly associated with the number of PR comments.

Similar to the previous model, *desc_length* presents a positive effect, indicating that PRs with a longer description tend to have more comments. The negative effect of *experience* shows that PRs submitted by more experienced contributors are being discussed less. We present an assumption that contributors with higher experience are more likely to submit high-quality PRs [54], [62] which tend to trigger a lower number of revisions and discussions. With respect to template-related variables, the coefficient for *intervention* reveals a discontinuity at the template adoption time. However, after adoption, templates do not impact PR discussion in the long run since the coefficient for *time_after_intervention* is not statistically significant.

Overall, we observe that with the adoption of templates, issues are being discussed less, while PR discussions remain unaffected. We assume that templates help to collect more relevant information for understanding issues, and consequently, comments posted to request the missing information [14] is fewer. However, for PRs, even if they are not properly described before template adoption, maintainers can still understand the PRs by examining the code changes and commit messages [41], [73] (to a certain degree at least) without having to launch a discussion thread. Moreover, PR discussions usually cover a wide variety of topics and maintainers may mention multiple concerns in a single comment [38]. Therefore, even though templates can avoid certain problems at the initial PR submission, they do not necessarily reduce the total number of PR comments.

#### 5.2.3 Resolution

Considering the model of issue resolution (the columns belonging to *Issue* in Table 4), we note that *n_open_tasks* explains most of the variability, indicating that the more tasks that are still open for discussion, the more time it takes for maintainers to close the subsequent issues. This is expected as a larger *n_open_tasks* means that maintainers are experiencing a higher workload and they are more likely to have low availability. As for template-related variables, the model does not detect any discontinuity at the template adoption time, as the coefficient for *intervention* is not significant. Before the template adoption, we can observe a slight decreasing trend in time-to-close of issues (the coefficient for *time* is negative). However, after the adoption, the decreasing trend in is reversed, toward a slight increase (the coefficient for *time_after_intervention* is positive).

For the model of PR resolution, (the columns belonging to *PR* in Table 4), we observe that *experience* explains most of the variability, indicating that PR submitters' experience has a strong effect on the time to close PRs. Moreover, the negative coefficient of *experience* reinforces the previous finding that PRs submitted by more experienced contributors have lower review latency [42], [70]. Regarding the template-related variables, *intervention* has a statistically significant, negative effect, indicating that the adoption of templates reduces the time-to-close of PRs. However, the coefficient for *time_after_intervention* shows that the effect is not significant in the long run.

Overall, we observe that after the template adoption, on average more time is required to close issues. However, the adoption of templates is not associated with PR resolution. A possible reason for the increase in the monthly time to close issues is that templates reduce the number of invalid issues which are usually closed faster than common issues [49], and consequently, the time-to-close computed over the issues submitted after the template adoption presents an increasing trend. Regarding PR resolution, we assume that PR revision and evaluation is usually a complex and iterative process [25], [34], [67] and it is not easily affected by templates applied in the very early stage.

TABLE 4: Issue/PR resolution models. The response is log(median *close_time*) per month.

| | Issue | | PR | |
|---|---|---|---|---|
| | Coeffs (Err.) | Sum Sq. | Coeffs (Err.) | Sum Sq. |
| (Intercept) | 6.437 (0.595) *** | | 8.311 (0.641) *** | |
| log(prj_age) | -0.297 (0.141) * | 7.9 | 0.037 (0.154) | 0.1 |
| log(n_stars) | -0.393 (0.043) *** | 153.5 | 0.010 (0.050) | 0.1 |
| log(commits) | -0.064 (0.028) * | 9.3 | 0.049 (0.043) | 2.5 |
| log(team_size) | -0.597 (0.043) *** | 342.0 | -0.325 (0.066) *** | 44.6 |
| log(n_open_tasks) | 0.811 (0.026) *** | 1778.8 | 0.490 (0.040) *** | 278.4 |
| log(experience) | -0.004 (0.022) | 0.1 | -0.315 (0.017) *** | 641.3 |
| log(desc_length) | 0.139 (0.042) *** | 19.5 | 0.287 (0.022) *** | 308.9 |
| time | -0.030 (0.007) *** | 35.8 | 0.003 (0.009) | 0.2 |
| interventionTRUE | -0.037 (0.062) | 0.7 | -0.324 (0.088) *** | 24.8 |
| time_after_intervention | 0.062 (0.009) *** | 88.5 | 0.024 (0.013) . | 6.9 |
| Marginal $R^2$ | 0.197 | | 0.231 | |
| Conditional $R^2$ | 0.612 | | 0.561 | |

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

***RQ$_2$:*** After adopting templates, on average, the monthly volume of incoming submissions slightly decreases for both issues and PRs. Moreover, there are fewer monthly discussion comments and longer monthly resolution latency on issues after template adoption.

## 6 RQ$_3$: Perceptions of templates

With this research question, we intend to understand OSS practitioners' perceptions of issue/PR templates, with respect to templates' usefulness, the challenges practitioners face in using templates, and the improvements they would like to add to the template feature. Below, we describe the method and the results.

### 6.1 Methodology

To understand how practitioners perceive issue/PR templates, we conducted surveys with contributors and maintainers of the studied projects, respectively. In both surveys, we first measured the usefulness of templates from multiple aspects inspired by templates' contents identified in Section 4: *"collect more relevant information"*; *"standardize issue/PR description"*; *"explain project guidelines"*; *"guide contributors to more easily submit issues/PRs"*; *"lead contributors to submit high-quality issues/PRs"*; *"warn about common problems"*; *"discourage undesired submissions"*. Specifically, we asked participants about their level of agreement with these statements using a 5-point Likert scale ("Strongly Disagree", "Disagree", "Neither Agree nor Disagree", "Agree", and "Strongly Agree"). We also included open-ended questions for participants to provide feedback on what challenges they are facing when using templates and what improvements they suggest to enhance the template feature.

From the projects using templates presented in Section 3, we selected the maintainers and contributors who submitted issues/PRs to these projects after template adoption. Then, we emailed surveys to a total of 500 maintainers and 2,000 contributors who were randomly selected from the candidates (yielding a 95% confidence level with a 1.96% error margin). After two weeks, we received 205 responses (42 from maintainers and 163 from contributors). Regarding the demographic information of respondents, 71.4% of maintainers have over 5 years of OSS experience and 71.2% of contributors have more than 3, which aligns with recent surveys of OSS practitioners [27], [30], [38]. For free-text responses to open-ended questions, we applied the open coding approach to classify them into meaningful categories.

## 6.2 Results

### 6.2.1 Usefulness

Figure 5 shows the perceived usefulness of templates in different tasks. For each task, we present contributors' responses and maintainers' responses in the same row. In addition to the frequency of responses, we add the aggregated score result (*i.e.*, median) for each bar, which ranges from 1 to 5, corresponding to the five answer options (as indicated by the legend of the figure).

From the figure, we can see that the majority of survey respondents agreed with most of the statements (median $> 3$), indicating the commonly recognized usefulness of templates. Especially, more than 80% of both contributors and maintainers considered templates as useful in collecting more relevant information and standardizing issue/PR descriptions. However, about 50% of maintainers did not positively rate templates' usefulness in warning about common problems and discouraging undesired submissions. This might be explained by the fact that some contributors just ignored the templates when opening a new issue/PR, as reported in Section 6.2.2.

Survey respondents also reported other benefits that they thought templates can bring in, which include suggesting the common solutions (*e.g.*, *"asking the software version the submitter might realize the problem can be solved by upgrading"*), facilitating impression formation (*e.g.*, *"project makes a respectable impression"*), and supporting multitasking [64] across projects (*e.g.*, *"it is reassuring to have a template, so that I can be confident I am remembering the right process and data fields. I do not want to accidentally apply the process and data fields from a different project."*)

### 6.2.2 Challenges

From survey responses on the challenges of using templates, we collected 19 challenges as shown in Table 5. For each challenge, the table also reports the number of mentions among contributors and maintainers, respectively.

TABLE 5: Challenges experienced when using issue/PR templates

| **Challenge** | **CTRs** | **MTRs** |
|---|---|---|
| Templates ask for irrelevant information | 15 | - |
| Templates require too many fields | 13 | 4 |
| Templates' text is too verbose | 8 | 2 |
| Template categories are not complete | 5 | 2 |
| Duplicate information requests | 4 | - |
| Templates are hard to follow | 4 | 1 |
| Additional time to create an issue/PR | 3 | - |
| Templates are poorly written | 3 | 1 |
| Language barriers | 1 | - |
| Lots of manual editing | 1 | - |
| Trouble with markdown syntax | 1 | - |
| Some information is not requested | 1 | - |
| Similar contents hinder general searching | 1 | - |
| Templates are ignored | - | 8 |
| Difficulty to create an ideal template | - | 8 |
| Lack of measures to enforce something | - | 3 |
| Burden of making templates up-to-date | - | 2 |
| Difficulty to configure the template | - | 2 |
| Templates are poorly filled | - | 2 |

*Abbreviations:* CTRs-Contributors, MTRs-Maintainers

From contributors' perspective, the top challenges are relating to the information request. Thirteen respondents complained that ***templates require too many fields***. Particularly, some of them pointed out: *"it raises the threshold of minimum work required to report an issue"* and *"turns it into a big chore to report a simple thing"*. And the possible negative consequences include *"you might feel demotivated to go that extra mile"* and *"can scare away contributors"*. This challenge was also confirmed by 4 maintainers. Maybe due to a wealth of information being collected, 15 contributors mentioned that ***templates ask for irrelevant information*** for some submissions. One example was *"Often the template contains many questions that seem irrelevant to the issue or PR that I create"*. Additionally, as for the way of offering the required information, ***lots of manual editing*** was pointed as a challenge by one respondent.

Some contributors suffered from problems with templates' text. Eight of them reported that ***templates' text is too verbose*** (*e.g.*, *"It can be too much text, overwhelming people"* and *"too much comments inside a template"*). They thought this challenge may *"ruin the contributors experience and slow down the process of opening issues/PR"* or even *"discourage submissions"*. Unfortunately, ***duplicate information requests***, mentioned by 4 contributors, can make a template more verbose than it should be. Another 3 contributors mentioned that ***templates are poorly written***, which makes it harder for contributors to understand the template. Additionally, two respondents cited ***language barriers*** and ***trouble with markdown syntax*** as challenges when reading templates.

Contributors also reported challenges regarding the completeness of templates. As aforementioned, maintainers can add multiple templates per type. However, five contributors' feedback revealed that ***template categories are not complete*** in some projects (*e.g.*, *"Sometimes an issue doesn't fit in any of the categories so we can't follow the proposed issue/PR structure"*). Actually, two maintainers considered it challenging to *"make sure there are templates for all situations"*. More specifically, with respect to the detailed content of templates, one contributor reported that ***some information***

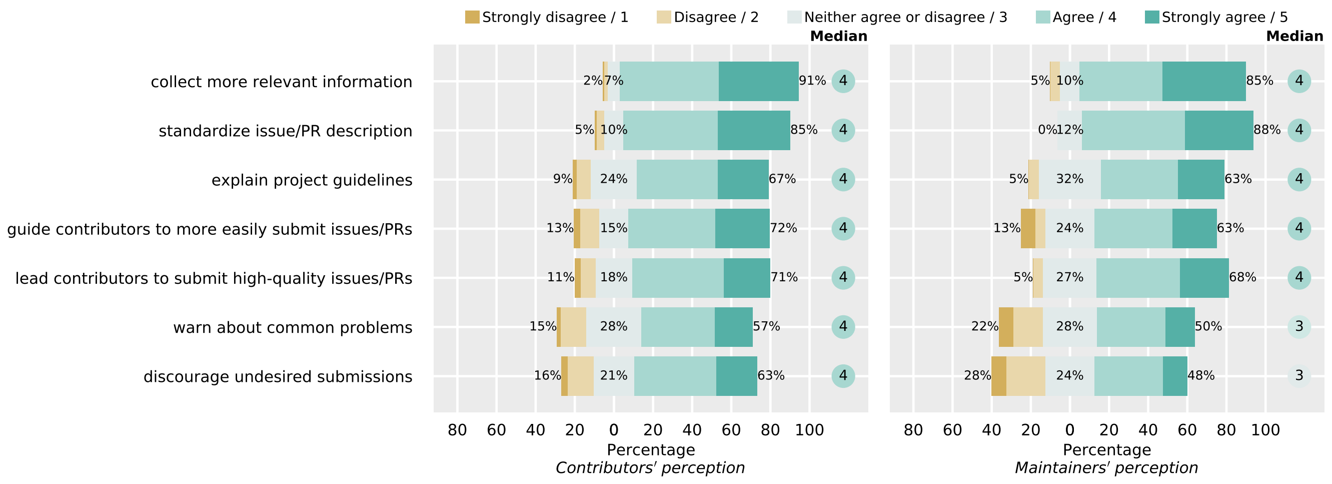


Fig. 5: Practitioners' perception of the usefulness of templates

***is not requested***. It seems that including a *"Anything else"* section in templates, where contributors are free to write what they think maintainers should know, helps to guarantee the completeness of collected information.

From the maintainers' perspective, we can observe that the main challenges are about template adherence. The most frequently mentioned one is that ***templates are ignored***, as a maintainer explained: *"Often bug reporters do not read the template at all"*. In some cases, the template may be partially completed with certain fields ignored, and consequently, several maintainers complained about the ***lack of measures to enforce something***. Additionally, even all fields of a template are filled, a potential challenge is that ***templates are poorly filled***, as reported by 2 maintainers.

Furthermore, maintainers faced challenges in ensuring templates' quality. In particular, eight respondents referred to the ***difficulty to create an ideal template***. For example, one maintainer specified that *"The main challenge is deciding how much information to ask for. Too much, and contributors will opt to not make issues due to how hard it is to fill it out. Too little, and maintainers will need to follow-up for extra details on almost every issue"*. Another 2 maintainers mentioned the ***burden of making templates up-to-date*** (*e.g.*, *"updating the template when it is no longer relevant"*).

### 6.2.3 Improvements

In the surveys, respondents suggested improvements on the template feature, which are summarized in Table 6. As in Table 5, this table also lists the number of mentions of each suggestion among contributors and maintainers, respectively.

For both contributors and maintainers, the most recurrent suggestion is ***improving user interaction***. Practitioners suggested this improvement mainly because a large set of comments, instructions, and information requests are mixed in a single textarea and it is somewhat inconvenient to read and fill out a template. The solutions suggested to tackle this problem are various, ranging from *"syntax highlighting"* to *"provide an actual form with multiple fields"*. Actually, GitHub recently supported to create a template with web form fields using YAML syntax (in beta) [5]. For future research, it would be interesting to investigate whether this update can fulfill practitioners' expectation of better user interaction. Another new feature requested to make it less burdensome to submit and inspect issues/PRs is ***improving automation***. For example, one contributor expected: *"if the tool is capable of detecting unit tests in the PR, it could not show this item to the contributor (or automatically check a check box)"*. ***Providing good template examples*** is also a recurrent suggestion (*e.g.*, *"Examples and defaults I can pick from"*). Since many templates have a similar structure, a list of ready-to-use templates for certain universal cases would make it easier for a project to configure templates. Interestingly, several respondents desired ***multiple templates per type*** even though GitHub already supported this feature.

TABLE 6: Suggested improvements on the template feature

| Suggested improvement | CTRs | MTRs |
|---|---|---|
| Improve user interaction | 13 | 7 |
| Provide good template examples | 5 | 2 |
| Improve automation | 4 | 3 |
| Fields can be separately sorted | 2 | - |
| Better labeling support | 2 | - |
| More flexible workflow | 1 | - |
| Multiple templates per type | 1 | 1 |
| Provide a way to enforce something | 1 | 5 |
| Improve template visibility | - | 2 |

*Abbreviations:* CTRs-Contributors, MTRs-Maintainers

For maintainers, one of the commonly mentioned improvements is ***providing a way to enforce something***. For example, one maintainer suggested: *"making some parts mandatory (so they cannot be removed)"*. In addition to that, a few maintainers expected to ***improve template visibility***, *e.g.*, by refining the layout (*"adding an explanation about the template next to the template"*).

For contributors, they also advocated better integration between templates and existing functionalities on GitHub. Two respondents suggested that ***fields can be separately sorted*** so that *"issues that belong to a similar topic can be filtered to ease reading and going through them instead of just* *listing them in chronological order"*. Another two respondents mentioned ***better labeling support*** (*e.g.*, *"the person submitting the issue could select from a selection of labels open to them by project maintainers"*), which helps *"to segment issues a bit better"*. Contributors also requested ***more flexible workflow*** as presented in this quote: *"Simpler initial reporting with a follow up template if the developers need it"*.

***RQ*$_3$*:*** Most OSS practitioners positively rated the usefulness of templates in supporting a variety of tasks. However, contributors complained about the excessive and irrelevant information requests and verbose text, and maintainers had difficulty in creating high-quality templates and suffered from contributors' neglect of templates. Both contributors and maintainers proposed suggestions to improve the template feature, including better user interaction and advanced automation.

## 7 Discussions

Based on our findings, we provide additional discussion and propose implications for tool designers.

### 7.1 Template roles

As presented in Section 4 $\sim$ Section 6, templates can play a variety of roles, including *communication protocols*, *pre-submission gatekeepers*, and *just-in-time mentors*.

***Communication protocols.*** To efficiently collaborate on issues/PRs, it is important for both contributors and maintainers to clearly, thoroughly, and accurately understand and satisfy the other party's information needs with low communication costs. Nevertheless, due to the differences in OSS participants' experience and expertise and the diversity of OSS projects' participation culture and collaboration style, it is often challenging to achieve effective communication between maintainers and contributors [14], [23], [30]. One of the main benefits of templates would be in establishing a set of *communication protocols* for information exchange. These protocols help maintainers specify what information they expect to obtain in issue/PR descriptions. At the same time, these protocols help contributors learn about what information they should provide and express themselves in a manner that resonates with the project community. Templates, therefore, are beneficial to transfer information among OSS participants and reduce communication overhead. They would be particularly useful when developers contribute to multiple different projects or join a new project [57], [64]. However, to further improve templates' effectiveness as communication protocols, the CSCW and HCI communities should design better user interaction mechanisms as suggested in practitioners' feedback.

***Pre-submission gatekeepers.*** For popular OSS projects, maintainers have to deal with a big volume of incoming contributions and it can be challenging to perform the triage and evaluation tasks [30]. Various tools (*e.g.*, bots [66] and CI [67]) have been used as *post-submission* gatekeepers to ensure contribution quality and reduce maintainers workload. Our results show that templates can be used as *pre-submission* gatekeepers increasing the likelihood that OSS projects receive high-quality issues and PRs. On the one hand, templates explain what makes a good issue/PR prior to submission, which helps to ensure that the submitted issues/PRs contain fewer problems. On the other hand, templates warn contributors about what kinds of submissions are considered as undesired by maintainers, preventing contributors from submitting invalid issues/PRs.

***Just-in-time mentors.*** Community-based OSS projects have high dependency on volunteers' contributions and thus a continuous influx of newcomers is important for the sustainability of these projects [57]. To support newcomers' onboarding, the proposed strategies include mentoring [16], training [57], and providing contribution guidelines [24], which are mainly applied *before* newcomers start their contributions. As a complement to these strategies, templates act as *just-in-time* mentors, guiding the activities of contributors at the time of filling a new issue/PR. Particularly, since many OSS contributors are one time contributors [36] and casual contributors [51], it seems more reasonable for them to quickly figure out the contribution process and criteria at the submission time without having to spend additional time learning the project in advance.

### 7.2 Template trade-offs

Our findings show that although templates are widely used in popular projects and most practitioners provided positive feedback on their usefulness (see Figure 5), contributors are still facing various challenges when filling out templates (see Table 5). Maintainers of OSS projects should be aware of the following tradeoffs when using templates.

***Completeness vs. Effortlessness.*** Complete information is important for project maintainers to understand the submitted issues/PRs and make informed decisions. Therefore, templates were generally perceived as useful by maintainers in collecting more relevant information upon issue/PR submission. However, many contributors complained about the heavy burden of providing a mass of information requested in templates. Especially for simple problems and trivial code changes, asking for too much information may result in negative contribution experience and lead contributors to abandon submitting an issue/PR [39]. Therefore, maintainers should consider the trade-off between the completeness of information requests and the effortlessness of information provision.

***Exhaustivity vs. Readability.*** In some templates, project maintainers include detailed comments next to the section headings for further elaboration, *e.g.*, explaining the necessity of the requested information and suggesting the way of providing such information. Providing exhaustive elaboration is helpful for inexperienced contributors and newcomers who lack knowledge about a specific project. But then each contributor would have to go through a lengthy template which can seem verbose and might discourage some submissions. Therefore, maintainers should consider the trade-off between the exhaustivity and readability of template content. This can be partially supported by better UI design: briefly listing all elements and allowing easy navigation to the exhaustive elaboration about each element.

***Standardization vs. Flexibility.*** Standardization of issues/PRs helps to reduce maintainers' cognitive workload.

However, not all issues/PRs can be characterized with the same set of attributes. As reported by some contributors, even a project added multiple templates, they still could suffer from incomplete template categories. Even worse, if the defined attributes set is too large, contributors would find many information requests are irrelevant to their submission, which may lead contributors to think *"I am not sure whether I should submit at all"*. Therefore, maintainers should consider the tradeoff between the standardization and flexibility in describing issues/PRs. As one of our survey respondents pointed out it is important to *"Let contributors know that the template is a recommendation, not a strict rule"*.

### 7.3 Implications for tool design

As shown in Table 6, there is room for improvements of the template feature, *e.g.*, better user interaction and advanced automation. In addition to those specific suggestions, we suggest GitHub as well as other OSS platforms focus on the following general topics.

***Structure.*** Our results to RQ1 show that a template might contain a wide variety of elements which actually can be grouped into several meaningful categories. A well-structured template, *e.g.*, organizing the elements according to their categories rather than placing them arbitrarily, makes intuitive sense and facilitates the readability of contents. One possible improvement of the template feature would be to allow maintainers to freely add multiple separated subareas to group related elements.

***Intelligence.*** The template feature would be more useful if it is designed with a higher intelligence level. For example, a lot of manual work would be reduced if templates can automatically check whether certain specifications were followed. Additionally, the template feature can be enhanced with awareness (*e.g.*, adaptively adjusting template contents according to contributors experience level) and recommendation (*e.g.*, monitoring template adherence and recommending which part of the template needs improvement).

***Integration.*** The template feature should be integrated with other tools in a more seamless way so that the information collected in templates can be effectively used by other tools. For example, some categorical information (*e.g.*, submission type) can be used to label the submitted issues/PRs. Besides, templates should avoid duplicating the work of other tools, *e.g.*, maintainers do not have to require CLA information in templates if they have adopted a bot to check such information after PR submission.

## 8 Threat to Validity

In this section, we discuss limitations and potential threats to validity of our work as follows.

***Internal Validity.*** It is possible that we introduced bias during the manual analysis of contents of template files and responses of open-ended survey questions. To mitigate this threat, we assigned two authors to perform the manual processes in pairs and they carefully discussed to resolve all categorization and annotation disagreement. As typical for a survey, our sample may suffer from selection bias. To mitigate this risk, we randomly selected the survey candidates and designed our survey to be anonymous and short in order to encourage responses. Moreover, as we chose survey participants from contributors and maintainers who were actively using templates, our findings might be biased towards the perceived benefits of templates. We advocate for future research on surveying OSS practitioners who have never used templates about what barriers prevented them from using templates.

***External Validity.*** As our research was conducted looking at only a set of public, popular projects in one OSS platform (*i.e.*, GitHub), the results may or may not generalize to other OSS projects, or other OSS platforms (*e.g.*, GitLab). Although we have tried to mitigate this threat by using a diverse set of projects with different domains and programming languages, the dataset is small and not entirely representative of all OSS projects. Future work can replicate our study on more other OSS projects and OSS platforms.

## 9 Conclusion

In this paper, we conducted an empirical study of the *issue/PR template* feature on GitHub. By performing qualitative and quantitative analyses on a set of top popular OSS projects and conducting surveys with OSS practitioners, we investigated the usage, impacts, and practitioners' perceptions of this feature. We found that templates generally include a wide variety of elements to greet contributors, explain project guidelines, and collect relevant information. We also observed that the adoption of templates has impacts on the submissions of both issues and PRs and the discussions and resolution of issues. Although most survey respondents considered templates useful in supporting a variety of tasks, both contributors and maintainers reported challenges of using templates and suggested improvements to enhance the template feature. Finally, we proposed implications and suggestions for OSS practitioners and tool designers.

## Acknowledgments

We would like to thank all our survey participants for their time and insights. This work was supported by National Grand R&D Plan (Grant No. 2020AAA0103504) and the Major Key Project of PCL.

## References

[1] About github draft pull request. https://github.blog/2019-02-14-introducing-draft-pull-requests/. Accessed: 2022-03-28.

[2] About github issue and pull request templates. https://github.blog/2016-02-17-issue-and-pull-request-templates/. Accessed: 2022-03-28.

[3] About github rest api. https://docs.github.com/en/rest. Accessed: 2022-03-28.

[4] About slack. https://www.slack.com/. Accessed: 2022-03-25.

[5] Creating issue forms on github. https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests/configuring-issue-templates-for-your-repository\#creating-issue-forms. Accessed: 2022-03-28.

[6] An open letter to github from oss maintainers. https://github.com/dear-github/dear-github/. Accessed: 2022-10-06.

[7] A replication package for our study. https://github.com/whystar/GH_Templates. Accessed: 2022-10-06.

[8] Stack overflow. http://stackoverflow.com. Accessed: 2022-03-25.

[9] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 35th International Conference on Software Engineering*, pages 712–721. IEEE Press, 2013.

[10] Olga Baysal, Reid Holmes, and Michael W Godfrey. No issue left behind: Reducing information overload in issue tracking. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 666–677, 2014.

[11] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 308–318, 2008.

[12] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344. IEEE, 2016.

[13] Hudson Borges and Marco Tulio Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018.

[14] Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310, 2010.

[15] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological Methods & Research*, 42(3):294–320, 2013.

[16] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012.

[17] Scott Chacon and Ben Straub. *Pro Git (Second Edition)*. Apress, 2018.

[18] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 396–407, 2017.

[19] An Ran Chen, Tse-Hsun Chen, and Shaowei Wang. Demystifying the challenges and benefits of analyzing user-reported logs in bug reports. *Empirical Software Engineering*, 26(8), 2021.

[20] Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology Press, 2014.

[21] Thomas D Cook, Donald Thomas Campbell, and Arles Day. *Quasi-experimentation: Design & analysis issues for field settings*, volume 351. Houghton Mifflin Boston, 1979.

[22] Steven Davies and Marc Roper. What's in a bug report? In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014.

[23] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. Confusion in code reviews: Reasons, impacts, and coping strategies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*, pages 49–60. IEEE, 2019.

[24] Omar Elazhary, Margaret-Anne Storey, Neil Ernst, and Andy Zaidman. Do as i do, not as i say: Do contribution guidelines match the github contribution process? In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 286–290. IEEE, 2019.

[25] Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. Beyond the code itself: how programmers really look at pull requests. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 51–60. IEEE, 2019.

[26] Patricia I Fusch and Lawrence R Ness. Are we there yet? data saturation in qualitative research. *The qualitative report*, 20(9):1408, 2015.

[27] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. The shifting sands of motivation: Revisiting what drives contributors in open source. *arXiv preprint arXiv:2101.10291*, 2021.

[28] Georgios Gousios, Martin Pinzger, and Arie Van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. ACM, 2014.

[29] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, pages 285–296. IEEE, 2016.

[30] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering*, pages 358–368. IEEE, 2015.

[31] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, pages 72–81. ACM, 2004.

[32] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie Van Deursen. Communication in open source software development mailing lists. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 277–286. IEEE, 2013.

[33] Guido W Imbens and Thomas Lemieux. Regression discontinuity designs: A guide to practice. *Journal of econometrics*, 142(2):615–635, 2008.

[34] Rahul N Iyer, S Alex Yun, Meiyappan Nagappan, and Jesse Hoey. Effects of personality traits on pull request acceptance. *IEEE Transactions on Software Engineering*, 2019.

[35] David Kavaler, Sasha Sirovica, Vincent Hellendoorn, Raul Aranovich, and Vladimir Filkov. Perceived language complexity in github issue discussions and their effect on issue resolution. In *Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 72–83, 2017.

[36] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. 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*, pages 187–197. IEEE Press, 2017.

[37] Renee Li, Pavitthra Pandurangan, Hana Frluckaj, and Laura Dabbish. Code of conduct conversations in open source software projects on github. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–31, 2021.

[38] Zhixing Li, Yue Yu, Tao Wang, Shanshan Li, and Huaimin Wang. Opportunities and challenges in repeated revisions to pull-requests: An empirical study. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–35, 2022.

[39] Zhixing Li, Yue Yu, Tao Wang, Gang Yin, Shanshan Li, and Huaimin Wang. Are you still working on this? an empirical study on pull request abandonment. *IEEE Transactions on Software Engineering*, pages 1–1, 2021.

[40] Zhixing Li, Yue Yu, Minghui Zhou, Tao Wang, Gang Yin, Long Lan, and Huaimin Wang. Redundancy, context, and preference: An empirical study of duplicate pull requests in oss projects. *IEEE Transactions on Software Engineering*, 2020.

[41] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 176–188. IEEE, 2019.

[42] Chandra Maddila, Chetan Bansal, and Nachiappan Nagappan. Predicting pull request completion time: a case study on large scale cloud services. In *Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 874–882, 2019.

[43] Audris Mockus, Roy T Fielding, and James Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 22nd international conference on Software engineering*, pages 263–272, 2000.

[44] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Auto-completing bug reports for android applications. In *Proceedings of the 2015 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software EngineeringNovember*, pages 673–686, 2015.

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

[46] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 76–85. ACM, 2002.

[47] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. ACM, 2012.

[48] Cassandra Overney, Jens Meinicke, Christian Kästner, and Bogdan Vasilescu. How to not get rich: An empirical study of donations in open source. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1209–1221, 2020.

[49] Sebastiano Panichella, Gerardo Canfora, and Andrea Di Sorbo. "won't we fix this issue?" qualitative characterization and automated identification of wontfix issues on github. *Information and Software Technology*, 139:106665, 2021.

[50] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. Information needs in contemporary code review. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–27, 2018.

[51] Gustavo Pinto, Igor Steinmacher, and Marco Aurélio Gerosa. More common than you think: An in-depth study of casual contributors. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering*, pages 112–123. IEEE, 2016.

[52] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. Categorizing the content of github readme files. *Empirical Software Engineering*, 24(3):1296–1327, 2019.

[53] Achyudh Ram, Anand Ashok Sawant, Marco Castelluccio, and Alberto Bacchelli. What makes a code change easier to review: an empirical investigation on code change reviewability. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 201–212, 2018.

[54] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. Relationship between geographical location and evaluation of developer contributions in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–8, 2018.

[55] Eric Raymond. The cathedral and the bazaar. *Knowledge Technology & Policy*, 12(3):23–49, 1999.

[56] Yang Song and Oscar Chaparro. Bee: A tool for structuring and analyzing bug reports. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software EngineeringNovember*, pages 1551–1555, 2020.

[57] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. Overcoming open source project entry barriers with a portal for newcomers. In *Proceedings of the 38th International Conference on Software Engineering*, pages 273–284. ACM, 2016.

[58] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurélio Gerosa. Almost there: A study on quasi-contributors in open-source software projects. In *2018 IEEE/ACM 40th International Conference on Software Engineering*, pages 256–266. IEEE, 2018.

[59] Xin Tan and Minghui Zhou. How to communicate when submitting patches: An empirical study of the linux kernel. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–26, 2019.

[60] Parastou Tourani, Bram Adams, and Alexander Serebrenik. Code of conduct in open source projects. In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*, pages 24–33. IEEE, 2017.

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

[62] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering*, pages 356–366. ACM, 2014.

[63] Jason Tsay, Laura Dabbish, and James Herbsleb. Let's talk about it: evaluating contributions through discussion in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 144–154. ACM, 2014.

[64] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. The sky is not the limit: multitasking across github projects. In *Proceedings of the 38th International Conference on Software Engineering*, pages 994–1005. IEEE, 2016.

[65] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 805–816. ACM, 2015.

[66] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW), November 2018.

[67] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, 59(8):080104, 2016.

[68] Mengxi Zhang, Huaxiao Liu, Chunyang Chen, Yuzhou Liu, and Shuotong Bai. Consistent or not? an investigation of using pull request template in github. *Information and Software Technology*, page 106797, 2021.

[69] Xunhui Zhang, Tao Wang, Yue Yu, Qiubing Zeng, Zhixing Li, and Huaimin Wang. Who, what, why and how? towards the monetary incentive in crowd collaboration: A case study of github's sponsor mechanism. *arXiv preprint arXiv:2111.13323*, 2021.

[70] Xunhui Zhang, Yue Yu, Tao Wang, Ayushi Rastogi, and Huaimin Wang. Pull request latency explained: An empirical overview. *arXiv preprint arXiv:2108.09946*, 2021.

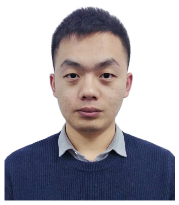
[71] Zheying Zhang, Outi Sievi-Korte, Ulla-Talvikki Virta, Hannu-Matti Järvinen, and Davide Taibi. An investigation on the availability of contribution information in open-source projects. In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 86–90. IEEE, 2021.

[72] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 60–71. IEEE, 2017.

[73] Shurui Zhou, Stefan Stanciulescu, Olaf Leßenich, Yingfei Xiong, Andrzej Wasowski, and Christian Kästner. Identifying features in forks. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 105–116. IEEE, 2018.

[74] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. What the fork: a study of inefficient and efficient forking practices in social coding. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 350–361. ACM, 2019.

[75] Thomas Zimmermann. Card-sorting: From text to themes. In *Perspectives on Data Science for Software Engineering*, pages 137–141. Elsevier, 2016.



**Zhixing Li** is an assistant professor in the College of Computer at National University of Defense Technology (NUDT). He received his Ph.D. degree in Computer Science from NUDT in 2021. His research goals are centered around the idea of making the open source collaboration more efficient and effective by investigating the challenges faced by open source practitioners and designing smarter collaboration mechanisms and tools.

**Yue Yu** is an associate professor in the College of Computer at National University of Defense Technology (NUDT). He received his Ph.D. degree in Computer Science from NUDT in 2016. He has won Outstanding Ph.D. Thesis Award from Hunan Province. His research findings have been published on ICSE, FSE, ASE, TSE, MSR, IST, ICSME, ICDM and ESEM. His current research interests include software engineering, data mining and computer-supported cooperative work.

**Tao Wang** is an associate professor in the College of Computer at National University of Defense Technology (NUDT). He received his Ph.D. degree in Computer Science from NUDT in 2015. His work interests include open source software engineering, machine learning, data mining, and knowledge discovering in open source software.

**Yan Lei** received the BA, MA, and Ph.D. degrees in computer science and technology, all from the National University of Defense Technology, China. He is currently an Associate Professor at the School of Big Data and Software Engineering, Chongqing University, China. His research interests include intelligent software engineering, fault localization, and program repair.

**Ying Wang** received her doctoral degree in software engineering from Northeastern University, China, in 2019. She is currently an associate professor at the Software College, Northeastern University, China. Her research interests include program analysis, dependency management, and software ecosystems. Her research work has been regularly published in top conferences and journals in the research communities of software engineering, including ICSE, ESEC/FSE, and TSE and has received ICSE 2021 ACM SIGSOFT Distinguished Paper Award. She received Outstanding Doctoral Dissertation Award of Liaoning province (2021), and Nominees Award for Outstanding Doctoral Dissertation of China Computer Federation (CCF) in 2020. She joined Microsoft Research Asia StarTrack Program (2020). More information about her can be found at: https://wangying-neu.github.io/.

**Huaimin Wang** received his Ph.D. in Computer Science from National University of Defense Technology (NUDT) in 1992. He has been awarded the “Chang Jiang Scholars Program” professor and the Distinct Young Scholar, *etc*. He has published more than 100 research papers in peer-reviewed international conferences and journals. His current research interests include middleware, software agent, and trustworthy computing.