Inside Bug Report Templates: An Empirical Study on Bug Report Templates in Open-Source Software

Junwei Zhang*
Zhejiang University, China

Zhongxin Liu*
Zhejiang University, China

Lingfeng Bao†
Zhejiang University, China

Zhenchang Xing
CSIRO's Data61 & Australian National University, Australia

Xing Hu
School of Software Technology, Zhejiang University, China

Xin Xia
Zhejiang University, China

ABSTRACT

In open-source software development, bug report templates (BRTs) have emerged as a crucial tool for ensuring the quality of bug reports. Despite their widespread use, developers have little knowledge about designing personalized BRTs. Therefore, it is necessary to understand the usage, effects, and design guidelines of BRTs. To this end, we conduct the first and most detailed study of BRTs on GitHub by performing quantitative and qualitative analyses in 3,194 projects and 5,987 commit messages of BRTs. We find that BRTs are widely used by open-source projects, especially prevalent in platform-type projects. Adopting BRTs can reduce the average number of comments and increase the likelihood of bug reports being addressed. Additionally, they may help developers identify duplicate reports and bug reports with missing description elements. We also classify the change history of existing BRTs and propose 14 design guidelines for BRTs. We survey 20 developers and 19 reporters on GitHub to investigate practitioners' perceptions of BRTs. The majority of respondents acknowledge the importance of BRTs. Based on our findings, we highlight future research directions and provide actionable suggestions for practitioners.

CCS CONCEPTS

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

KEYWORDS

Bug Report Template, Empirical Study, Open Source Software

ACM Reference Format:
Junwei Zhang, Zhongxin Liu, Lingfeng Bao, Zhenchang Xing, Xing Hu, and Xin Xia. 2024. Inside Bug Report Templates: An Empirical Study on Bug Report Templates in Open-Source Software. In *15th Asia-Pacific Symposium on Internetware (Internetware 2024), July 24–26, 2024, Macau, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3671016.3671401

*Both authors contributed equally to this research.
†Lingfeng Bao is the corresponding author.

Internetware 2024, July 24–26, 2024, Macau, China

ACM ISBN 979-8-4007-0705-6/24/07
https://doi.org/10.1145/3671016.3671401

1 INTRODUCTION

Bug reports are instrumental in guiding developers to identify, understand, and rectify software bugs [25, 30]. These detailed reports provide insights into the problem and its location within the codebase, facilitating accurate bug resolution [8, 49]. However, the quality of bug reports can be inconsistent, with many lacking crucial details or containing inaccurate information [23, 27]. Developers often invest additional time understanding these reports, seeking clarifications, or filtering through redundant information [7, 28]. Furthermore, poor-quality reports can lead to the submission of duplicate bugs, as they hinder the identification of similar existing bugs [27]. This highlights the need to improve the quality of bug reports to streamline the debugging process and improve software reliability.

To address the challenges in the quality of bug reports, various tools have been proposed to aid reporters in writing effective bug reports [45, 49]. For instance, Bettenburg et al. [5] proposed a tool to measure the quality of bug reports based on their content. Song et al. [39] designed BEE, a tool capable of identifying the structure of bug descriptions and determining the presence of essential fields. Chaparro et al. [33, 49] employed regular expressions, natural language processing (NLP), and machine learning (ML) techniques to pinpoint missing details in bug descriptions. While these tools promise to elevate the quality of bug reports, their adoption in real-world scenarios remains limited. Currently, the prevalent and most effective strategy to reduce low-quality bug reports is the use of bug report templates (BRTs). BRTs, primarily designed for reporters to fill out, are extensively embraced in open-source initiatives and serve as a structured guide for reporters and developers. Their use can bolster the efficiency of both parties [46]. There is an evident need to guide practitioners in creating and maintaining BRTs.

In this paper, we bridge the existing research gap by conducting a large-scale study to understand the impact of BRTs and derive their design guidelines. We employ both quantitative (mining projects using BRTs and their change history) and qualitative methods (a survey). **The quantitative method** aims to understand how BRTs influence bug reports and establish guidelines for designing BRTs. Specifically, we first collect a list of GitHub projects, categorize them based on the number of stars and project type, and analyze the distribution of BRTs across project types and popularity levels. Then, we identify the projects with BRTs (hereafter, BRT projects) from our initially collected projects and adopt four indicators of bug report quality to demonstrate the effect of BRTs. Finally, we collect the commit messages of the commits that change BRTs (hereon, BRT commit) from the BRT projects and develop a hierarchical

taxonomy of BRT changes based on the quality attribute targeted by each BRT change. According to the taxonomy, we formulate a set of design guidelines. The purpose of **the qualitative method** is to gain insight into developers' and reporters' perspectives, which can guide practitioners in tailoring BRTs for their projects and help researchers devise advanced techniques for improving bug report quality. We conduct a survey with 200 developers and 200 reporters on GitHub and create an anonymous survey capturing demographic details, scoring on BRT usage, BRT quality, and design guidelines. Then, we adopt the non-probabilistic sampling method [31] to select developer and reporter participants from BRT projects and send the questionnaire to them. Finally, we analyze the survey results based on the type of question.

From our research on GitHub, we gathered data from a significant pool of projects. Specifically, we compiled a list of **3,194** projects, categorized by their number of stars. Out of these, **1,506** projects were identified as using BRTs. We traced **5,987** commits that entailed changes to BRTs. These BRT modifications were systematically grouped into 19 distinct categories. Furthermore, our survey targeted the GitHub community and garnered feedback from 20 developers and 19 reporters, all experienced with BRTs. Based on the collected BRTs, their change history, and the survey, we focus our study on the following research questions:

RQ1: Can BRTs improve the quality of bug reports? We find that adopting BRTs can improve the ratio of fixed bug reports and reduce the average number of comments. Meanwhile, BRTs may help increase the ratio of the bug reports identified as duplicates and the percentage of the bug reports with missing description elements. These effects are consistently observed across different project types. Interestingly, projects with lower popularity see a marked improvement in bug report quality upon adopting BRTs.

RQ2: What are the guidelines for designing BRTs? We construct a hierarchical taxonomy of BRT changes. This taxonomy covers 19 subcategories that belong to six quality attributes related to the contents in BRTs (eight subcategories) and the organization of BRTs' contents (eleven subcategories). Drawing from our classification insights, we have distilled 14 design guidelines to enhance BRT creation and usage.

RQ3: What design guidelines are perceived as applicable by practitioners? Most respondents agree that BRTs are instrumental in pinpointing duplicate bug reports and those with incomplete descriptions. About 52.6% of developers and 47.4% of reporters consider up-to-dateness as the most important metric. A considerable 73.7% of developers and 47.4% of reporters believe that "Provide concise field description" is the most important guideline for all respondents.

Given our observation that changes related to BRT readability are prevalent, developers should prioritize enhancing the readability of BRTs. In addition, we recommend developing tools that can automatically synchronize version numbers in BRTs with project version releases. In general, this study offers both practical and theoretical insights, enriching the existing literature on understanding and crafting bug report templates. Further contributions include:

(1) We deepen the understanding of BRTs by investigating their distribution in different types of projects, the relation between BRTs and projects' popularity, and the common fields appearing in BRTs.

(2) We propose four indicators to judge the effect of BRTs on bug reports and explore how BRTs affect the quality of bug reports.

(3) We propose a taxonomy of BRT changes and a set of BRT design guidelines to help developers design BRTs.

(4) We release our replication package [35] for future works.

2 DATA COLLECTION AND ANALYSIS

2.1 Data Collection

2.1.1 Project selection. Given that GitHub hosts more than 200 million projects [21, 22], analyzing every project would be impractical and time-consuming. To address this, our approach aligns with methodologies adopted in prior related research [29, 34, 47, 50]. Specifically, we use the Restful API [15] of GitHub to collect a list of projects with more than 100 stars. Then, we divided the collected projects into six groups based on the number of stars: (100k, +∞), (50k, 100k), (10k, 50k), (5k, 10k), (1k, 5k), and (100, 1k). For each group, we select the top 1000 most starred projects. We ignore the forked projects to avoid including the same project multiple times during analyzing. This data acquisition method could reflect the mechanism of BRTs on GitHub.

To safeguard the quality of the dataset, we exclude the following projects from the initial dataset: (1) Projects that reported no issues or had fewer than 10 reported issues; (2) Projects with content not predominantly in English; (3) Projects labeled as "Achieved", indicating they are no longer active or maintained; (4) Non-software-centric projects, such as collections of study materials or tutorials. We identified these projects by searching for specific keywords like "material", "tutorial", and others (complete list available in our replication package [35]). We follow the previous work to determine the type of projects from their description information [24, 36] and manually check the annotated results. 3,194 OSS projects remained after filtering the above conditions. We refer to these projects as **initial projects**.

2.1.2 BRTs information selection. From our initial projects, not all were suitable for our experiments due to the absence of BRTs. To identify projects with BRTs, we undertook the following steps: (1) We automatically verified the presence of BRTs in each project. (2) We crawled the project directories to extract template files located in the *".github/ISSUES_TEMPLATES"* folder. Further, we found that some projects do not place the templates in the specified folder. We manually checked such projects to ensure all templates could be obtained. Finally, 1,506 projects with BRTs remained. We refer to these projects as **BRT projects**.

Afterward, we collect more detailed information about BRT projects for further analysis to identify the relationship between the BRTs and the experimental indicators. Specifically, we first collect the bug reports of each project and filter out reports that have not been closed because they are not addressed and may not have enough information for our analysis [46]. Next, for each bug report, we collect its creation time, finished time, comment content, and the number of comments. Then, we collect BRT projects' commits related to BRT (referring to the change history) using Pydriller [40]. In total, there are 5,987 commits, which can be found in our replication package [35].

2.2 The Usage Analysis of BRTs

2.2.1 The distribution of BRTs in different types of projects. We focus on some types of projects with the largest number of GitHub projects [38], including the platform, plugin, database, framework, toolkit, library, and programming language. The percentage of projects with BRTs varies across different types of projects, with overall usage averaging about 60%. Platform-type project has the highest percentage of BRTs, while plugin-type projects have the lowest proportion. We think the reason is that the plugin projects are usually relatively small, so developers may feel that BRTs are unnecessary. Platform projects generally have larger communities with more participants and bug reports, necessitating templates to ensure consistent bug report formats.

2.2.2 The relationship between the BRT mechanism and the project popularity. The number of projects containing BRTs increases as their popularity rises. In the (100k, +∞) interval, there are ten projects with BRTs, accounting for about 84.46%. This interval has the highest template usage. It seems that whether a project uses BRTs is correlated with its popularity. To validate this assumption, we adopt the Spearman correlation to perform a correlation analysis [41, 46]. The result (correlation coefficient: 0.393, p−value<0.001) indicates that the factors related to a project's BRTs are complex.

2.2.3 The common fields appearing in BRTs. We summarize multiple fields that BRTs typically contain and calculate the percentage of BRTs for each field. In order, the top five fields with the most occurrences (from a total of 14 fields) are reported as:

- **Version** (83.13%): It specifies the version of the current or related projects the user uses when an issue arises. This is the most frequently occurring field in BRTs.
- **Reproduction** (82.27%): It provides detailed steps to replicate the reported issues. This field also frequently occurs in BRTs.
- **Expected behavior** (71.38%): It describes the expected outcome or behavior from a certain action or set of actions. This often contrasts with what is reported in an "Observed Behavior" field (if present).
- **Environment** (49.73%): It outlines the necessary information for the project's execution, such as the platform, browser, or module version. Many BRTs incorporate this field, offering essential context for developers.
- **Description** (49.60%): It offers a concise introduction or overview of the reported bug. This field is present in nearly half of the templates, serving as a quick reference point for developers.

We find that only 3.45% of projects have adopted all five of the most frequent fields mentioned above. It seems that only a few project developers are concerned about BRTs. They are unclear whether BRTs affect the quality of bug reports and whether the existing BRT is suitable for their project.

3 RQ1: CAN BRTS AFFECT THE QUALITY OF BUG REPORTS?

In this RQ, we investigated changes of bug report quality from four aspects and conducted an in-depth analysis based on the collected dataset.

3.1 Research Methodology

Referring to prior studies [23, 46], we demonstrated the impact of BRTs on bug report quality from four aspects: **(I)** the proportion of the duplicated bug reports that have been recognized, **(II)** the percentage of bug reports where missing elements were identified, **(III)** the number of comments, and **(IV)** the ratio of fixed bug reports. According to Zhang et al.'s investigation [46] on Quora [37] and Stack Overflow [43], these four metrics are the most frequently asked questions by developers about managing GitHub. Further, we randomly chose 100 projects (confidence level: 90%, margin of error: 0.205% [4]) from the BRT projects. For the collected bug reports, we divided them into two groups based on the time of adopting BRTs: the bug reports before and after adopting BRTs, namely the before group and the after group.

The percentage of duplicated bug reports. It is difficult to identify duplicated reports from numerous reports because users may use different descriptions and have different understandings of the same problem. Since developers will propose similar solutions to duplicated issues, the duplicated bug reports annotated by developers are more accurate and objective. Therefore, we use the proportion of identified duplicated reports before and after introducing the BRTs as an evaluation metric. There are two cases to determine which bug report is duplicated. One is whether it has been marked with the label of "Duplicate/Repeated" by developers. The other learns whether the review comments have related statements, such as "Duplicate of #ID". As for the second case, we first read and analyzed several projects' bug reports to understand how developers mentioned a bug report is duplicated in comments. Then, we use specific strings and regular expressions to detect whether the bug reports are duplicated. To identify the strings and design the regular expressions, we study 1,000 bug reports randomly selected from the BRT projects. We collect a pool of strings commonly used to refer to different bug report elements until we do not find any new keys in the context of the bug reports.

The percentage of the bug reports with missing elements. To explore whether BRTs can help reporters provide the necessary information, we discuss the impact of BRTs on bug report quality through this aspect. Similar to the percentage of identified duplicated bug reports, we utilize the percentage of the bug reports identified by developers as missing element descriptions as one of our evaluation metrics. Specifically, we summarize related keywords (e.g., "lack of expected behavior" and "provide steps to reproduction") from the BRT projects to construct the pool of keywords.

The ratio of the bug reports with fixed status. We determine the final status of each closed bug report by matching keywords in their comments [20, 25]. Bug reports with comments matching the keywords "resolve", "fix", and "implement" are regarded as fixed.

The number of comments. It refers to the number of conversations between developers and reporters in solving issues. Referring to previous work [8, 39], developers ask reporters to provide more specific information about the bug during the resolution process for low-quality bug reports. We investigate whether the introduction of BRTs could reduce the number of comments between developers and reporters. We count the number of comments for each bug report and average the number of comments for each project.

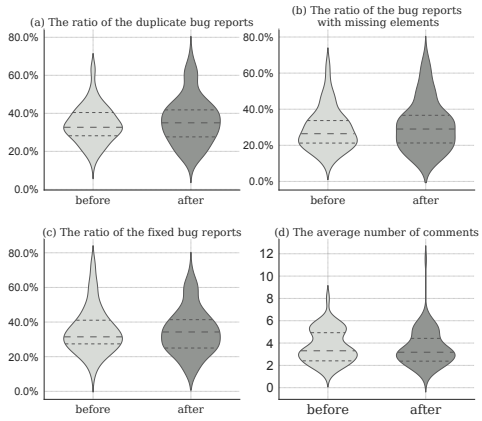

Figure 1: The effect of BRTs on the four aspects.

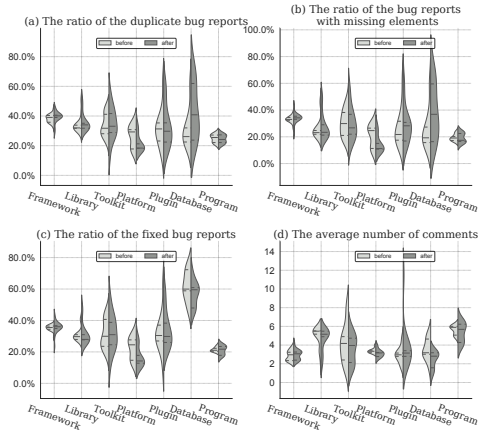

Figure 2: The effect of BRTs on four indicators regarding different project types.

3.2 Research Results

Figure. 1 depicts the comparison results of the four aspects before and after adopting BRTs. **(I)** We can see from the plot that the bug reports with BRTs references may peak in areas of high duplicated rates. The median line (35%) of the after group is higher than that (32%) of the before group. We further adopt the Wilcoxon test to validate whether there is a significant difference between the two groups in this indicator. The result (p−value:0.0456) suggests a significant difference. **(II)** For the ratio of bug reports with missing element descriptions, the median ratio (29%) of the before group is lower than that of the after one (31%). Meanwhile, the number of projects with a low missing ratio in the before group is smaller than that in the after group. The result of the variance test (p−value:0.040) suggests a statistically significant difference between the median missing ratios of the two groups. **(III)** Regarding the ratio of fixed bug reports, the median line (29%) of the after group is higher than that (26%) of the before group. The Wilcoxon result (p−value:0.323) suggests that the difference between the two groups in the ratio of fixed bug reports is significant. **(IV)** We find that the average number of comments (3.54) in the after group is lower than that in the before group (3.72). Further, the result of the Wilcoxon test (p−value:0.028) suggests that a statistically significant difference between the two.

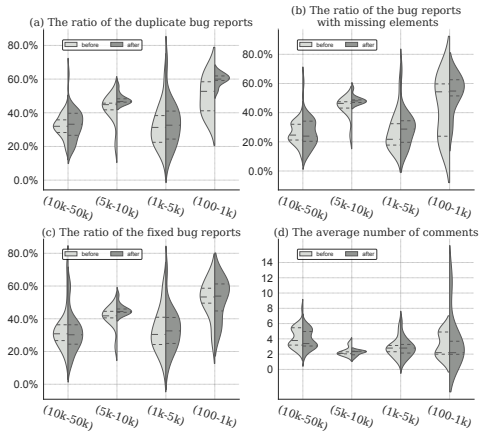

Figure 3: The effect of BRTs on four indicators regarding different star intervals.

We also investigate the effect of BRTs on four indicators across different types of projects. The detailed results are in Figure. 2. We can see that the distribution of four indicators varies widely among different types of projects. **(I)** According to Figure. 2(a), for most types of projects, the median line of the after group is higher than that of the before group. **(II)** Figure. 2(b) shows that for the framework, library, plugin, and database projects, the identified percentage of the bug reports with missing elements increases after adopting BRTs. For other types of projects, the difference between the two groups in this indicator is not significant. **(III)** Figure. 2(c) shows that the two groups differed significantly in the ratio of the fixed reports for three types of projects (i.e., platform, plugin, and database). **(IV)** Figure. 2(d) presents that the average number of comments decreases after adopting BRTs in the toolkit and database-type projects.

The effect of BRTs on projects with different popularity is presented in Figure. 3. **(I)** Figure. 3(a) shows that projects with the star intervals of (5k, 10k) and (100, 1k) have large proportions of duplicated bug reports being identified. **(II)** As seen in Figure. 3(b), the identification ratio of the bug reports with missing elements increases for less-popular projects (intervals of (1k, 5k) and (100, 1k)) after adopting BRTs. **(III)** Introducing BRTs can help projects with varying popularity increase the probability of fixing an issue, especially those with lower popularity. **(IV)** Regarding the average number of comments, the number of bug report comments for projects in the intervals of (5k, 10k) and (100, 1k) is less than the number of bug report comments after introducing the BRTs.

4 RQ2: WHAT ARE THE GUIDELINES FOR DESIGNING BRTS?

In this section, we describe the research methodology and present the detailed results of design guidelines.

4.1 Research Methodology

To explore the design guidelines of BRTs, we first mined and analyzed commits containing BRT changes from the BRT projects. Then, we constructed a taxonomy of BRT changes to identify the shortcomings of existing BRTs in terms of design. To this end, we start with the software documentation quality attributes defined by prior work [1, 2]. In early studies [1, 2], there are 16 documentation

quality attributes related to software document quality. We classify each BRT change based on the quality attribute this change aims to improve. Our reason for this classification is that BRTs are a type of software documentation. Defining the taxonomy based on these quality attributes can help us unveil the reasons behind BRT changes. Therefore, the quality attributes of software documentation should also apply to BRTs. It is worth mentioning that our collected BRT changes only cover a subset of all the quality attributes defined by prior work [1, 2]. We followed the card sorting process [42] to identify which quality attributes each BRT change aims to improve, which has been proven effective in prior works [26]. We first created a card for each sampled BRT change. Then, we employed 20% of the cards to identify possible categories by reading their detailed changes and commit messages, and individually coding. Next, we summarized potential commonalities of the information types and discussed the contents of the BRT changes. Finally, we discussed our findings and disagreements. After this iteration, we independently labeled the remaining cards into two categories and six subcategories. We adopted Cohen's Kappa coefficient to measure the agreement between the two annotators [9]. Their Kappa value is 0.941, indicating substantial agreement between the two annotators. Finally, based on the taxonomy of BRT changes, we distilled a series of actionable guidelines to help developers design BRTs.

4.2 Research Results

From the above labeling process, we obtained **5,987** labeled BRT changes in the BRT dataset. In this subsection, we present the details of our taxonomy.

4.2.1 The content of BRTs (What). This category refers to the changes aiming to improve the contents of the BRTs (i.e., what should be written in the BRTs), which is the predominant category in the taxonomy. It relates to three BRT quality attributes: **Correctness** (e.g., erroneous spelling), **Completeness** (e.g., missing the field of steps to reproduce), and **Up-to-dateness** (e.g., version number varies with the project update). A total of **3,081** BRT changes are classified into this group, including eight types of BRT changes and accounting for about 51.35% of the labeled BRT changes.

Correctness (1,400). Correct templates provide accurate information in accordance with facts [48]. Incorrect templates might have unforeseen consequences, such as wasting time and replicating bad examples. This quality attribute only includes one BRT change type: Typos fixes.

- **Typos fixes (1,400).** This type of BRT changes contains the most changes, accounting for 23.33%. Typical examples are the correction of spelling errors and the removal of spaces. For example, in the *astro* project [11], due to the correctness of the BRTs, the developers submitted 22 changes.

Completeness (1,264). If a BRT does not include necessary fields, the reporter would be continually asked to supplement more relevant information, increasing the communication time between developers and reporters. Hence, to ensure the completeness of the template, developers have made the following changes, which include three types of changes:

- **Create multiple templates based on different intentions (899).** The developer expands the original BRTs into multiple templates based on reporters' requirements, such as feature request template, help&question template, and document template. About 14.98% of the BRT changes belong to this type. The *dva* project [14] has the most types of BRTs (i.e., eight). The most common other type of BRTs is the feature request template, adopted by about 49.78% of the projects.
- **Add essential fields (339).** Developers can prompt the reporters to fill in essential information by adding the corresponding fields to the existing template. 239 projects contain this type of BRT changes. The *cli* project [13] contains the most BRT changes (seven changes) to add fields. Furthermore, the "Version" field is the most frequently added field, followed by the "Checklist" field. Besides, we also observed that many projects added multiple fields in a commit, with a maximum of six fields added simultaneously in the *redux* project [16]. It illustrates the imperfection in BRTs' initial design and the importance of some fields, such as "Version", "Checklist", "Code", and "S2R".
- **Modify the required limitation (26).** To ensure that bug reports include some necessary fields and avoid missing essential elements, some developers modify the submission requirements of fields when designing templates, requiring reporters to fill in relevant fields before submitting reports. This type contains 26 BRT changes, with "S2R" being the most frequently modified field.

Up-to-dateness (404). The template that is out of sync with the project would confuse reporters. Ensuring the up-to-dateness of BRTs helps reporters write better bug reports and make it easier for developers to locate bugs. There are two types of BRT changes aimed at improving BRTs quality in up-to-dateness.

- **Delete the old BRTs (194).** This type of BRT change involves deleting old BRTs in time, corresponding to 194 changes from 139 projects. These changes usually contained the creation of new BRTs. Duplicate templates caused trouble for reporters, especially those with the same content but in two places.
- **Modify the version information (210).** This type of BRT change aims to update the version numbers in BRTs, which keeps bug reports in sync with the project and improves the up-to-dateness. Discrepancies between the BRTs' and the project's versions can make the problem irreproducible and confuse developers. For example, in the *uikit* project [12], with the upgrade of the project version, the description information in the "Version" field has been modified ten times.

Internationalization (13). This quality attribute covers two types of BRT changes related to the translation process. The lack of translated documentation hinders the development of projects, especially when English BRTs are unavailable.

- **Modify the description of field (1).** Some developers translated the description information in the BRTs into other languages to realize the internationalization standard of the BRTs. For example, in the *vueComponent* project [10], developers added Chinese descriptions for all fields.

- **Add multiple language BRTs (12).** Several projects extended multiple language BRTs to attract more practitioners from different countries. For example, the *minikube* project adopted multilingual BRTs [17], including Arabic, Indonesian, Kurdish, Spanish, Portuguese, and Chinese. These BRTs allow more external users to submit bug reports.

Discussion and Implications. Our results highlight that developers may make changes to improve the Correctness, Up-to-dateness, and Internationalization of BRTs. Based on the above classification results, we summarized six design guidelines for BRTs, as follows:

1. Completeness

- **Guideline 1.1: Provide multiple templates based on different intentions.** Frequently used template types include feature request template, help&question template, and document template.
- **Guideline 1.2: Provide the essential field.** The template should contain some important fields, such as "Checklist", "Description", "S2R", "Expected behavior", and "Environment".
- **Guideline 1.3: Set the required limitation of some essential fields.** If some information is essential for bug fixing, developers can consider requesting reporters to fill in such information by setting the required fields.

2. Up-to-dateness

- **Guideline 2.1: Migrate old templates to a unified location instead of deleting them.** The developers should not directly delete older versions of templates as they may contain important information.
- **Guideline 2.2: Modify the version in time.** The developer should update the version numbers in BRTs as the project is upgraded.

3. Internationalization

- **Guideline 3.1: Provide multilingual BRTs or field descriptions.** The developer can consider translating their templates or field descriptions into other popular or common languages, such as Chinese and Spanish.

To guide developers on essential fields and template types to include in their projects, researchers can consider proposing tools to find suitable templates from similar projects and recommend them as a references. For the up-to-dateness of templates, we suggest that researchers propose a tool to update version numbers in BRTs as the project and related projects evolve.

4.2.2 The organization of BRTs (How). A total of **2,906** BRT changes are related to the information organization of BRTs. These subcategories capture BRT changes aiming to improve two documentation quality attributes: **Usability** (e.g., the organization and consistency of document content) and **Readability** (e.g., confusing template fields).

Usability (1,022). The usability of BRTs refers to the extent to which users can use them to achieve their objectives effectively. This quality attribute covers five types of BRT changes affecting users' experience with the BRTs to alleviate poor organization and findability problems.

- **Modify the format of BRTs (390).** 6.5% of the BRT changes are related to modifying the template format. There are multiple BRTs forms, including structured and unstructured BRTs. In structured BRTs, reporters cannot change the template's content. They can only fill in bug-related information in a fixed position, which guarantees the usability of the bug reports. This report format is convenient for developers to find the bug information. When using unstructured templates, reporters can arbitrarily modify the template's content, which is highly flexible. 117 projects are converting unstructured BRTs into structured BRTs. However, three projects are converting the structured BRTs to unstructured BRTs. We think the possible reason for this is that the structured templates limit the flexibility of reporters and may prevent them from submitting bug reports in some cases.
- **Switch the branch (50).** Some developers merge the bug report templates from a non-master branch to the master branch. This is one of the least frequent types, accounting for about 0.83%.
- **Modify the location of the BRTs (464).** 7.73% BRT changes move BRTs to standard locations, which helps developers locate and modify the templates and improve the usability of BRTs. We find that most projects place template files in the "*.github/issue_template/*" folder.
- **Modify the location of the field (95).** In addition to moving templates to standard locations, placing fields in fixed locations is an effective way to help developers find relevant information. 83 projects contain this type of BRT change, accounting for about 7.20%. Developers adjust the positions of elements in BRTs to ensure the intuitiveness and clarity of information organization. Placing some elements in standard locations helps developers quickly find the information and improves the review experience. The most frequently moved field is the "Version" field, usually placed in the middle of the BRTs. The second most frequent one is the checklist field, generally placed at the beginning of the BRTs.
- **Add the link in BRTs (23).** These BRT changes added the intra-documentation links in field descriptions, making navigation more convenient and ensuring BRT usability. We observed that developers are accustomed to adding links to project chat rooms.

Readability (1,884). It is the extent to which BRTs are easy to read. BRT changes related to improving clarity are divided into five subcategories. We describe them as follows.

- **Add description of field (688).** Some reporters do not know the meaning of the BRT fields when filling out bug reports, especially for first-time reporters. Hence, some developers added field descriptions to help reporters understand the meaning of the field and improve readability. 381 projects contain this type of BRT change. The "Checklist" field appears the most among these added descriptions, followed by "Version", "S2R", "Environment".
- **Remove the fields (124).** Developers in 101 projects removed redundant or duplicate fields in BRTs to improve the readability of the template (e.g., the *swoole-src* project [19]).
- **Remove the field descriptions (100).** Similar to removing fields, some developers removed field descriptions to improve readability.
- **Distinguish fields from field descriptions (52).** Developers optimized field descriptions by turning them gray to enhance

readability. For example, the *tools* project [18] distinguished fields and their descriptions in structured templates.
- **Add the emojis (24).** Some developers added emojis to BRTs to improve their readability (e.g., the *tools* project [18]).
- **Modify the description of field (896).** Developers rewrite unclear field descriptions to reduce the verbosity and noise.

Discussion and Implications. In addition to the information content, the organization of information also influences BRT quality. Developers can refer to the following design guidelines to improve the quality of BRTs:

1. Usability
- **Guideline 1.1: Provide the structured templates.** The structured template can make the structure more transparent, avoid missing information, and make it easier for developers to find information in bug reports.
- **Guideline 1.2: Place the template in a fixed position.** The BRTs are usually placed in the folder of "*.github/ISSUES_TEMPLATE*".
- **Guideline 1.3: Place the field in a fixed position.** The developers could place the "Checklist" at the beginning of the template content, followed by the "Description" and "S2R", and put the "Additional context" at the end.
- **Guideline 1.4: Provide the links related to the project.** The developers could put the link to project chat rooms or contribution guidelines in the description of the "Checklist" field.

2. Readability
- **Guideline 2.1: Provide a concise field description.** It can tell the reporters the meaning and the purpose of the field and help them better fill it in (e.g., the description of "S2R" is "1. Go to...; 2. Click on...").
- **Guideline 2.2: Do not provide verbose/noisy descriptions.** For example, the "Version" and "Environment" fields often contain overlapping bug information.
- **Guideline 2.3: Add emojis to the template.** It can distinguish the functions of different templates and different fields.

Developers should focus on the readability of BRTs. We observed that readability-related changes are the most common type. This indicates that template contents are sometimes not easy to understand for reporters, resulting in the templates not playing the expected role. Hence, we suggest developers refer to our design guidelines to improve template readability. Researchers could propose automatic approaches to identify redundant and repeated content in BRTs for different projects, helping developers create concise templates.

5 RQ3: WHAT DESIGN GUIDELINES ARE PERCEIVED AS APPLICABLE BY PRACTITIONERS?

In this section, we conducted an online survey to investigate practitioners' perceptions of BRTs, including the BRTs' usage, effect on bug reports, and design guidelines.

5.1 Research Methodology

Following previous studies [6, 31], we adopted an anonymous survey for personal opinion surveys. Our survey includes various types of questions, such as multiple-choice questions and free-text answer questions. We first piloted the preliminary survey with three Ph.D. candidates to ensure the length of the survey and the clarity of terms. Then, we made minor modifications based on the received feedback and produced a final version. Note that the collected responses from the pilot survey are excluded from the presented results.

The survey consists of four sections: **(I) Demographic information**, including geographical location, the number of years of respondents' experience in GitHub, types of projects they are currently involved in, and types of programming languages that they mainly use. **(II) Statement scoring about BRTs usage.** We provided four statements and asked participants to score them according to their opinions. Respondents assessed the importance of each statement on a 5-point Likert scale *(Strongly Disagree, Disagree, Neutral, Agree, and Strongly Agree)* and an additional option *(I Don't Know)* [3]. The "*I Don't Know*" option was provided in case some statements did not apply to respondents' experience or for those who did not understand the statement. **(III) Statement scoring about the quality and design guidelines of BRTs.** We provided a list of statements describing the importance of different quality evaluation criteria and design guidelines. Respondents ranked the importance of each statement on a 5-point Likert scale *(Not at all, Slightly, Moderately, Very, Extremely, and I Don't Know)*. Asking participants to read the complete list of BRT changes would be excessive and could significantly increase the survey abandonment rate. We grouped them that are very similar and shared the same parent category to reduce the number of questions surveyed referring to previous work [1, 2]. Specifically, we summarized the 17 subcategories in the original taxonomy into 12 options: six in the Information Content (What) category and six in the Information Content (How) category. **(IV) Rationale and suggestions.** Apart from scoring the statements, they can also provide rationale and suggestions for each statement.

We adopted Non-Probabilistic Sampling methods [31] to select 200 developers and 200 reporters from the candidate list. Developers in the candidate list submitted the BRT changes mentioned in Section 4. Reporters in the candidate list submitted bug reports on projects in our dataset. We sent emails with a survey link to 200 developers and 200 reporters from GitHub projects. We assigned different questionnaires to developers and reporters. For developers, since the respondents who submitted template modification had a deep understanding of BRTs, we immediately investigated which guidelines affect the template's design. For reporters, we asked them whether they had submitted a bug report. If they submitted a report but did not adopt the template, we would ask them why they did not report using the template. If the respondents used the template information, we would ask them to judge whether the rules we summarized were acceptable to improve the quality of the template.

We analyzed the survey results based on the question types. For multiple-choice questions, we reported the number of each option selected. In terms of open-ended questions, we followed an inductive approach: two authors separately performed open card sorting and regularly discussed emerging themes until reaching an agreement. The complete questionnaire is available online in our replication package [35].

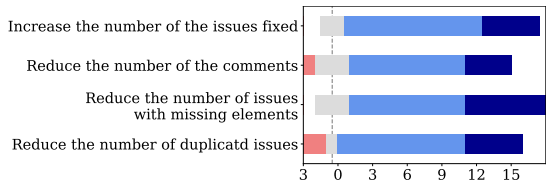

Figure 4: The effect of BRTs (Developer).

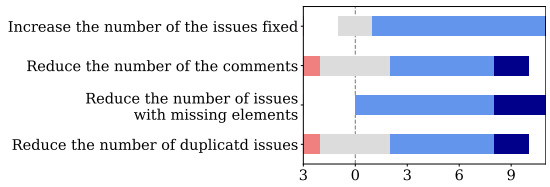

Figure 5: The effect of BRTs (Reporter).

5.2 Research Results

5.2.1 Demographic information. We received 39 valid answers (incomplete surveys were discarded). These responses have 19 reporters and 20 developers from four countries. Most respondents (22) have 1~5 years of professional experience. Our survey respondents are distributed across different types of projects, with frameworks being the most common project type.

5.2.2 Statement scoring about BRT usage. Most developers (75%) have submitted bug reports fewer than five times since using GitHub. 35% or 55% respondents agree or strongly agree that the guidelines are important to GitHub. 45% strongly agree that projects with different popularity have different BRT requirements, while 15% are neutral. 90% of developers believe that projects with different types have different template design requirements.

For reporters, more than 90% believe that guidelines are important to the open-source community. 91.7% believe that different popularity and types of projects should have different templates. This illustrates the importance and specificity of templates.

5.2.3 Statement scoring about the different effects of BRTs on bug reports. As shown in Figure. 4, 50% of developers agree, and 35% strongly agree that BRTs can reduce the ratio of bugs with missing description elements. 60% agree, and 25% strongly agree, that increasing the ratio of fixed bug reports is the most critical impact. These findings are consistent with our data analysis results in RQ2. The second most important impact is reducing the ratio of duplicate bug reports (50% agree and 20% strongly agree).

Figure. 5 indicates that almost all reporter respondents (eight respondents agree and five respondents strongly agree) believe that BRTs can reduce the ratio of bug reports with missing description elements. The second important effect is increasing the fixed-bug reports ratio (more than 76.9% of reporters agree). One respondent disagrees that the BRTs can reduce the number of comments and the ratio of duplicate bug reports.

5.2.4 Statement scoring about the quality of BRTs. As shown in Figure. 6, 50% and 35% of developer respondents consider that readability is very or extremely important, which is the most important quality attribute. Eight developers respectively rate template completeness as very or extremely important, which is the second most important one. On the other hand, two respondents rate up-to-dateness and internationality as slightly important, and half rate internationality as moderately important.

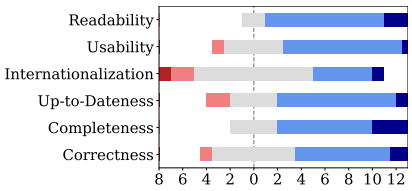

Figure 6: The importance of factors affecting BRTs quality (Developer).

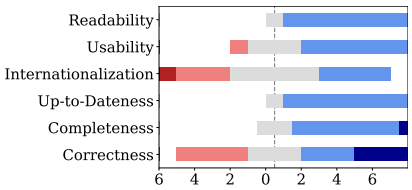

Figure 7: The importance of factors affecting BRTs quality (Reporter).

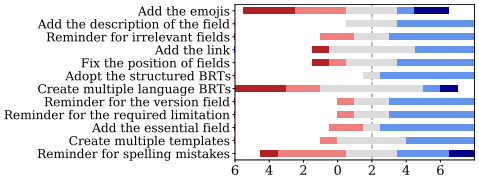

Figure 8: The importance of different guidelines affecting BRTs quality (Developer).

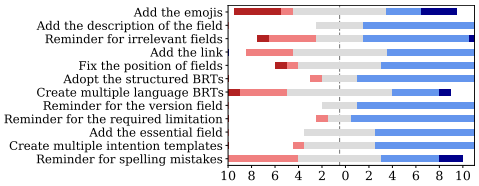

Figure 9: The importance of different guidelines affecting BRTs quality (Reporter).

For reporters, eight respondents and four respondents respectively indicated that readability was very or extremely important to the quality of the template. As shown in Figure 7, about 69.2% and 23.1% of reporters rate up-to-dateness as very or extremely important. These two factors are the most important factors for reporters. Besides, four respondents considered that correctness to be slightly important to the quality of the template. Around 23.1% and 7.7% of reporters believed internationality was slightly or not important to the quality of templates.

5.2.5 Statement scoring about the design guidelines of BRTs. Figure. 8 shows the importance of different guidelines affecting BRTs quality from the developers' perspective. Developers indicated that "Add field descriptions" is the most important factor (75%), followed by "Requirement constraints for adding necessary fields" (60%), "Modify version fields in templates" (55%) and "Adopt structured templates" (55%). As for other design guidelines, "Add emojis", "Add link", and "Create multiple language BRTs" are not concerns by developers. Figure. 9 shows the importance of different guidelines

affecting BRTs quality from the reporters' perspective. 69.2% of respondents thought "Have field descriptions" was very important. 61.53% respondents indicated that "Adopt structured templates" and "Have required restrictions for fields" were very important. Other guidelines deemed very important were "Have multiple templates based on different intents", "Have Links", and "Have required fields". Besides, we found that "Have emojis" and "Have multiple language BRTs" were considered slightly or not important by reporters.

5.2.6 Rationale and suggestions. In addition, six reporters did not refer to the template when completing the report since they joined GitHub. They mentioned that the main reason was not noticing the templates provided by the developers. Other reporters mentioned that using templates is cumbersome. Some developers noted, *"Many times the user ignores the template, even though the instructions state to follow it."*.

6 RELATED WORK

Bug reports are documents detailing software anomalies. Reporters typically craft these reports to communicate issues they've encountered. Bettenburg et al. [5] conducted an initial review of bug report research. They summarized that the most widely used information in bug reports includes "Steps to Reproduce (S2R)", "Stack traces", and "Test cases". They also proposed a tool to measure the quality of a new bug report. Xie et al. [44] investigated how projects like Gnome and Mozilla utilize non-developer reporters to improve bug report quality. They found that the primary impact of bug reports lies in filtering reports, supplementing missing details, and specifying the affected product. Besides, they asked some bug reporters and developers why they changed the contents of bug reports and then conducted an empirical study on four open-source projects. Their research found that fixing a changed bug report took more time. Zhang et al. [45] conducted a comprehensive analysis of bug report literature, categorizing studies based on bug report stakeholders. They summarized that it is urgent to ensure the quality of bug reports and automate bug reports triage and localization. The work most relevant to our research is another study by Zhang et al. [46]. They examined the utility of the pull request template (PRT) and conducted an in-depth analysis of four progressive research questions. Their detailed investigation revealed that PRTs aid developers in overseeing open-source projects, streamlining review processes, and identifying duplicate requests. To the best of our knowledge, no prior research explores BRTs. Unlike prior studies, our study not only explores BRTs but also offers a set of guidelines derived from BRT change histories. We aim to provide developers with robust strategies for creating effective templates.

7 THREATS TO VALIDITY

Construct Validity. It concerns the relationship between the treatment and the outcome. The first threat involves the rationality of the questions asked. Our interest lies in assessing bug report templates on GitHub. To achieve this goal, we focus on its current usage, effects on bug reports, and practices. We believe that these questions can provide unique insights and value for practitioners and researchers. The second threat involves potential measurement imprecision when constructing our dataset. To calculate the four influence aspects of BRTs, we leverage keyword matching to identify related bug reports, which could result in false negatives and false positives. To reduce false negatives, we collected 10 related keywords. To mitigate false positives, the first two authors reviewed and discussed all identified bug reports to find and remove false positives. Thus, we believe this threat is limited. Besides, we have made our dataset and results public, including all keywords used in our work [35]. The third threat relates to the process of selecting the types of BRT changes for classification. These selections may not represent all possible change types and template-related design guidelines. To mitigate this threat, we included a free-form "Other" option in the survey answers.

Internal Validity. It concerns the threats to how we perform our study. Manual labeling of BRT changes poses a subjective threat to validity. To minimize this threat, two authors labeled BRT changes independently and consulted an experienced doctor in qualitative research to reach an agreement through several discussions. The high agreement level indicates reliability. The second threat relates to selecting projects with BRTs when answering RQ2. Although we selected a sample of projects, the margins of error are acceptable [32]. The third factor is the response rate of the survey. Although it appears low (9.75%), it aligns with the suggested minimum response rate for survey studies [32]. Finally, respondent fatigue bias is a typical co-factor in survey studies. We mitigated this threat by running a pilot study with three professional developers and three Ph.D. students to ensure that both surveys could be answered within five minutes.

External Validity. It concerns the threats to generalize our findings. When conducting this study, we mainly focus on popular open-source projects on GitHub (Section 4). While all open-source projects could benefit from BRTs, the BRT mechanism is rarely applied to less popular projects, as discovered in this study. This is likely because popular projects receive more bug reports, and therefore BRTs can bring them more benefits.

8 CONCLUSION AND FUTURE WORK

In this work, we empirically explored the usage, effects, and design guidelines of bug report templates (BRTs) in open-source projects. Our meticulous analysis shows that BRTs can significantly improve the probability of addressing bug reports and reduce the average number of bug report comments. By analyzing how developers update BRTs, we defined a taxonomy of BRT changes and distilled a set of design guidelines for BRTs. In addition, we collected practitioners' perceptions of BRTs through an online survey, which reveals that most users acknowledge the importance and usefulness of BRTs. Based on our findings, we propose a series of suggestions to help developers design good BRTs and point out several promising research directions related to BRTs for researchers.

ACKNOWLEDGMENTS

This work was partially supported by the the National Science Foundation of China (No.62372398, No. 72342025, and No. U20A20173), the National Key Research and Development Program of China (No. 2021YFB2701102), the Fundamental Research Funds for the Central Universities (No. 226-2022-00064), and the Ningbo Natural Science Foundation (No. 2023J292).

REFERENCES

[1] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. 2020. Software documentation: the practitioners' perspective. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 590–601.

[2] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 1199–1210.

[3] I Elaine Allen and Christopher A Seaman. 2007. Likert scales and data analyses. *Quality progress* 40, 7 (2007), 64–65.

[4] Mohsin Alvi. 2016. A manual for selecting sampling techniques in research. (2016).

[5] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*, Mary Jean Harrold and Gail C. Murphy (Eds.). ACM, 308–318.

[6] Tingting Bi, Xin Xia, David Lo, John Grundy, and Thomas Zimmermann. 2020. An empirical study of release note production and usage in practice. *IEEE Transactions on Software Engineering* (2020).

[7] Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. 2010. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, CSCW 2010, Savannah, Georgia, USA, February 6-10, 2010*, Kori Inkpen, Carl Gutwin, and John C. Tang (Eds.). ACM, 301–310.

[8] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 396–407.

[9] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[10] GitHub. 2024. Ant-design-vue project. https://github.com/vueComponent/ant-design-vue.

[11] GitHub. 2024. Astro project. https://github.com/withastro/astro.

[12] GitHub. 2024. Cefsharp project. https://github.com/cefsharp/CefSharp.

[13] GitHub. 2024. Cli project. https://github.com/urfave/cli.

[14] GitHub. 2024. Dva project. https://github.com/dvajs/dva.

[15] GitHub. 2024. GitHub REST API. https://docs.github.com/en/rest.

[16] GitHub. 2024. Kubebuilder project. https://github.com/kubernetes-sigs/kubebuilder.

[17] GitHub. 2024. Minikube project. https://github.com/kubernetes/minikube.

[18] GitHub. 2024. Rome project. https://github.com/rome/tools.

[19] GitHub. 2024. Swoole project. https://github.com/swoole/swoole-src.

[20] Mehdi Golzadeh, Alexandre Decan, Eleni Constantinou, and Tom Mens. 2021. Identifying bot activity in GitHub pull request and issue comments. In *3rd IEEE/ACM International Workshop on Bots in Software Engineering, BotSE@ICSE 2021, Madrid, Spain, June 4, 2021*. IEEE, 21–25.

[21] Georgios Gousios. 2013. The GHTorent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 233–236.

[22] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: Github's data from a firehose. In *9th IEEE Working Conference of Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland*, Michele Lanza, Massimiliano Di Penta, and Tao Xie (Eds.). IEEE Computer Society, 12–21.

[23] Anna Gromova, Iosif Itkin, Sergey Pavlov, and Alexander Korovayev. 2019. Raising the Quality of Bug Reports by Predicting Software Defect Indicators. In *19th IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2019, Sofia, Bulgaria, July 22-26, 2019*. IEEE, 198–204.

[24] Junxiao Han, Shuiguang Deng, Xin Xia, Dongjing Wang, and Jianwei Yin. 2019. Characterization and Prediction of Popular Projects on GitHub. In *43rd IEEE Annual Computer Software and Applications Conference, COMPSAC 2019, Milwaukee, WI, USA, July 15-19, 2019, Volume 1*. IEEE, 21–26.

[25] Dongyang Hu, Tao Wang, Junsheng Chang, Gang Yin, and Yang Zhang. 2018. Multi-Discussing across Issues in GitHub: A Preliminary Study. In *25th Asia-Pacific Software Engineering Conference, APSEC 2018, Nara, Japan, December 4-7, 2018*. IEEE, 406–415.

[26] Qiao Huang, Xin Xia, David Lo, and Gail C. Murphy. 2020. Automating Intention Mining. *IEEE Trans. Software Eng.* 46, 10 (2020), 1098–1119.

[27] Haruna Isotani, Hironori Washizaki, Yoshiaki Fukazawa, Tsutomu Nomoto, Saori Ouji, and Shinobu Saito. 2021. Duplicate Bug Report Detection by Using Sentence Embedding and Fine-tuning. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, Luxembourg, September 27 - October 1, 2021*. IEEE, 535–544.

[28] Mona Erfani Joorabchi, Mehdi MirzaAghaei, and Ali Mesbah. 2014. Works for me! characterizing non-reproducible bug reports. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, Premkumar T. Devanbu, Sung Kim, and Martin Pinzger (Eds.). ACM, 62–71.

[29] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. 2014. The promises and perils of mining GitHub. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, Premkumar T. Devanbu, Sung Kim, and Martin Pinzger (Eds.). ACM, 92–101.

[30] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2021. Predicting issue types on GitHub. *Sci. Comput. Program.* 205 (2021), 102598.

[31] Barbara A. Kitchenham and Shari Lawrence Pfleeger. 2008. Personal Opinion Surveys. In *Guide to Advanced Empirical Software Engineering*, Forrest Shull, Janice Singer, and Dag I. K. Sjøberg (Eds.). Springer, 63–92.

[32] JWKJW Kotrlik and CCHCC Higgins. 2001. Organizational research: Determining appropriate sample size in survey research appropriate sample size in survey research. *Information technology, learning, and performance journal* 19, 1 (2001), 43.

[33] Eero I. Laukkanen and Mika Mäntylä. 2011. Survey Reproduction of Defect Reporting in Industrial Software Development. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM 2011, Banff, AB, Canada, September 22-23, 2011*. IEEE Computer Society, 197–206.

[34] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empir. Softw. Eng.* 22, 6 (2017), 3219–3253.

[35] Replication Package. 2024. Anonymous. Artifact for "Inside Bug Report Templates: An Empirical Study on Bug Report Templates in Open-Source Software". Zenodo. https://doi.org/10.5281/zenodo.10949378.

[36] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the Content of GitHub README Files. *Empir. Softw. Eng.* 24, 3 (2019), 1296–1327.

[37] Quora. 2024. Quora is a place to gain and share knowledge. https://quora.com/.

[38] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. 2017. Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 314–319.

[39] Yang Song and Oscar Chaparro. 2020. BEE: a tool for structuring and analyzing bug reports. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1551–1555.

[40] Davide Spadini, Maurício Finavaro Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 908–911.

[41] Charles Spearman. 1961. The proof and measurement of association between two things. (1961).

[42] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.

[43] Stackoverflow. 2024. Stack overflow is the largest, most trusted online community for developers to learn, share their programming knowledge. https://stackoverflow.com/.

[44] Jialiang Xie, Minghui Zhou, and Audris Mockus. 2013. Impact of Triage: A Study of Mozilla and Gnome. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013*. IEEE Computer Society, 247–250.

[45] Jie Zhang, Xiaoyin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. 2015. A survey on bug-report analysis. *Sci. China Inf. Sci.* 58, 2 (2015), 1–24.

[46] Mengxi Zhang, Huaxiao Liu, Chunyang Chen, Yuzhou Liu, and Shuotong Bai. 2021. Consistent or not? An investigation of using Pull Request Template in GitHub. *Information and Software Technology* (2021), 106797.

[47] Mengxi Zhang, Huaxiao Liu, Chunyang Chen, Yuzhou Liu, and Shuotong Bai. 2022. Consistent or not? An investigation of using Pull Request Template in GitHub. *Information and Software Technology* 144 (2022), 106797.

[48] Junji Zhi, Vahid Garousi-Yusifoglu, Bo Sun, Golara Garousi, S. M. Shahnewaz, and Günther Ruhe. 2015. Cost, benefits and quality of software development documentation: A systematic mapping. *J. Syst. Softw.* 99 (2015), 175–198.

[49] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Trans. Software Eng.* 36, 5 (2010), 618–643.

[50] Weiqin Zou, Weiqiang Zhang, Xin Xia, Reid Holmes, and Zhenyu Chen. 2019. Branch Use in Practice: A Large-Scale Empirical Study of 2, 923 Projects on GitHub. In *19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019, Sofia, Bulgaria, July 22-26, 2019*. IEEE, 306–317.