An Empirical Study of the Bug Link Rate

Chenglin Li[1], Yangyang Zhao[1,*], and Yibiao Yang[2]
[1] Zhejiang Sci-Tech University, Hanzhou, Zhejiang, China
[2] Nanjing University, Nanjing, Jiangsu, China
lichenglin2110@163.com, yangyangzhao@zstu.edu.cn, yangyibiao@nju.edu.cn
*corresponding author

***Abstract*—Defect data is critical for software defect prediction. To collect defect data, it is essential to establish links between bugs and their fixes. Missing links (i.e. low link rate) can cause false negatives in the defect dataset, and bias the experimental results. Despite the importance of bug links, little prior work has used bug link rate as a criterion for selecting subjects, and there is no empirical evidence to know whether there are simpler alternative criteria for evaluating a project's link rate to aid selection. To this end, we conduct a comprehensive study on the bug link rate. Based on 34 open-source projects, we make a detailed statistical analysis of the actual link rates of the projects, and examine the factors affecting link rates from both quantitative and qualitative perspectives. The findings could improve the understanding of bug link rates, and guide the selection of better subjects for defect prediction.**

Keywords*—*bug link rate; defect data; defect prediction; mining software repositories; Bug priority

I. Introduction

Defect data plays an important role in software maintenance especially predicting defects. Most of the existing studies leverage the SZZ algorithms to collect defect data [11, 14, 16, 15, 18, 29], which rely on the premise that developers leave hints or links about bug fixes in the change logs. Ideally, when a developer fixes a bug, they will attach the necessary information (e.g., bug id) in the change log to help trace the bug so as to establish a **link** between the bug and its fixing commit. However, it is not compulsory for developers to do that [7]. As a result, in fact, many bug fixing commits do not carry information about which bugs they fix [10, 26]. This may lead to the loss of bug links, resulting in a low **linking rate** (i.e., the proportion of bugs of a project that are linked to the bug-fixing commits).

In prior studies, especially defect prediction, it has been suggested that missing links (i.e. low link rates) can adversely affect algorithms and functions that rely on such data [2, 5]. If some links are missing, the bug-fixing changes cannot be identified. Hence the involved modules will be treated as defect-free, which leads to false negatives in the defect dataset. Building defect prediction models based on such biased datasets will threaten the validity of conclusions and findings. Therefore, the links between bugs and their fixes play an important role in the dataset quality for defect prediction. Hence it is suggested to perform studies on the subjects with higher link rate to ensure the defect data quality. However, despite the significance of the bug links, little prior work leverages bug link rate as a criterion for selecting high-quality research subjects before data collection, which may skew the conclusions. One possible reason for this may be that calculating the link rate of a project is time-consuming [9, 12, 25]. As a result, it is valuable to investigate whether there are more accessible factors to help select projects with high link rates.

To this end, we are motivated to conduct a comprehensive study on the bug-fixing links and the factors that affect the link rate of a project. Based on 34 large open-source projects, we make a detailed statistical analysis of the actual linking rate of the projects and examine the factors affecting the link rate from both quantitative and qualitative perspectives.

Following are the main contributions:

1) We collect data from more than 600 popular projects and screen out 34 projects to investigate the actual link rate. The results reveal that, for most studied projects, the link rates are over 60%. However, there are still 23.5% of projects whose link rate are relatively low (less than 60%), with a minimum value of only 9%.
2) We analyze the correlation between the bug priority and the link rate. The results show that the proportion of bugs with high priority is positively correlated with link rates at the significant level of 0.01. This can provide researchers with an alternative criterion to estimate the link rate when selecting projects.
3) We study the factors affecting link rate from both quantitative and qualitative perspectives. For the quantitative study, we design five metrics from two dimensions of software development efficiency and bug reporter quality, and the results reveal a significant correlation between bug reporter quality and link rate. For the qualitative study, we manually analyzed the possible reasons for the low link rate and summarized the reasons into three aspects, including lack of attention, lack of standards, and invalidation of bug reports.

The findings and implications in this study will help researchers to identify better research subjects with higher linking rate, and give some inspiration for developers to improve the link rate of their project.

Paper outline The remainder of this paper is organized as follows. SectionII explain our research motivation. SectionIII describe the data we collected and the experimental setup, in SectionIV, we present results for three research questions and discuss the implications, and Section V discloses threats to the validity of this study. Finally, Section VII concludes our

2693-9177/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS57517.2022.00028

paper.

II. Motivation

In software development, version control systems (e.g. GIT) and issue tracking systems (e.g. Jira) are widely used for project management. The former contains information on software developments, including the source codes, commits, change logs, etc. While the latter includes defect information, such as bug reports and their status. Such information is valuable for software quality assurance, especially predicting defects [3]. To collect defect data from them, existing studies mostly leveraged the SZZ algorithm [33] and its variants [8]. The main steps are as follows: (1) identify the bug-fixing commits by establishing links between bugs and their fixes; (2) leverage the diff command to locate the modified lines purely for fixing bugs; (3) trace back through the code history to identify the changes that induce buggy lines by the annotate and blame commands; (4) finally, filter out the innocent ones, such as changes on blank lines, comments, etc [33, 9].

In the process of the SZZ algorithm, the links between bugs and their fixes play an important role in identifying the bug-fixing commits. Establishing such links relies on the premise that developers leave information about bug fixes when they commit code changes to deal with a bug. For example, when a developer submits a commit for purely fixing a bug, he is supposed to attach the necessary information (e,g. bug id) in the change log to help trace the bug. However, in practice, some developers neglect to do that since it is not a mandatory action. As a result, many bug links cannot be successfully established, which causes the link rate of the project to be relatively low. For a project with a low link rate, many bug-fixing commits cannot be identified, the related modules hence will be treated as defect-free, resulting in false negatives in defect data. Training defect prediction models using the defect data collected from such projects will bias the experimental results. In addition, it has been pointed out in prior works that, link loss can also lead to dataset quality problems, which adversely affect applications and algorithms relying on such data [6, 7, 2, 5]. Therefore, before collecting defect data, it is necessary to select those projects with the higher link rate as experimental subjects to ensure the data quality. However, few prior studies have used bug link rate as one of the criteria for selecting subjects.

Even though researchers are aware of the importance of link rate and try to select projects based on their link rates, there are still some difficulties in practice. One reason is that calculating the link rate is time-consuming [9, 12, 25]. It may be frustrating that, despite a lot of effort in calculating the link rate of a project, researchers should still abandon it if it has a low link rate. So it might be more acceptable if there were simpler alternative criteria for evaluating a project's link rate to help with selection.

Due to the importance of link rate, many studies have been conducted to investigate bug links, mainly including two aspects: (1) investigated the impact of missing links on the dataset quality [2, 26]; (2) proposed a number of automated or semi-automated tools to create data on links between bugs and bug fixing changes, such as Linkster [6], Relink [35], and so on. These studies provided a foundation for subsequent research about bug links. However, as mentioned above, there are still some limitations regarding bug links that are unresolved in the existing studies.

(1) When collecting defect data, existing studies ignored the link rate when selecting experimental subjects. Consequently, the link rates of candidate projects were unclear, which may adversely affect the reliability of defect prediction results.

(2) Many previous studies have investigated the impact of low link rates on the quality of datasets, but there is still no empirical evidence to know which factors may affect link rates.

(3) Calculating link rates may consume a significant amount of work, such as some consumption on projects with low link rates may be worthless. However, previous studies have not presented effective ways to help researchers preliminarily determine the link rate of a project.

In order to fill the gaps left by the previous work, we collected data from 34 large open-source projects from the Apache Software Foundation as our dataset and conducted a comprehensive study on the bug link rate.

III. Methodology

In this section, we first highlight the three research questions of our study. Then we describe the data collection method in detail. After that, we present the definition and calculation approach of link rate. Finally, we introduce experimental methods for these three research questions.

A. Research questions

RQ1*: What are the characteristics of projects' link rates?*

As we know, collecting defect data for projects with high link rates is critical to ensure data quality. But unfortunately, it is still unknown how the link rate is distributed in open-source software projects. This gives rise to the question of whether there is a risk in randomly selecting subjects from them to collect defect data. Intuitively, if the link rates of all (at least most) projects are very high, then the quality of candidate projects selected from them will be satisfactory. But if, on the contrary, a large percentage of projects have a low link rate, it would be unwise to select such projects for defect prediction randomly. Therefore, to gain insights into the link rate, this question is proposed for investigating the characteristics of projects' link rates.

RQ2*: What is the relationship between project link rate and bug priority?*

To some extent, the priority of a bug reflects its severity. In general, the developers usually pay more attention to the bugs with a higher priority. Therefore, intuitively, the higher the priority of a bug, the higher the probability of being dealt with. If there are more high-priority bugs in the project, more bugs may be dealt with, and thus the link rate may be higher. This leads us to naturally conjecture that there is a positive association between the link rate and the proportion of high-priority bugs. To verify this assumption, we propose RQ2.

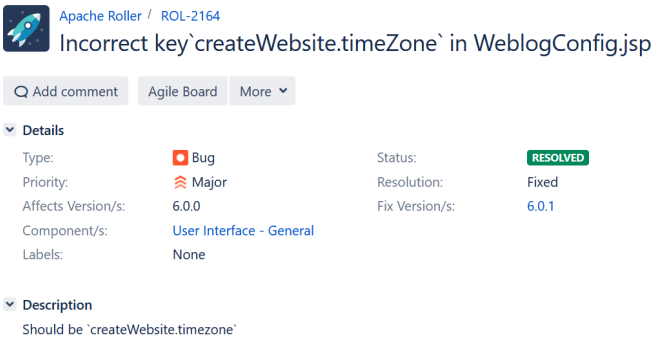

Figure 1. A bug report of JIRA with issue ID ROL-2164.

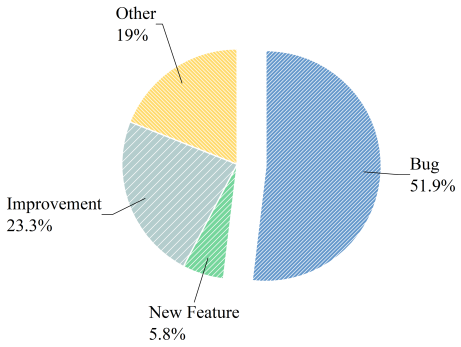

Figure 2. The distribution of issue report types in 659 projects.

For this research question, we will investigate the relationship between the priority of bugs and the link rate of these research projects.

***RQ3**: What factors affect the link rate of a project?*

Despite the importance of link rate, few existing defect prediction studies have used link rate as a screening criterion when selecting study subjects. The main reason is that calculating the link rate is time-consuming, and there are no relevant criteria to help estimate the link rates of candidate projects. To address this deficiency, in RQ3, we analyze which factors affect the link rate of a project. Based on the experimental results, we will derive informative implications to help researchers filter out these high link-rate projects.

B. Data collection

Issue tracking systems store valuable data for software quality assurance. In particular, Jira, as an issue tracking system, is widely adopted to manage issues by a large number of open-source projects. Since Jira contains detailed event information, many previous studies have constructed defect datasets by mining bug data from Jira [9, 37, 22, 19, 32]. Specifically, bug reports in Jira are often linked to corresponding bug fix commits. Therefore, in order to study the links between bug reports and commits, we also collect bug data from Jira. In the Jira system, each Jira issue report is assigned a unique issue id. The issue id is composed of the project name and number. For example, ROL-2164, as shown in Figure 1, is an issue id for the project Apache Roller. Table I illustrates fields of an issue we collect from Jira. The first column is the name of the attributes and the second column is the corresponding description.

TABLE I
THE ATTRIBUTES COLLECTED IN ISSUE REPORTS.

Attribution	Description
Title	A title of reports which briefly describe this issue
Description	The detail description of issue
Id	Consists of keywords and an unique numbers that identify issue reports and establish link between bug and bug-fixing commits
Type	The type of this issue report
Status	The current status of the issue being processed
Resolution	The result of the issue being processed
Priority	Indicates the priority of the issue being processed
Reporter	The person who reported the issue

Among the projects that use Jira to manage their issues, we select Apache open-source projects as our research projects. The major reasons are as follows: 1) they employ Jira to store and manage their issues, which ensures we can obtain sufficient information on issue reports to build a dataset; 2) they are mature, which means these projects have abundant data of issue reports and commits log; 3)open-source extensively applied in various domains, even make our study results more convincing and reduce threats to external validity; 4) they are widely served as research subjects in previous studies, which proves that the data collected from Apache projects are suitable for software engineering study [9, 7, 34, 23].

659 Apache open-source projects' issue reports are collected. According to the report's type, we count the proportion of different types of issue reports and present the results in Figure 2. Each sub-part represents the proportion of the corresponding type. As it is shown in Figure 2, 51.9% of issue reports report bugs. In addition, for these 659 projects, we also collect the data of commits, pull requests, and git issues from GitHub.

Project filtering: From these 659 projects, we will filter the dataset to make the projects closer to the ones that might be selected in a real defect prediction study. We formulate three rules to select projects, and the incompetent projects will be excluded. The three rules are as follows. First, by manually checking, we find some projects are inactive or retired. To avoid the influence of outdated data, the projects should be active in the recent year. Second, we find that in some projects, the data of bug reports is insufficient. These projects either have not enough issue reports, or the percentage of bug reports is too low. Therefore, we retain the projects with over 1000 issue reports and a bug report percentage of over 50%. Third, the projects should only host their code in one GitHub repository. Since some projects are distributedly managed in multiple repositories and lack an explicit list to record the information of repositories. This case may affect data integrity.

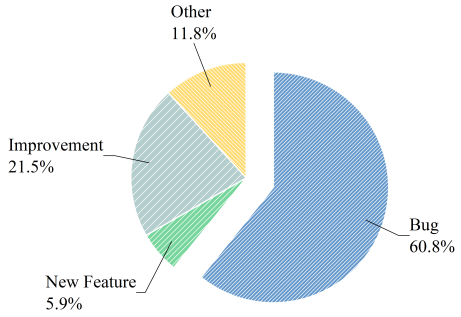

Figure 3. The distribution of issue report types in the selected 34 projects.

TABLE II
THE STATISTIC OF QUANTITY RANGES OF ISSUE REPORTS IN 34 PROJECT

Range	Projects	Proportion
Between 1000 and 2000	13	38.23%
Between 2000 and 3000	7	20.59%
More than 3000	14	41.18%

Finally, after the filtering process, 34 projects are left. We further collect the data for these 34 projects, which contains issue reports from JIRA and the information of all commits, pull requests and git issues from Github. Table II shows the distribution of the number ranges of issue reports for the 34 projects. We divide the projects into three different intervals based on the number of issue reports. The first column indicates the quantity ranges of issue reports, the second column indicates the number of projects belonging to this range, and the third column is the proportion of these projects. For example, the first row of Table II indicates that there are 13 projects with the number of issues between 1000 and 2000, accounting for 38.23%(13/34).

Furthermore, we count the proportion of different types of reports for these 34 projects and show their average distribution in Figure 3. As shown in Figure 3, 60.8 percent of issue reports report bugs. Compared with Figure 2, the filtering process increases the percentage of bug reports. More specifically, Figure 4 further illustrates the statistics of bug ratios of these 34 projects using a boxplot, which shows the median (the horizontal line within the box), the 25th, and the 75th percentiles (the lower and upper sides of the box) of the proportion of bugs. Each point in Figure 4 stands for a project, and the value in the vertical axis represents the corresponding percentage of bugs. As it is shown in Figure 4, the bug ratios of these 34 projects are relatively high, with an average bug ratio of 62.04%. Particularly, of these projects, four have a bug report percentage above 80%.

C. Definition of link and link rate

To study link rate, we first filter out the valid bugs, then establish links between them and their fixes, and finally calculate the link rate for each project (i.e. the proportion of bugs of a project that are linked to the bug-fixing commits).

Bug filtering: In order to filter out the valid bugs, we did the following process. First, we ignore all the bugs marked

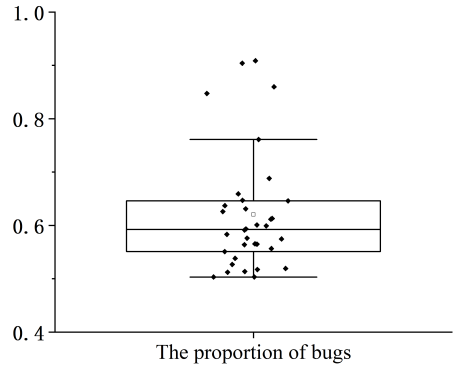

Figure 4. The percentage of bugs in 34 projects.

with *OPEN*, since apparently there is no full fix commits for them. Second, we exclude the invalid bugs, as they would not be dealt with. More specifically, in the *CLOSED* bug reports, there are many invalid issue reports, which are indicated in the *RESOLUTION* field. Table III lists the resolutions to the common invalid bugs. In Table III, the first column shows the common resolution types of invalid bugs, and the second column is the corresponding description. For these bugs with the resolutions in Table III, we call them invalid bugs. In summary, both the unresolved and invalid bugs, they do not have corresponding bug-fixing commits. If we take them into account when we calculate the link rate, it will reduce the real link rate of the project, thus adversely affecting the results. Therefore, we exclude them to ensure the validity of our study.

TABLE III
THE INFORMATION OF INVALID BUGS.

Resolution	Description
Duplicate	The problem is a duplicate of an existing issue.
Won't Fix	The problem described is an issue which will never be fixed.
Cannot Reproduce	All attempts at reproducing this issue failed, or not enough information was available to reproduce the issue. Reading the code produces no clues as to why this behavior would occur. If more information appears later, please reopen the issue.
Incomplete	The problem is not completely described.
Not A Problem	The described issue is not actually a problem - it is as designed.
Not A Bug	Not A Bug.
Won't Do	Won't Do.
Invalid	The problem isn't valid and it can't be fixed.

Link establishing: To establish links between bugs and their fixes, many existing studies took advantage of the Jira add-ons, which is convenient to refer to the bug report which they are addressing in the commit logs. With such benefit, the links between bugs and commits can be automatically inferred when commits are checked into the repository [7]. However, with manual checking, we found that many projects did not adopt this function to manage their bugs. Therefore, to guarantee the integrity of the link data, we take a different approach to build links. We use the hints about bugs left by the developer in the change log, such as bug ID, to establish the links. During the process of building links, we found two kinds of links,

which we call direct links and indirect links. The *direct link* means that the defect information is directly attached to the commit log. While the *indirect link* indicates that there is no direct correlation between bug report and bug fix commit, but through issue or pull request indirectly. For different types of links, we use different strategies to establish bug links. Details are as follows. In particular, for the one-to-many case (i.e. a bug is fixed through multiple commits), we only create one link. What is more, for the correctness of the data collection process, the authors double checked the accuracy of the link-establishing scripts.

Direct link The bug information is directly attached to the bug-fixing commit log. Figure 5 shows an example of a direct link. As can be seen, the bug id "OPENJPA-2861" is explicitly attached in the message of a fixing commit. In such cases, we establish links by extracting bug ID from bug-fixing commits straightly. As is shown in Figure 6, we first extract the bug id from bug-fixing commits. Second, we search bug reports to find the report with the same bug id. Finally, we establish a link between them.

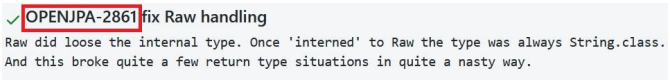

Figure 5. An example of bug fixing commit that contains Bug ID .

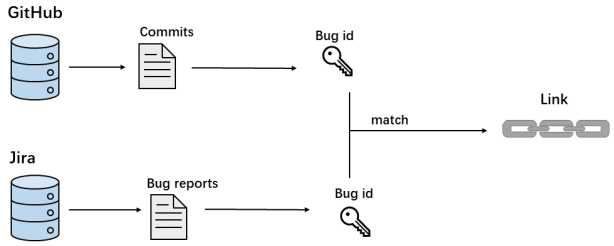

Figure 6. The method of establishing direct links for the bug fixing commits .

Indirect link There are two cases of indirect links. (1) Developers attached bug ID to the pull request page but neglected to attach it to the involved commits. Figure 7 shows an example of an indirect link with a pull request. The bug ID CXF-8345 is attached to the title of the pull request. As can be seen, this pull request is merged for fixing bug CXF-8345. However, the corresponding bug ID is not added to the change log of the commits involved in this pull request. In such case, if we only search the bug information in the commit log, we will lose the link for this bug. In order to establish the indirect links based on pull requests, as is shown in Figure 8, for each project, we check if the pull requests contain bug-fixing information. We consider the pull requests with bug ID to fix bugs. We mark the involved commits in this pull request as bug-fixing commits. Finally, we establish an indirect link between these commits and the bug report.

Figure 7. An example of Pull Request that contains Bug ID .

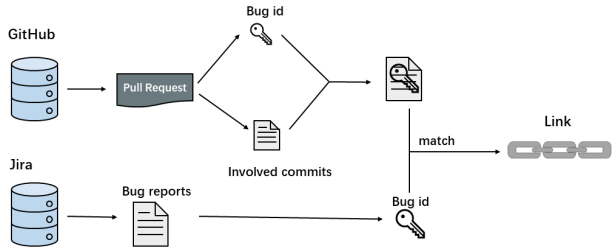

Figure 8. The method of building links for the merge commits of Pull Request which contains Bug id .

(2) Developers attached bug ID to the issue page but neglected to attach it to the involved commits. Figure 9 shows an example of an indirect link with a git issue. The bug ID COUNCHDB-3221 is attached to an issue page. As can be seen, issue #858 is created for fixing bug COUNCHDB-3221. However, the corresponding bug ID is not added to the change log of the commits involved. In such a case, we should extract the bug ID from the Issue information to prevent the loss of the link. As is shown in Figure 10, first, we search for bug id from the issue pages. second, we mark the involved commits based on the issue ID as bug-fixing commits. Third, we search for bug report by the same bug ID and establish links between the bug report and the involved commits.

Figure 9. A example of Issue that contains Bug ID .

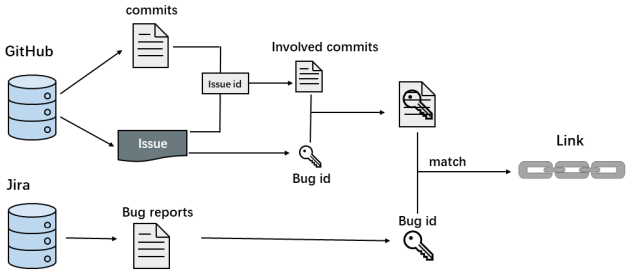

Figure 10. The method of building links for the corresponding commits of Issue which contain Bug ID .

Link rate calculation: With the dataset collected above, we calculate the link rate. In this study, the link rate means the

percentage of valid bugs which are linked to their bug-fixing commits. We establish the links for the valid bug reports that meet the three conditions described above. Specifically, for a project, assuming that the number of valid bug reports is Y, out of Y bug reports, the number of linked bug reports is X. Then, the link rate of a project is X/Y.

D. Validate method

In this section, we illustrate the validation methods for each research question.

To answer RQ1, we will calculate the link rate for each project following the steps in section III-C, and perform statistical analysis to study the characteristics of projects' link rates.

For RQ2, we will study the relationship between project link rate and bug priority. In Jira, each bug report is prioritized based on the property of the bug report and then assigned to the fixer. By observation, the priority of the bugs in these projects are labeled in five levels, including Blocker, Critical, Major, Minor, and Trivial from highest to lowest severity, which is shown in Table IV. In Table IV, the first column is the name of priorities and the second column is their description. We calculated the proportion of various priority bugs of each project. Moreover, we classify the Blocker, Critical and Major bugs as Urgent bugs, and the Minor and Trivial bugs as non-Urgent bugs. We will analyze the correlation between different level priorities and the link rate by the Pearson correlation analysis method.

TABLE IV
THE PRIORITY LEVEL IN OUR REPORTS DATA.

Attribution	Description
Blocker	Blocks development and/or testing work, production could not run
Critical	Crashes, loss of data, severe memory leak.
Major	Major loss of function
Minor	Minor loss of function, or other problem where easy workaround is present.
Trivial	Cosmetic problem like misspelt words or misaligned text.

In order to answer RQ3, we investigate the factors which may influence the link rates of projects from both quantitative and qualitative perspectives.

In the quantitative study, we define the following six metrics from two dimensions, including software development efficiency and bug reporter quality. We adopt Pearson correlation analysis to analyze the correlation between the metrics and the link rate of projects.

Software development efficiency: Intuitively, in order to deliver software products quickly, the development efficiency of the project is often very high, and its issues are also processed quickly. This leads us to conjecture that bugs in projects with high development efficiency may be quick to deal with, resulting in a higher link rate. To verify this, we collect the following development efficiency metrics for each project.

- **Mean time interval of commits(MTIC)**: Average interval time between commits and the their next commit in a project, in hours.
- **Mean number of commits each month(MNC)**: For a project, this is the average number of commits submitted per month.
- **Number of pull requests opened/closed per month(NPRO/NPRC)**: We count the quantity of pull requests opened or closed for per month.
- **Mean pull request latency per mouth(MPRL)**: in hours, computed as the difference between the timestamp when the PR was closed and that when it was opened each month. The mean is computed over all PRs since the project was created.

Bug reporter quality: Previous studies have mentioned that a bug reporter's contribution may affect bug fixing, that is, the more the bugs reporter participates in, the more likely the bug will be fixed successfully [7]. Thus, we believe that the quality of the bug reporter may have an impact on the link rate. The quality of the bug reporter metric is described as follows:

- **Average Quality of the bug reporters(AQR)**: We measure a bug reporter's quality by the proportion of valid bug reports in all bug reports. For instance, if a bug reporter reported Y bug reports, and X of those bug reports were valid, then the quality of this bug reporter is X/Y. For the whole project, the average quality of reporters (AQR) is the sum of each reporter's proportion of valid reports, and then divided by the total number of reporters. In this study, valid bugs refer to real and fixed bugs, excluding open bugs and invalid bugs (with the resolutions in TABLE III).

In the qualitative study, we manually analyze the factors affecting the link rate. The manual analysis was completed by two researchers, a master student and a supervisor of master's. Specifically, the student discovers the phenomena in data, analyzes the reasons and summarizes the findings. The mentor is responsible for checking these phenomena, verifying these findings, and summarizing the implications.

IV. EXPERIMENTAL RESULTS

In this section, we report the detailed experimental results. Section IV-A presents the calculation results of the link rate of our dataset(RQ1). Section IV-B shows the results of analyzing the correlation between link rate and priority of bugs(RQ2). Section IV-C reveals the results from both quantitative and qualitative studies(RQ3).

A. RQ1: What are the characteristics of projects' link rates?

To answer RQ1, we establish links for valid bug reports and calculate the link rate for each project. The method of link establishing and link rate calculation has been introduced in Section III-C.

Figure 11 shows the distribution of link rates of all projects. In Figure 11, each point stands for a project, and the corresponding value in the vertical axis represents the link rate of this project. As can be seen, the highest link rate is 96.4% and the lowest one is 9.5%. Most projects' link rates are between 60% and 90%. In addition, Figure 12 reveals the distribution of link rate ranges. According to the link rate, we divide the

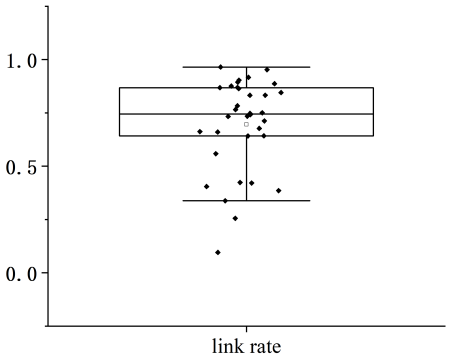

Figure 11. the link rate for each project

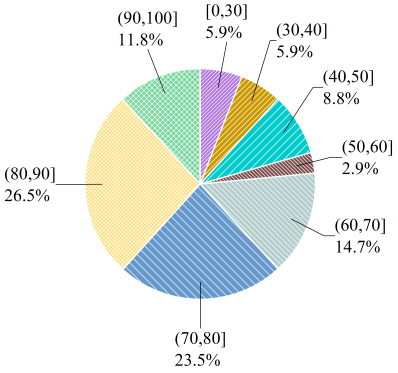

Figure 12. The distribution of link rate ranges

projects into eight ranges. For Example, 23.5% projects have a link rate in the range of (70,80], which means the link rate is more than 70% but less than 80%. As shown in Figure 12, there are approximately 76.5% (26/34) of projects whose link rate is more than 60%, which belong to high link rate projects. Particularly, 11.8 percent of studied projects have a link rate of more than 90%, which is promising as experimental subjects. However, in contrast, there are about 23.5% (8/34) of projects whose link rate is less than 60%, which belong to low link rate projects.

Finding 1: For most studied projects, the link rates are high (over 60%). However, there are still 23.5% (8/34) projects whose link rate are relatively low (less than 60%), with a minimum value of only 9%.

Implication 1: Since there are still many projects with low link rates, randomly selecting projects to collect bug data may bias the results. It is recommended to use the link rate as a criterion to select the experimental subjects.

B. RQ2:What is the relationship between project link rate and bug report priority?

To answer RQ2, we analyze the correlation between the priority of bugs and the link rate of projects by Pearson correlation analysis. For each project, we calculated the ratio of all priorities. Then, we analyzed the correlation between both Urgent bugs and non-Urgent bugs and the link rate.

Table V shows the result of the Pearson correlation coefficient between the two kinds of priority levels and link rate. The

TABLE V
THE CORRELATION COEFFICIENT OF PRIORITY AND LINK RATE BY PEARSON ANALYZE.

Metrics	correlation coefficient
Urgent	0.635 (***)
Non-Urgent	0.101

first column is the name of the bug's priorities, and the second column is the correlation coefficient of different priorities and the link rate at the significant level of 0.05 (denoted by *), 0.10 (denoted by **), or 0.01 (denoted by ***). As shown in Table V, the proportion of Urgent bugs is significantly associated with the link rate of projects at the significant level of 0.01. To further figure out which of the three different priorities of Urgent bugs (i.e. Blocker, Critical and Major priorities) are easier to be fixed. We analyzed the correlation between them and the link rate separately. The results of the correlation analysis are shown in Table VI. The result shows Major bugs have a significant positive correlation with link rate, while Blocker and Critical, two higher priority grades, are not correlated with link rate. The reason for this result may be that the dataset is collected from Apache projects, which are mature and stable. In addition, the development teams of these projects are experienced. As a result, the number of Blocker and Critical bugs is extremely less. To verify this inference, we average the percentages of different priorities for the 34 projects and reveal the result in Figure 13. As it can be seen from Figure 13, the average proportion of bug reports for Blocker and Critical is only 5.31% and 6.10%, respectively, resulting in no significant correlation between them and links.

TABLE VI
THE CORRELATION COEFFICIENT OF BLOCKER, CRITICAL AND MAJOR PRIORITY AND LINK RATE BY PEARSON ANALYZE.

Metrics	correlation coefficient
Blocker	-0.025
Critical	0.099
Major	0.661 (***)

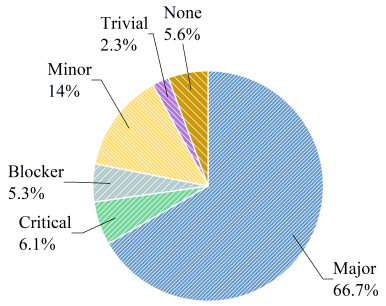

Figure 13. the distribution of different priorities in all projects

Finding 2: The proportion of Urgent bugs is positively correlated with the link rate at the significant level of 0.01. Particularly, in the Urgent priority bugs, the proportion of the Major bugs have a significant positive correlation with the link rate.

Implication 2: For a project, the higher proportion of bugs with high priority, the higher its link rate is probably

to be. When selecting experimental projects, researchers can prioritize those with a higher proportion of Urgent bugs.

C. RQ3:What factors affect the link rate of a project?

To answer RQ3, we present our results from both quantitative and qualitative analysis.

Quantitative study we analyze the correlation between the link rate and two kinds of metrics, including four metrics for software development efficiency and one metric for bug reporter quality, by Pearson correlation analysis. Table VII shows the results of the Pearson correlation analysis. In Table VII, each cell with a value indicates the correlation between the metric and the link rate at the significant level of 0.01 (denoted by ***), 0.10 (denoted by **), or 0.50 (denoted by *).

From Table VII, we find that the metrics of software development efficiency are not significantly associated with link rate. On the contrary, the correlation coefficient between AQR and link rate is 0.603 at the significant level of 0.01, which means the average quality of bug reporters is positively associated with the link rate of projects. The reason may be that, low-quality bug reports cause a lot of extra work for developers, e.g. bug reports with vague descriptions can mislead developers. It has been noted that bugs submitted by high-quality reporters were more likely to be fixed [40]. Therefore, for developers, high-quality bug reports submitted by experienced reporters can help them complete bug fixes, thereby establishing bug fixing links. Moreover, the daily development effort involves many activities in addition to bug fixing, such as implementation of new features and software refactoring. High development efficiency does not mean high defect fixing efficiency. Therefore, development efficiency does not necessarily correlate with link rate.

TABLE VII
THE CORRELATION COEFFICIENT OF SOFTWARE DEVELOP EFFICIENCY AND LINK RATE BY PEARSON ANALYZE.

Metrics	Metrics	correlation coefficient
Software development efficiency	MTIC	-0.196
	MNC	0.153
	NPRO	0.271
	NPRC	0.156
	MPRL	0.241
Bug reporter quality	AQR	0.603 (***)

Findings 3: The quality of bug reporters is positively associated with the link rate. And the development efficiency of projects does not show a significant association with link rate.

Implications 3: The projects with higher average quality of reporters have higher link rate. For researchers, this result provides them with a preliminary judgment about the link rate. Compared with calculating the link rate of all projects, this preliminary judgment may reduce the effort in project screening.

Qualitative study we further study the factors affecting the link rates by manual inspection of our dataset. The findings are as follows. We attempt to give reasonable explanations and implications, so as to give some inspiration to developers who are trying to improve the link rate of their projects. The results are as follows.

(1) Lack of attentions For some projects, in the early days of using Jira to manage bugs, their link rates are relatively low. However, in the later period, the link rates of these projects increase significantly. We find that choosing a time point is critical, and link rates differ significantly before and after that point. For example, in the Daffodil project, we select 1 January 2018 as a time point and calculate the link rate for two periods. The link rate before the point is 8.18%, and after the point is 83.61%. The link rate of the two periods in Daffodil was significantly different. The reason may be that in the early period of using Jira, the development team did not pay enough attention to Jira usage specifications. Therefore, the link rate is relatively low during this period. After a certain moment, the development team paid more attention to the usage specification of Jira, and consciously added bug information to the commit log when repairing bugs. Therefore, the link rate increases after this time. In addition, to check the influence of data imbalance on this conjecture, we further examine the number of bugs before and after the time split point. We find that, there were 653 fixed bugs before the split point and 406 fixed bugs after the split point, which indicates that there is no significant imbalance in the data.

Finding 4: Link rates of some projects significantly change with time.

Implication 4: Developers who want to improve bug traceability should pay more attention to the mechanism of the bug tracking system when managing and fixing bugs. In addition, if researchers want to preliminarily judge the link rate of a project by observing the historical data, they need to observe the link situations in different periods.

(2) Lack of standards By checking the GitHub repositories, we find that some projects with a high link rate have an explicit contribution guideline that regulates how developers can formally contribute a bug-fixing commit. These contribution guidelines are usually documented in README, wiki, or other files. The contribution guidelines illustrate the process for contributing bug-fixing commits and the specification for attaching bug IDs to the commits' log. For example, the link rate of the Ranger project is 84.49%, which is high. It is a contribution guideline, as shown in Figure 14, requires developers to commit the changes with the comment containing the Apache Jira number. Therefore, a standardized contribution guideline can help developers to standardize commits, thus avoiding link loss.

Finding 5: A standardized contribution guideline may help improve the link rate of the project.

Implication 5: For software development, a contribution guideline should be developed to guide contributors' commit specifications, especially the standards of bug-fix information in the change log.

(3) Invalidation of bug reports we examine invalid bug reports and count the proportion of different resolutions of

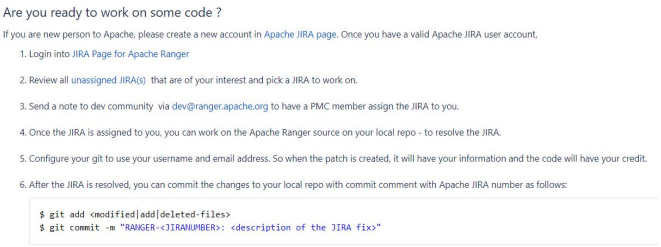

Figure 14. Contribution guides of Ranger.

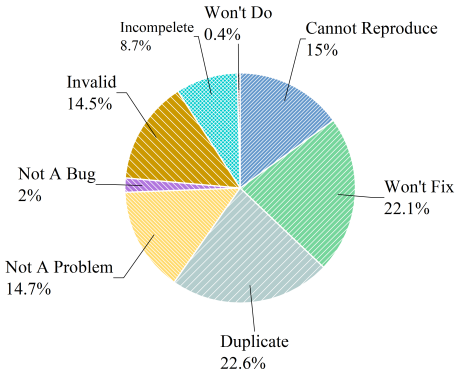

Figure 15. The distribution of different resolutions of invalid bug reports.

invalid bug reports. Figure 15 shows the distribution of different resolutions. These data can be obtained from the *RESOLUTION* field of reports, which reflects the processing result of bugs. Of these different resolutions, *Duplicate*, *Won't Fix* and *Cannot Reproduce* are the three most common, causing almost 60% of bug reports to fail. In addition, invalid bugs will cause some disturbance to the fixers, thereby reducing the efficiency of bug fixing and delaying the establishment of bug links. In this context, we further attempt to give the reporter guidance on improving the report's quality by analyzing the causes of invalid bugs. We manually analyze the reasons for the top three resolutions of invalid reports and provide inspiration to avoid submitting invalid reports.

Duplicate: This resolution indicates that the bug has been reported in the past. It is emphasized that bug reporters should first inquire about Jira to see if this bug has been reported when reporting a bug. However, some reporters ignore this rule when reporting, resulting in duplicate bugs, which is the most common reason for invalid bug reports.

Won't Fix: This resolution indicates that the bug does not need to be processed. There are three main reasons why bug reports are marked as *Won't Fix* :

- *The reporter reported other purposes in bug reports.* The main reason for this is that they do not understand the mechanics of Jira. For example, in Figure 16, the reporter submits a report to seek project support rather than recording a bug.
- *The reporter reported a report that is not a bug.* For example, in Figure 17, the reporter reports a problem but it is the default configuration and design of the system.

Figure 16. Aa example of a bug report ROL-1784.

Figure 17. An example of a bug report ZEPPELIN-5389.

- *The reporter reported a solved bug.* This is different from Duplicate. For example, in Figure 18, the reporter illustrates a bug that has been fixed in the new version. Thus, the bugs won't fix.

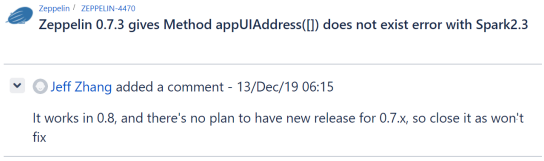

Figure 18. An example of a bug report ZEPPELIN-4470.

Cannot Reproduce: This resolution indicates that the information about the bug is too little to reproduce the bug. This is mainly because the reporter did not specify the bug in detail. Therefore, developers can not reproduce this bug. In the READMEs of some projects, it is mentioned that the reporter should accurately describe the bug and its occurrence, so that developer can solve the problem faster and better.

Findings 6: The three types that account for the largest percentage of invalid bug reports are *Duplicate*, *Won't Fix*, and *Cannot Produce*.

Implication 6: The suggestions to avoid the three invalid bugs and increase the success rate of reporting bugs are as follows:

- In order to avoid *Duplicate* bugs, reporters should query whether it has already been reported in Jira before reporting bugs. Development teams should post available guidelines to constrain reporters. To support our inference, we manually checked two projects, Syncope and Openjpa, with high link rates of 96.46% and 89.39%, respectively, which clearly state that reporters should search existing issues in Jira to check whether the same issues have been reported. Particularly, in terms of invalid bugs, the percentage of duplicated bug reports in these two projects are 9.75% and 22.16% respectively, lower than the average value of all projects. To some extent, this result suggests that a clarity guideline can help reduce Duplicate reports.
- To avoid reporting a *Won't Fix* bug, firstly, reporter should make sure they want to report the bug rather

than other purposes, such as asking for help; Secondly, reporter should determine if it's a real bug rather than a configuration or design problem; Finally, reporter should avoid reporting bugs that occurred in older version but have been fixed in the new version.
- In order to avoid reporting a bug that *Cannot Reproduce*, reporter should add enough detailed information in the report so that the fixer can reproduce the bug easily.

V. Validity

In this section, we discuss the most critical threats to the validity of our study, including Internal validity, External validity, and Construct validity.

Internal Validity Threats to internal validity are concerned with potential deficiencies in the link establishment. In our study, if the method of establishing links is flawed, it will impact the results. To mitigate the threats, we integrally collect commit, pull request, and issue data, which almost contain all project's bug-fix information. Additionally, like previous studies, we match the bug id extracted from such data with the bug report to create a link. While we've collected almost all the information about bugs from GitHub, it's still possible that some factors that cause bug links to be lost. For example, a link cannot be established because of a typo. This case may cause false negative effects, but the existing method of building bug links cannot wholly avoid human causes. In addition, in previous studies, the accuracy of link recognition methods cannot reach 100%. For example, Relink, and traditional heuristics identify links with 89% and 91% accuracy, respectively.

External Validity Threats to external validity are concerned with the generality of our conclusions. To mitigate external threats, we conducted experiments on 34 Apache open-source projects with applications in various domains. Each project's data size is different (more than 1000 submissions, more than 50% of bugs in all reports). To enhance the generality of conclusions, we build a large dataset in our study. However, since our research projects are only based on the JAVA language, and their bug report data is only collected from Jira, the conclusions may not be appropriate for every project. In future work, we will attempt to study projects from other languages and other Issue Track systems, such as Bugzilla, to expand our research scope. In addition, we did not use big data to support the manual findings. For example, in finding 5, we did not use supportive data to demonstrate a direct relationship between self-report and link rate. In future research, we will try to find some reasonable supporting evidence to help us understand our conclusions.

Construct Validity The main threats to construct validity are related to the correlation analysis method. In the correlation research, we only use the Pearson correlation analysis method to measure it. However, Pearson correlation analysis has been widely used in the field of empirical software engineering in the past, which proves that the results obtained by this method are effective. Besides, for RQ3, we investigated the factors which may influence the link rates through five metrics, which may not be comprehensive. We will design more metrics for more systematic analysis in future work.

VI. Related works

A. Bug link

Many previous studies focus on the bug link. On the one hand, some studies proposed various tools to identify bug-fixing commits and build bug links.

Fischer et al. [10] proposed a technique to establish the relationship between code modification and bugs based on keyword extraction. More specifically, they mainly extracted bug id from historical data to link bugs. After that, Sliwerski et al. [33] and Bachmann et al. [1] respectively added the rule of time filtering to their studies based on keyword extraction and further proposed methods for building bug links. Then, Wu et al. [35] proposed Relink model to recover bug links. Compared with prior studies, they considered the similarity of natural language text. Except for text similarity, some researchers also associated words between the description of a bug report and commit message based on context relationship, such as MLink proposed by Nguyen et al. [24] Moreover, for these bugs whose information of bug report and commit are empty or minimum, Le et al. [21] proposed RClinker to solve this problem. RCLinker relied on Change Scribe, an automatic tool to produce the description information of bug reports and commits, and extracted the feature from the information to create a model. This method can predict if a link exists between a commit and a bug report. Besides, the scenario of bug-fixing commit is also used in link establishment studies, such as PaLiMod model [28]. In recent years, Machine learning appeared more and more in the research of recovering bug links. For example, Xie et al. [36] proposed Deeplink, a deep learning approach for link recovery.

On the other hand, some researchers proposed studies about the impact of bug links on software projects. A series of studies have discussed the effects of the bug link on other research. For example, Bachmann et al. [3] mentioned that missing links in the dataset might cause quality problems in research, adversely affecting applications or algorithms in academic research. Moreover, Rahman et al. [26] also indicated that such data may cause bias in defect prediction, but these biases can be alleviated by increasing the dataset's sample size.

B. Bug report

The bug report is closely related to the management of software projects, and many papers have studied the quality and content of bug reports. A good bug report can reduce the workload of developers in maintaining projects. To help reporters report bugs better, Bettenburg et al. [4] and Zimmermann et al. [39] both proposed papers about how to write a good bug report. In addition, many studies have offered automated tools to manage bug reports. Herzig et al. [13] found that bug reports were frequently misclassified, and there were 39% of files were misclassified as defective. To solve this problem, Terdchanakul et al. [30] proposed a bug report

classification model to classify bugs. Yu et al. [38] proposed an automated method to classify bug reports by mining text information. There are also many automated methods for bug priority judgment and prediction. For example, Kanwal et al. [17] proposed a practical approach to recommend bug prioritization to triage based on machine learning automatically. Yuan et al. [31] proposed an automated prediction model for report priority. Moreover, there are many automated methods to judge and predict the bug's severity. Lamkanfi et al. [20] proposed a tool to predict the severity of bug reports through text mining algorithms, which can help the inexperienced triage make estimates. Ramay et al. [27] also proposed an automatic classification method to predict the severity of bug reports based on neural networks.

These studies have researched the impact of bug reports from different perspectives and proposed various automatic tools. Besides, they have greatly improved the quality of bug reports and reduced the consumption of manual inspection.

VII. Conclusion

In this paper, we conduct a comprehensive study on bug-fixing links. First, we perform statistical analysis to study the characteristics of projects' link rates. The results show that for most studied projects, the link rates are high. However, there are still 23.5% of projects whose link rate are relatively low (less than 60%), with a minimum value of only 9%. Second, we study the relationship between project link rate and bug priority. We find that the proportion of Urgent bugs is positively correlated with the link rate at the significant level of 0.01. Particularly, in the Urgent priority bugs, the proportion of the Major bugs have a significant positive correlation with the link rate. In addition, we examine the factors affecting the link rate from both quantitative and qualitative perspectives. In terms of quantitative study, the results reveal that the quality of bug reporters is positively associated with the link rate. In terms of the qualitative study, we manually summarize three main reasons for the low link rate, including lack of attention, lack of standards, and invalidation of bug reports. To better understand the findings, we also raise the implications for each finding. The findings and implications in this study will help researchers to identify better research subjects with higher link rate, and give some inspiration for developers to improve the link rate of their projects.

Acknowledgement

This work was supported by the National Nature Science Foundation of China (Grant No. 62132014 and 62072194), and Zhejiang Provincial Key Research and Development Program of China (No.2022C01045).

References

[1] Adrian Bachmann and Abraham Bernstein. "Data retrieval, processing and linking for software process data analysis". In: *University of Zurich, Technical Report* (2009).

[2] Adrian Bachmann and Abraham Bernstein. "Software process data quality and characteristics: a historical view on open and closed source projects". In: *IWPSE-Evol*. 2009, pp. 119–128.

[3] Adrian Bachmann et al. "The missing links: bugs and bug-fix commits". In: *FSE*. 2010, pp. 97–106.

[4] Nicolas Bettenburg et al. "What makes a good bug report?" In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 2008, pp. 308–318.

[5] Christian Bird et al. "Fair and Balanced? Bias in Bug-Fix Datasets". In: ESEC/FSE '09. 2009, pp. 121–130.

[6] Christian Bird et al. "LINKSTER: Enabling Efficient Manual Inspection and Annotation of Mined Data". In: FSE '10. 2010, pp. 369–370.

[7] Tegawendé F Bissyandé et al. "Empirical evaluation of bug linking". In: *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE. 2013, pp. 89–98.

[8] Daniel Alencar da Costa et al. "A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes". In: *TSE* 43.7 (2017), pp. 641–657.

[9] Yuanrui Fan et al. "The Impact of Mislabeled Changes by SZZ on Just-in-Time Defect Prediction". In: *TSE* 47.8 (2021), pp. 1559–1586.

[10] Michael Fischer, Martin Pinzger, and Harald Gall. "Analyzing and relating bug report data for feature tracking". In: *WCRE*. Vol. 3. 2003, p. 90.

[11] Takafumi Fukushima et al. "An Empirical Study of Just-in-Time Defect Prediction Using Cross-Project Models". In: MSR 2014. 2014, pp. 172–181.

[12] Steffen Herbold et al. "Problems with szz and features: An empirical study of the state of practice of defect prediction data collection". In: *Empirical Software Engineering* 27.2 (2022), pp. 1–49.

[13] Kim Herzig, Sascha Just, and Andreas Zeller. "It's not a bug, it's a feature: How misclassification impacts bug prediction". In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 392–401.

[14] Tian Jiang, Lin Tan, and Sunghun Kim. "Personalized defect prediction". In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2013, pp. 279–289.

[15] Yasutaka Kamei et al. "A large-scale empirical study of just-in-time quality assurance". In: *IEEE Transactions on Software Engineering* 39.6 (2012), pp. 757–773.

[16] Yasutaka Kamei et al. "Studying just-in-time defect prediction using cross-project models". In: *Empirical Software Engineering* 21.5 (2016), pp. 2072–2106.

[17] Jaweria Kanwal and Onaiza Maqbool. "Bug prioritization to facilitate bug report triage". In: *Journal of Computer Science and Technology* 27.2 (2012), pp. 397–412.

[18] Sunghun Kim, E James Whitehead, and Yi Zhang. "Classifying software changes: Clean or buggy?"

In: *IEEE Transactions on software engineering* 34.2 (2008), pp. 181–196.

[19] Sunjae Kwon, Duksan Ryu, and Jongmoon Baik. "eCPDP: Early Cross-Project Defect Prediction". In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE. 2021, pp. 470–481.

[20] Ahmed Lamkanfi et al. "Predicting the severity of a reported bug". In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE. 2010, pp. 1–10.

[21] Tien-Duy B Le et al. "Rclinker: Automated linking of issue reports and commits leveraging rich contextual information". In: *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE. 2015, pp. 36–47.

[22] Ke Li et al. "Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: an empirical study". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 566–577.

[23] Pooya Rostami Mazrae, Maliheh Izadi, and Abbas Heydarnoori. "Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data". In: *ICSME*. 2021, pp. 263–273.

[24] Anh Tuan Nguyen et al. "Multi-Layered Approach for Recovering Links between Bug Reports and Fixes". In: FSE '12. 2012.

[25] Sophia Quach et al. "An empirical study on the use of SZZ for identifying inducing changes of non-functional bugs". In: *Empirical Software Engineering* 26.4 (2021), pp. 1–25.

[26] Foyzur Rahman et al. "Sample size vs. bias in defect prediction". In: *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. 2013, pp. 147–157.

[27] Waheed Yousuf Ramay et al. "Deep neural network-based severity prediction of bug reports". In: *IEEE Access* 7 (2019), pp. 46846–46857.

[28] Gerald Schermann et al. "Discovering loners and phantoms in commit and issue data". In: *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE. 2015, pp. 4–14.

[29] Shivkumar Shivaji et al. "Reducing features to improve code change-based bug prediction". In: *IEEE Transactions on Software Engineering* 39.4 (2012), pp. 552–569.

[30] Pannavat Terdchanakul et al. "Bug or not? bug report classification using n-gram idf". In: *2017 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE. 2017, pp. 534–538.

[31] Yuan Tian et al. "Automated prediction of bug report priority using multi-factor analysis". In: *Empirical Software Engineering* 20.5 (2015), pp. 1354–1383.

[32] Haonan Tong, Bin Liu, and Shihai Wang. "Kernel spectral embedding transfer ensemble for heterogeneous defect prediction". In: *IEEE Transactions on Software Engineering* 47.9 (2019), pp. 1886–1906.

[33] Jacek undefinedliwerski, Thomas Zimmermann, and Andreas Zeller. "When Do Changes Induce Fixes?" In: *SIGSOFT Softw. Eng. Notes* 30.4 (May 2005), pp. 1–5.

[34] Renan Vieira et al. "From Reports to Bug-Fix Commits: A 10 Years Dataset of Bug-Fixing Activity from 55 Apache's Open Source Projects". In: PROMISE'19. Recife, Brazil: Association for Computing Machinery, 2019, pp. 80–89.

[35] Rongxin Wu et al. "ReLink: Recovering Links between Bugs and Changes". In: ESEC/FSE '11. Szeged, Hungary, 2011, pp. 15–25.

[36] Rui Xie et al. "DeepLink: A code knowledge graph based deep learning approach for issue-commit link recovery". In: *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2019, pp. 434–444.

[37] Suraj Yatish et al. "Mining software defects: Should we consider affected releases?" In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. 2019, pp. 654–665.

[38] Yu Zhou et al. "Combining text mining and data mining for bug report classification". In: *Journal of Software: Evolution and Process* 28.3 (2016), pp. 150–176.

[39] Thomas Zimmermann et al. "What makes a good bug report?" In: *IEEE Transactions on Software Engineering* 36.5 (2010), pp. 618–643.

[40] Weiqin Zou et al. "An Empirical Study of Bug Fixing Rate". In: *2015 IEEE 39th Annual Computer Software and Applications Conference*. Vol. 2. 2015, pp. 254–263.