

Be Careful of When: An Empirical Study on Time-Related Misuse of Issue Tracking Data

Feifei Tu

School of Electronics Engineering and
Computer Science, Peking University
Key Laboratory of High Confidence
Software Technologies, MoE
China
tufeifei@pku.edu.cn

Jiaxin Zhu

State Key Lab of Computer Science,
Institute of Software, Chinese
Academy of Sciences
University of Chinese Academy of
Sciences
China
zhujiabin@otcaix.iscas.ac.cn

Qimu Zheng

Minghui Zhou*
zheng.qm@163.com
zhmh@pku.edu.cn

School of Electronics Engineering and
Computer Science, Peking University
Key Laboratory of High Confidence
Software Technologies, MoE
China

ABSTRACT

Issue tracking data have been used extensively to aid in predicting or recommending software development practices. Issue attributes typically change over time, but users may use data from a separate time of data collection rather than the time of their application scenarios. We, therefore, investigate data leakage, which results from ignoring the chronological order in which the data were produced. Information leaked from the “future” makes prediction models misleadingly optimistic. We examine existing literature to confirm the existence of data leakage and reproduce three typical studies (detecting duplicate issues, localizing issues, and predicting issue-fix time) adjusted for appropriate data to quantify the impact of the data leakage. We confirm that 11 out of 58 studies have leakage problem, while 44 are suspected. We observe biased results caused by data leakage while the extent is not striking. Attributes of *summary*, *component*, and *assignee* have the largest impact on the results. Our findings suggest that data users are often unaware of the context of the data being produced. We recommend researchers and practitioners who attempt to utilize issue tracking data to address software development problems to have a full understanding of their application scenarios, the origin and change of the data, and the influential issue attributes to manage data leakage risks.

CCS CONCEPTS

• **Software and its engineering** → *Software libraries and repositories; Software maintenance tools;*

KEYWORDS

Issue tracking data; data misuse; data leakage

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5573-5/18/11...\$15.00

<https://doi.org/10.1145/3236024.3236054>

ACM Reference Format:

Feifei Tu, Jiaxin Zhu, Qimu Zheng, and Minghui Zhou. 2018. Be Careful of When: An Empirical Study on Time-Related Misuse of Issue Tracking Data. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18)*, November 4–9, 2018, Lake Buena Vista, FL, USA. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3236024.3236054>

1 INTRODUCTION

Issue tracking systems (ITS) have been extensively used in software projects to track the process of the issues being reported, confirmed, assigned, and resolved. The data produced by ITS (i.e., issue tracking data), particularly the attributes of an issue report, such as what it is (*summary*), who reports it (*reporter*), where it is located (*product/component*), and how important it is (*severity*), are widely used to understand the nature of software quality and to aid in various development practices ranging from triaging reports (e.g., detecting duplicate reports [59], identifying issue severity and priority [66, 67]) to fixing issues (e.g., recommending developers [7], localizing issues [88]) and building Open Source Software (OSS) communities (e.g., how to attract long-term contributors [89]).

However, data users may use data that do not match their intended application scenarios, which may threaten the validity of findings or produce irrelevant conclusions. In particular, data leakage (i.e., data mining models accepting information from the “future” [27]) is known to be one of the top ten mistakes in data mining [47]. It is a logic issue of using data from the future, which is caused by ignoring the chronological order in which data were produced. For example, Rosset et al. described a problematic model that predicts whether a potential customer would buy a product by using deal data that is only available after the products have been purchased [51]. This may constitute the misuse of issue tracking data (ITD) because the values of issue attributes are often changed over time in order to provide more accurate and complete information to aid in issue reproduction and resolution. ITD-based research that fails to recognize these changes may suffer from data leakage. Considering the extensive use of ITD discussed earlier and the lack of research on data leakage in software engineering, data leakage may pose a serious threat to existing ITD-based research results, yet it is still unclear how broad their impact might be.

Therefore, we begin with the research question: Did the use of issue tracking data in existing literature have data leakage (RQ1)? We

believe the data leakage is caused by the change of issue attributes (and the unawareness of the changes), this leads to our second question: How often do issue reports change their attributes during their life cycle and which attributes may change (RQ2)? Further, we investigate the possible impact of the data leakage on research results: If the data leakage is fixed, are the results still valid (RQ3)? Finally, we attempt to answer the following question: If the validity of the results shifts, which attributes would have the largest impact (RQ4)? The answer to the last question allows us to know the most influential attributes to monitor when addressing related problems.

We extensively examined the literature related to (using) ITD to answer our research questions. We searched ITD-related papers which were published from 2000 to 2017 in the DBLP¹ database and obtained 58 papers that were relevant to this study, among which only three definitely do not have the data leakage. The remaining 55 papers cover eleven application scenarios, including issue localization, duplicate report detection, issue assignment, issue-fix time prediction, etc. We confirmed that the eleven papers that made their datasets available have data leakage by manual examination. Specifically, they used retrieved data directly for training and testing, without accounting for the difference between the time when their intended application occurred and the time when the data were retrieved. Because 42 out of the 55 papers targeted Mozilla or Eclipse, we obtained two of their official dumps to conduct further research. We found that 15 of the 18 attributes used in the 55 papers are changeable after the issues being reported, with only three remaining constant (i.e., *reporter*, *report time* and *description*). We selected one paper for each of the three most commonly studied applications from the 55 papers: issue localization, duplicate report detection, and issue-fix time prediction. We reproduced the three studies with non-leaked and leaked data (we obtained the non-leaked data for the application scenarios of the papers by tracing the issue change history), and found that the results are typically biased with leaked data. We applied a method of controlling variables over a number of rounds and found that the *summary*, *component*, and *assignee* attributes create the most variation in results in all three reproduced studies, which highlights their importance when applying ITD to aid in software development.

In summary, we make the following contributions:

- Reveal the data leakage problem when using ITD in the literature.
- Reproduce three existing studies with non-leaked and leaked data and find leaked data cause biased results.
- Find that leaking the information of the *summary*, *component*, and *assignee* of issues has the largest impact on the results.
- Make recommendations for researchers and practitioners who attempt to use ITD to avoid data leakage.

The remainder of this paper is organized as follows. We introduce ITD and their typical applications in Section 2. We describe the methodology in Section 3 and present the results in Section 4. We discuss the implications of our findings in Section 5 and the limitations in Section 6. We introduce related work in Section 7 and make conclusions in Section 8.

2 CONTEXT

2.1 Issue Tracking Data

Software projects often employ ITS (e.g., Bugzilla², JIRA³) to track the submission and resolution of various kinds of issues, including bugs, feature requests, etc. Each issue report has a number of attributes describing what it is (*summary*, *description*), how important it is (*severity*, *priority*), where it is found (*product*, *component*, *version*), which step it is in (*status*), who is resolving it (*assignee*), etc. During the course of resolution, an issue's attributes may be modified [48, 90]. For example, when confirmed, the *status* of an issue will be changed from *UNCONFIRMED* to *NEW*; when completed, it will be changed to *RESOLVED*; when people find the *summary*, *product*, or *component* of the issue is inaccurate, incorrect, or incomplete, they may revise or complete the information; when the responsibility of fixing the issue shifts to a different developer, the *assignee* will be changed. As a result, the data retrieved at a specific time may be different from the data retrieved at other time [90].

2.2 Applications of ITD

In this section, we present three typical applications that we investigate and reproduce in this study: issue localization, duplicate issue report detection, and issue-fix time prediction.

2.2.1 Issue Localization. In order to fix reported issues, localizing the source code files that need to be modified is an inevitable step [88]. In the literature, one common approach to issue localization is to measure the text similarity between issue reports and source code [42, 53, 88]. Using Eclipse Issue#80720 [88] as an example, its *summary* is “**Pinned console** does not remain on top”, and its *description* is “Open two **console views**, ... Both **consoles display** the last launch. The **pinned console** should remain **pinned**.” The *summary* and *description* of the issue report are compared with all the code files, and the files with the highest similarity are selected as candidates. The source code corresponding to Issue#80720 includes the terms occurring in the *summary* and *description* (e.g., “*public class ConsoleView extends PageBookView*,” “*display(console)*,” and “*setPinned(true)*”).

2.2.2 Duplicate Issue Report Detection. Contributors may report the same issues multiple times in the issue tracking system. These duplicate issue reports may incur extra maintenance effort in triaging and fixing issues. It is essential to determine if an issue report is a duplicate of others. Existing solutions use issue attributes (e.g., *summary* and *description*) to compare new reports with existing reports and detect duplicates [19, 37, 59, 60, 62, 73].

2.2.3 Issue-Fix Time Prediction. Predicting issue-fix time is useful in several areas of software evolution, such as predicting software quality [29] or coordinating effort during issue triaging [21]. Developers are often asked to estimate the amount of time they will need to fix specific issues to aid the project planning process. Accurate estimation of task completion time allows project planners and managers to effectively schedule releases and allocate effort to meet those targets. Because developers spend much of their time on legacy code and maintenance tasks, it is important to be able to estimate the required time and effort for fixing issues or completing enhancement tasks. Issue-fix time has been modeled with various

¹<https://dblp.uni-trier.de>

²<https://www.bugzilla.org/>

³<https://www.atlassian.com/software/jira>

Table 1: ITD of Mozilla and Eclipse

Name	Author	# of issues	Last report time	URL
Mozilla2016	Mozilla Community	1160599	2016-03-03 13:08:06	github.com/jxshin/mzdata/tree/master/data/2016/level0
Eclipse2010	MSR Mining Challenge	316911	2010-06-25 23:05:19	2011.msrconf.org/msr-challenge.html

issue attributes [4, 10, 31, 48, 75], including *product*, *component*, *severity*, etc.

3 METHODOLOGY

We proposed four research questions covering the following aspects: confirming the existence of data leakage that use ITS from the future scenarios rather than the application scenarios (RQ1), observing the extent of changes in ITD (RQ2), quantifying the impact of data leakage (RQ3), and determining the influential attributes (RQ4). We describe the methods of answering these questions in this section.

3.1 Confirmation of Data Leakage (RQ1 and RQ2)

3.1.1 Examination of Literature. We examine the literature in the DBLP database to answer RQ1. We first used the keywords *bug report* and *issue report*⁴ to search for papers published between 2000 and 2017 in the DBLP database. After obtaining results, we looked up into the references of target papers and examined whether they were relevant. If it was, it would be included in the collection of relevant papers, and their references would be examined. The process was repeated till new relevant paper was not found.

We initially obtained 319 papers. We then manually examined them to find studies of interest. To ensure the quality of papers, we selected 110 papers that were published in main software/data conference proceedings and journals, including FSE, ICSE, ASE, AAAI, CSCW, CHI, SANER, ICSM, ICSME, MSR, CSMR, WCRE, ICPC, SEKE, TSE, TOSEM, IST, ESE, and JSS. Among the 110 papers, 67 papers were marked as irrelevant to this study. They fall into one of three categories: 1) qualitative studies that do not use ITD for prediction or recommendation; 2) studies on commercial projects without publicly available datasets; 3) studies on application scenarios that are time-irrelevant (e.g., linkage recovering, where it does not matter when the linkage between the issue report and the source code was recovered because the application scenario could happen at any time regardless of whether the issue was newly reported or already resolved). Through the references of the remaining 43 papers, we found another 15 relevant papers. In total, we obtained 58 relevant papers on predictions or recommendations based on OSS projects.

We explore whether the attributes used in these papers are changeable in order to identify whether they may suffer from data leakage. We carefully examined whether the 58 papers used the non-leaked data (or were aware of the existence of changing data), and whether they conducted the necessary data preprocessing to extract data that matched their applications for later analysis. We divided the problematic papers into “definitely problematic” and “potentially problematic” groups according to whether or not the paper provides dataset. If a dataset is provided, we can confirm

the existence of data leakage by examining the dataset.⁵ The paper can then be marked as “definitely problematic” if data leakage is confirmed. If the dataset is not provided, we cannot confirm the existence of data leakage. But we suspect the paper may have data leakage if the measures are not taken to avoid data leakage. The paper is then marked as “potentially problematic.” We report the results in Section 4.1.

3.1.2 Quantification of the Change Rate and Change Frequency of Issue Reports. We quantify the change rate and change frequency of issue reports to answer RQ2. First, we collect the ITD of relevant projects for our investigation. Because the ITD of Mozilla and Eclipse are the most commonly used data in the literature (as mentioned in Section 1, the majority of the inspected papers used these sources), we conduct our investigation on two of their official data dumps: *Mozilla2016* and *Eclipse2010*⁶, as summarized in Table 1.

Second, in order to reveal the relevance of each issue attribute, we investigate how widely an attribute is used (i.e., the number of papers using it).

Third, we determine if the attributes are changeable and calculate the change rate and change frequency of each changeable attribute. Change rate is calculated as:

$$\text{ChangeRate}(attr) = \frac{|I_{attr}|}{|I|} \quad (1)$$

where I represents the set of all resolved issue reports. I_{attr} represents the set of resolved issue reports that had the attribute $attr$ changed. Change frequency is calculated as:

$$\text{ChangeFrequency}(attr) = \frac{\sum_{i \in I_{attr}} T_{attr_i}}{|I_{attr}|} \quad (2)$$

where T_{attr_i} represents the number of changes on attribute $attr$ of issue report i . Taking *assignee* for example, Mozilla2016 has 1,005,351 issue reports that have been resolved. Among them, 395,731 issue reports’ *assignee* have been changed. *Assignee* has been changed 534,992 times for these 395,731 issue reports. Therefore, the change rate of *assignee* in Mozilla is 39.36% (395,731/1,005,351), and the change frequency is 1.3519 (534,992/395,731). We report the results in Section 4.1.

3.2 Quantification of Impact (RQ3)

In order to answer RQ3, we select several representative studies from the relevant papers and reproduce their models (which are used to make predictions or recommendations) with non-leaked and leaked data for comparison.

We classify the papers we obtained into different categories according to the application scenarios they intended to address, as reported in Section 4.1. From each of the most studied categories, we choose one paper to investigate. The choice of paper is based on

⁴Previous studies may mix *issue report* and *bug report* [18]. Thus, in order to cover a broader scope, we use both *bug report* and *issue report* as keywords.

⁵It is important to note that the effort spent on providing dataset is highly appreciated which helps to examine the possible problem.

⁶This dump provided by MSR Mining Challenge is widely used by other studies [16, 20, 74].

Table 2: Distribution of Definitely & Potentially Problematic Studies on Different Applications

Application	Definitely problematic papers	# of definitely problematic papers	Potentially problematic papers	# of potentially problematic papers
Issue Localization	[88] [28] [82] [53] [72] [77] [30] [84]	8	[39] [45] [15] [83]	4
Duplicate Detection	[59] [46]	2	[73] [60] [58] [14] [25] [69] [37] [38] [62] [35] [1] [19] [36] [81]	14
Issue Assignment	[23]	1	[7] [6] [9] [24] [40] [61] [49] [8] [80] [55] [56] [86] [70]	13
Issue-Fix Time Prediction	–	0	[75] [31] [4] [10]	4
Priority Prediction	–	0	[67] [68]	2
Severity Prediction	–	0	[33] [66]	2
API Recommendation	–	0	[65]	1
Classes Recommendation	–	0	[3]	1
Packages Recommendation	–	0	[22]	1
Configuration Bugs Prediction	–	0	[76]	1
Blocking Bugs Prediction	–	0	[71]	1
Total	–	11	–	44

three criteria: 1) the paper is the most referenced in the category to which it belongs, meaning its results have a wider audience; 2) the paper provides sufficient implementation details or executable programs for us to reproduce; 3) the paper does not include leaked factors in its model, e.g., the number of developers who have ever been assigned to an issue in its life cycle when predicting issue-fix time at submission.

We reproduce the investigated studies with non-leaked and leaked data respectively, and then compare the results (i.e., the performance of the models proposed in the papers) to evaluate the impact of the data leakage. From the database dumps introduced in Table 1, we extracted two versions of issue data for each issue. The data when an issue was reported, we call it *initial data*. The data when the dataset was retrieved, we call it *modified data* (due to the fact that most of issue reports are changed in modified data). Using modified data introduces data leakage while using initial data does not, because the intended scenario of these investigated studies is at the time when the issues had not been resolved. Modified data are extracted directly from MySQL dumps, while initial data are recovered from the modified data by tracing the issue change history. Consider Issue#117508 in Eclipse as an example. In the data used by [88], its *summary* is “Cannot commit empty folder using Team/Commit,” but this was not the *summary* when Issue#117508 was submitted. From the change history in *Eclipse2010*, we find that the earliest change of *summary* of Issue#117508 happened at 2005-11-24 10:45:35 EST, and that the initial *summary* was “COMMIT OF NEW EMPTY FOLDER DOES NOT WORK FROM NAVIGATOR, DOES WORK FROM SYNCHRONIZE WINDOW.” At this point, we obtain the initial and modified *summary* of Eclipse Issue#117508. The recovering process of other attributes is the same as *summary*. We recover every attribute to its value at the time of submission and we obtain complete initial data of each issue report.

If the paper provides programs/scripts that implement and evaluate its model, such as the paper on duplicate detection [59], the reproducing process is simple: we reuse the programs/scripts, switch the input between non-leaked and leaked data, and obtain results for comparison. If the paper only provides programs/scripts that implement their approach, such as the paper on issue localization [88], we reuse the programs/scripts to obtain the output of the model and calculate its performance using the metrics described in the paper, such as precision and recall. If the paper does not provide

any programs/scripts at all, such as the paper on issue-fix time prediction [31], we read the details of the approach and implement it by ourselves. We report the results in Section 4.2.

3.3 Investigation of the Most Influential Attributes (RQ4)

To answer RQ4, we analyze the effect of correcting the value of each attribute on the results of the investigated studies, thereby determining the most influential attributes.

Evaluations of the degree to which predictors (i.e., the measured aspects within software development) contribute to the performance of prediction/recommendation models are commonly conducted to aid people in understanding how the models work [13]. In data-driven software engineering studies, such evaluations are conducted to determine which aspects of development can be adjusted to obtain better practice [13]. Similarly, in order to understand the impact of change in ITD attributes on models and guide people to develop more robust ITD-based applications, we examine each attribute used in the literature. We explore the impact of each attribute by observing the differences in study results after replacing leaked data with non-leaked data. Specifically, the differences are revealed through a method of controlling variables, which is one of the most commonly used approaches in impact analysis [5, 52, 57]. For each investigated paper, we reproduce the study a number of times (the number of changeable attributes used in the reproduced paper). The data used for reproduction is different each time: we choose one changeable attribute and set its value to that used in the non-leaked data and keep (i.e. control) the other attributes as what they are in the leaked data. We observe the extent to which the results change compared to those derived from fully leaked data (i.e., the partial results of RQ3). The bigger the difference caused by setting the value of the attribute is, the larger the impact of the leaking information of the attribute is. For example, the paper on duplicate detection [59] used four changeable attributes: *summary*, *component*, *product* and *priority*. This study would be reproduced four times. In the data for first reproduction, the value of *summary* is set to that when the reports were submitted, and the values of other attributes are those when the reports were resolved. In the other three rounds, the values of *component*, *product*, and *priority* are set respectively. At the end of each round, we calculate the performance difference between the model reproduced in

that round ($Perf_{init}$) and the model reproduced with fully leaked data ($Perf_{mod}$), written as $(\frac{Perf_{mod}}{Perf_{init}} - 1)$. We report the results in Section 4.3.

4 RESULTS

4.1 Data Leakage in the Literature

Using the method described in Section 3.1.1, we obtained 58 papers and determined whether they used leaked data. Among the inspected papers, only three mentioned that issue attributes change during their life cycle [32, 48, 54] though they did not provide datasets. The other 55 papers cover various applications of ITD, including issue localization, duplicate report detection, prediction of issue-fix time, priority and severity, issue assignment, API recommendation, classes recommendation, packages recommendation, configuration bugs prediction and blocking bugs prediction, as shown in Table 2. We confirmed that eleven papers have data leakage. In their application scenarios, the issue reports should not have been resolved. However, all of them used modified data⁸ where most of the issues had already been resolved to train/fit and test the prediction/recommendation models. Forty-four papers neither provided datasets nor mentioned that issue attributes could be changed, thus potentially have data leakage.

Using the method described in Section 3.1.2, we observed the change rate and change frequency of the issue attributes. In the inspected papers, the attributes used in the models vary from textual attributes (e.g., *summary*, *description*) to structured attributes (e.g., *severity*, *priority*). We summarized the attributes as shown in Table 3. The first column lists the 18 issue attributes used by the selected papers. The final column explains these attributes, with most descriptions taken from the Bugzilla official document.⁹

In Table 3, the attributes are ranked according to the number of papers that used them. Among the 18 attributes, *description* and *summary* are the most commonly used attributes. Forty-eight papers used *description* and forty-six papers used *summary*. This may be because *summary* and *description* are textual attributes which provide informative details regarding the issue (for any use). The occurrence of other attributes varies widely from 1 to 15 uses. *Component*, *product*, *priority*, *severity*, *version*, and *comment* also attract significant attention and are used in various applications, such as duplicate issue report detection [38], priority prediction [67], and blocking bugs prediction [71]. The rest are used only occasionally. For example, *target milestone* is used in [49] to help assign issues to developer, *keywords* is used in [38] to detect duplicate issue reports, and *assignee* is used in [31] to predict issue-fix time.

Except for *reporter*, *report time* and *description*, all the attributes are changeable during the life cycle of an issue (see the third column of Table 3). The fourth and fifth columns of Table 3 show the change rate of each attribute in *Mozilla2016* and *Eclipse2010*, respectively. The sixth and seventh columns show the change frequency of each attribute. Change rate and change frequency vary among the attributes. Several attributes have high change rate, such as *comment*,¹⁰ which was changed in 99% of the

resolved issue reports. It is also modified repeatedly (8.4384 average modifications for Mozilla and 4.7831 for Eclipse). *OS* and *platform* are rarely modified and have low change rates and frequencies. Their change rates are less than 10% for both Mozilla and Eclipse. Their change frequencies are no more than 1.1, suggesting that among all issue reports whose *OS* or *platform* have been changed, less than 10% of issues would have them changed twice. The other attributes' change rates vary from 10% to 76%, while their change frequencies vary from 1.1 to 3. For example, almost one fifth of the issues are triaged to incorrect components initially. On average, more than 20% of changed issues are triaged incorrectly more than once (28% for Mozilla, 23% for Eclipse according to the change frequency).

In summary, we have confirmed that the data leakage exists in the literature and at least 11 out of the 55 related papers have the problem, i.e., they use modified data for both training and testing. These papers cover a wide range of applications. We found that 15 out of the 18 issue attributes are changeable. The change rate can be higher than 90% (e.g., *comment*), and for all attributes other than *severity*, *OS*, and *platform*, it is over 10%, either in Mozilla or Eclipse. The change frequency is larger than 1 for all changeable attributes, meaning all of the changeable attributes may be modified many times. It suggests that the results of the studies could be biased because of data leakage. Therefore, the impact of the data leakage should be examined further.

4.2 Biased Results

Based on the criteria described in Section 3.2, we select one paper from each of the three most popular application categories, which are issue localization, duplicate detection and issue-fix time prediction. We skip issue assignment because the practice of issue assignment in OSS projects requires further investigation.¹³ Table 4 summarizes the three papers we chose to reproduce, including the problems they tried to address, the data they employed, the issue attributes they used, their approaches and evaluation measures, and their status of data leakage (definitely or potentially problematic).

Issue localization. [88] ranked the files related to an issue through an information retrieval method that calculates the similarity between the files and the *summary* and *description* attributes of the issue report. Using the data from Eclipse, the authors evaluated their model through the commonly used measures of mean reciprocal rank (MRR), mean average precision (MAP) [2], and recall rate at ranking top-k (recall rate@k).

Table 5 and Figure 1 present the results of reproducing the study [88] with non-leaked (initial) and leaked (modified) data. Figure 1 shows the recall rate@k for Eclipse. "*init*" represents the results with initial data and "*mod*" represents modified data. The experiment with modified data outperforms the experiment with initial data in terms of recall@k for all k. With initial data, the MRR and MAP of the returned ranking file list are 0.39893 and 0.24767, respectively. With modified data, the MRR and MAP grow to 0.42458 and 0.26303. Thus, the leaked data result in a boost of

⁸Four of them provided experimental datasets that can be verified [28, 59, 82, 88], while the other seven reused their datasets [23, 30, 46, 53, 72, 77, 84].

⁹<https://bmo.readthedocs.io/en/latest/using/understanding.html>

¹⁰The content of submitted comments is constant, and what varies is that new comments are added as time goes by.

¹³It should be noted that 12 out of the 14 issue assignment papers studied Eclipse, and 11 out of 12 papers that studied Eclipse aimed to recommend an individual developer for an issue report. However, we found that almost one fifth of *assignee* in Eclipse are not individual developers, but mailing lists of subscribers (e.g., platform-help-inbox@eclipse.org). Thus, the studies regarding issue assignment must be reconsidered, and we were unable to choose an appropriate paper.

Table 3: 18 Attributes in 55 Investigated Studies

Attributes	# of occurrences	Could it be changed?	Change rate%		Change frequency		Interpretation
			Mozilla	Eclipse	Mozilla	Eclipse	
<i>Description</i>	48	F	–	–	–	–	Detailed outline of the issue, such as what is the issue and how it happens.
<i>Summary</i>	46	T	16.49	20.16	1.2912	1.2088	A one-sentence summary of the problem, displayed in the header next to the issue number.
<i>Component</i>	15	T	22.83	17.45	1.2811	1.2310	Issues are divided up by <i>product</i> and <i>component</i> , with a <i>product</i> having one or more <i>components</i> in it.
<i>Product</i>	13	T	30.59	6.62	1.2050	1.1317	
<i>Priority</i>	12	T	9.70	13.42	1.1889	1.2323	The <i>priority</i> attribute is used to prioritize issues, either by the <i>assignee</i> , or someone else with authority to direct their time such as a project manager. The default values are P1 to P5.
<i>Severity</i>	9	T	7.10	9.06	1.1432	1.1102	The <i>severity</i> attribute indicates how severe the problem is – from blocker ("application unusable") to trivial ("minor cosmetic issue"). You can also use this field to indicate whether an issue is an enhancement request.
<i>Version</i>	9	T	10.49	6.95	1.1404	1.1820	The <i>version</i> attribute usually contains the numbers or names of released versions of the product. It is used to indicate the version(s) affected by the issue report.
<i>Comment</i>	9	T	99.06	99.95	8.4384	4.7831	Discussion about the issue.
<i>Reporter</i>	6	F	–	–	–	–	The person who reported the issue.
<i>Report Time</i>	6	F	–	–	–	–	The data and time when the issue is reported.
<i>OS</i>	5	T	9.55	2.70	1.0516	1.0262	These indicate the computing environment where the issue was found.
<i>Platform</i>	4	T	8.05	1.39	1.0348	1.0171	
<i>Type</i> ⁷	4	T	–	–	–	–	The type of the report, i.e., defect, task, feature.
<i>Assignee</i>	2	T	39.36	58.72	1.3519	1.4778	The person responsible for fixing the issue.
<i>Keywords</i>	2	T	19.14	6.35	1.9612	1.1674	The administrator can define keywords which you can use to tag and categories issues, e.g., crash or regression.
<i>Attachments</i>	1	T	36.00	28.00	2.6574	1.8663	Attached files (e.g. test cases or patches) to issues.
<i>CC List</i>	1	T	76.72	46.85	3.4213	2.0520	A list of people who get mail when the issue changes, in addition to the Reporter, Assignee and QA Contact.
<i>Target Milestone</i>	1	T	24.61	56.22	1.4210	1.5009	A future version by which the issue is to be fixed.

⁷ *Type* is an attribute that is used in JIRA, not in Bugzilla. So the issue reports of Mozilla and Eclipse do not have the *Type* attribute, and we cannot obtain its change rate and change frequency.

Table 4: Summary of the Three Reproduced Studies

Application	Data	Approach	Evaluation Measurements	Attributes	Data Leakage
Issue Localization[88] ¹¹	Eclipse	TF-IDF cosine similarity	MRR MAP recall rate@k	<i>Summary/Description</i>	definitely problematic
Duplicate Detection[59] ¹²	Eclipse Mozilla	BM25F _{ext}	MAP recall rate@k	<i>Summary/Description/Product/Component/Version/Priority/Type</i>	definitely problematic
Issue-Fix Time Prediction[31]	Eclipse Mozilla	Naive Bayes	AUC	<i>Component/ Priority/Severity/Platform/OS/Reporter/Assignee/Reported date(year/month/day)</i>	potentially problematic

¹¹ The experimental data (i.e., modified data) was downloaded from <https://code.google.com/archive/p/bugcenter/wikis/BugLocator.wiki>

¹² The experimental data (i.e., modified data) was downloaded from <https://www.comp.nus.edu.sg/~specmine/suncn/ase11/index.html>

The text fields (*summary* and *description*) in the experimental data extracted by the authors were processed as dictionary data, and cannot be recovered to textual data, so they cannot be used in our comparative study. Thus, we extracted the initial and modified data from Eclipse2010 and Mozilla2016, which include the same issues as the original data used in the papers.

Table 5: Performance (MRR and MAP) for Issue Localization [88] (Eclipse)

Eclipse	MRR	MAP
Initial data	0.39893	0.24767
Modified data	0.42458	0.26303

Table 6: Performance (MAP) for Duplicate Detection [59] (Mozilla and Eclipse)

MAP	Mozilla	Eclipse
Initial data	0.226127	0.312378
Modified data	0.247306	0.323553

performance in the issue location model. Notice that the increase of developer effort may be much more significant than the performance shift may reveal. We retrieve the ranks of the target files (i.e., the file that corresponds to the issue report) for each issue in both models and calculate their differences. After filtering out the highest and lowest 1% of the rank difference, the mean value is +73. This suggests that, on average, developers must examine 73 more irrelevant files to find the target file. For example, when the leaked

data are used to localize buggy file for Eclipse Issue#97644, the target file *org.eclipse.ant.internal.ui.debug.AntSourceContainer.java* ranks #1, while it ranks #77 when the initial data are used. This means that, when using the issue localization model in [88] to find related files for Eclipse Issue#97644, developers have to check 76 irrelevant files before they find the target file. For nearly 10% of issue reports, the drop in rank is larger than 100. Another case is Eclipse Issue#94497, where the difference between rankings is much larger. The target file *org.eclipse.ui.internal.browser.BrowserLauncher.java* ranks #26, while it ranks #1450 when the initial data are used. This suggests that this method would encounter great difficulty for Eclipse Issue#94497 in practice.

Duplicate issue report detection. [59] ranked potentially duplicated issue reports for newly submitted issues using an extended BM25F algorithm [50, 85]. In their model, they detected duplicate reports by comparing the *summary*, *description*, *product*, *component*, *version*, and *type* attributes of issue reports, and then ranking the most similar issue reports. They evaluated their model using MAP and recall rate@k.

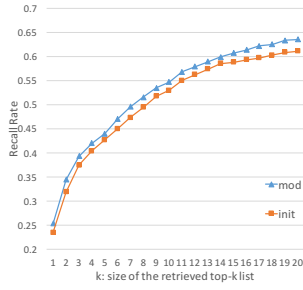


Figure 1: Performance (recall@k) for Issue Localization [88] (Eclipse)

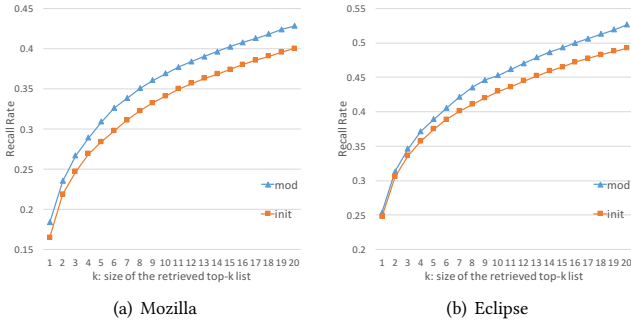


Figure 2: Performance (recall@k) for Duplicate Detection [59] (Mozilla and Eclipse)

The MAP increases from 0.226127 to 0.247306 for Mozilla, and from 0.312378 to 0.323553 for Eclipse, respectively, when we replaced the initial data with the modified data (Table 6). The impact is larger in Mozilla than in Eclipse, which implies that the degree of differences in the attribute values between the initial and modified data in Mozilla may differ from that in Eclipse. This may be the result of different users. Mozilla’s products are aimed at ordinary Internet users, while Eclipse targets developers with professional software development knowledge. This may explain why issue reports created by Mozilla’s users are of lower quality. Figure 2 shows the recall rate@k of [59] for Mozilla and Eclipse. For both Mozilla and Eclipse, the top-k recall rate of “init” is consistently lower than “mod” for any k from 1 to 20. It should be noted that a performance drop in a duplicate detection model can be directly translated into an efficiency drop in practical software development. For Mozilla, the 3% drop in recall rate at top-20 means that 202 (3% of 6725) duplicate reports will not be detected after examining the top 20 potential duplicate candidates. For Eclipse, the 3.5% drop in recall rate at top-20 means that 101 (3.5% of 2880) duplicate reports will not be detected after examining the top 20 potential duplicate candidates.

Issue-fix time prediction. [31] proposed using a preprocessing step to filter issue reports to improve the performance of issue-fix time prediction. They filtered issues that were fixed too quickly and used a Naive Bayes model to compare the pre-filtering and post-filtering results. They evaluated the preprocessing step using area under the RoC curve (AUC), which is a measurement that considers precision and recall simultaneously [12].

Table 7 presents the AUC of the models obtained for Mozilla and Eclipse when we reproduce [31]. The first column shows the

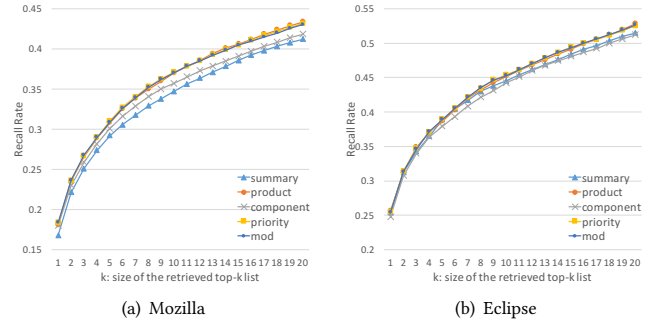


Figure 3: Impact of Attributes (recall@k) on Duplicate Detection [59] (Mozilla and Eclipse)

Table 7: Performance (AUC) for Issue-Fix Time Prediction [31] (Mozilla and Eclipse)

Project	Initial data		Modified data		Improved with filtering?	
	Before filtering	After filtering	Before filtering	After filtering	Initial data	Modified data
Mozilla Core	0.720	0.712	0.740	0.744	-	+
Mozilla Bugzilla	0.926	0.941	0.953	0.958	+	+
Mozilla Firefox	0.680	0.699	0.662	0.701	+	+
Mozilla Thunderbird	0.614	0.618	0.646	0.666	+	+
Mozilla Seamonkey	0.905	0.886	0.886	0.898	-	+
Eclipse Platform	0.702	0.703	0.685	0.691	+	+
Eclipse PDE	0.663	0.661	0.655	0.659	-	+
Eclipse JDT	0.650	0.662	0.667	0.672	+	+
Eclipse CDT	0.776	0.799	0.755	0.776	+	+
Eclipse GEF	0.640	0.655	0.648	0.666	+	+

Table 8: Impact of Attributes (MAP) on Duplicate Detection [59] (Mozilla and Eclipse)

MAP	Modified data	Initial data			
		summary	product	component	priority
Mozilla	0.247306	0.230845	0.24589	0.240918	0.24742
($\frac{Perf_{mod}}{Perf_{init}} - 1$)%		7.130759	0.575867	2.651525	-0.04608
Eclipse	0.323553	0.321243	0.324447	0.316071	0.32418
($\frac{Perf_{mod}}{Perf_{init}} - 1$)%		0.7191	-0.2756	2.3672	-0.1934

studied sub-projects.¹⁴ The second and third columns are results based on initial data, with the former being the results without the data preprocessing step and the latter being those with data filtering. The fourth and fifth columns are similar to the second and third columns, but they are based on the modified data. The sixth and seventh columns indicate whether the model performance is improved with data filtering when using the initial data and the modified data. When using the modified data, the data filtering method works on all the sub-projects, with a rate of 100%. However, when using the initial data to reproduce their model, for three out of the ten studied sub-projects in Mozilla and Eclipse, we found that the filtering does not improve the results. For example, for Mozilla Core, the AUC of the initial data with data filtering (0.712) is lower than the AUC without filtering (0.720). The same phenomenon occurs for Mozilla Seamonkey and Eclipse PDE.

In summary, we have found that using leaked data in common ITD applications (e.g., issue localization, duplicate detection and

¹⁴Because [31] was conducted using sub-projects of Mozilla and Eclipse, we reproduced the study with the same sub-projects.

Table 9: Impact of Attributes (AUC) on Issue-Fix Time Prediction [59] (Mozilla and Eclipse)

Project	Modified data	Initial data (AUC / ($\frac{Perf_{mod}}{Perf_{init}} - 1$))					
		Assignee	Component	OS	Platform	Priority	Severity
Mozilla Core	0.744	0.711/4.6414	0.752/-1.0638	0.743/0.1346	0.744/0	0.741/0.4049	0.743/0.1346
Mozilla Bugzilla	0.958	0.938/2.1322	0.957/0.1045	0.96/-0.2083	0.96/-0.2083	0.957/0.1045	0.958/0
Mozilla Firefox	0.701	0.677/3.5451	0.675/3.8519	0.704/-0.4261	0.704/-0.4261	0.701/0	0.701/0
Mozilla Thunderbird	0.666	0.619/7.5929	0.668/-0.2994	0.666/0	0.668/-0.2994	0.665/0.1504	0.664/0.3012
Mozilla Seamonkey	0.898	0.887/1.2401	0.898/0	0.898/0	0.898/0	0.898/0	0.898/0
Eclipse Platform	0.691	0.723/-4.4260	0.691/0	0.693/-0.2886	0.69/0.1449	0.673/2.6746	0.682/1.3196
Eclipse PDE	0.659	0.671/-1.7884	0.658/0.1519	0.659/0	0.659/0	0.652/1.0736	0.653/0.9188
Eclipse JDT	0.672	0.685/-1.8978	0.67/0.2985	0.675/-0.4444	0.672/0	0.659/1.9727	0.655/2.5954
Eclipse CDT	0.776	0.806/-3.7221	0.765/1.4379	0.776/0	0.776/0	0.775/0.1290	0.773/0.3881
Eclipse GEF	0.666	0.662/0.6042	0.667/-0.1499	0.659/1.0622	0.67/-0.5970	0.664/0.3012	0.645/3.2558

issue-fix time prediction) may lead to biased evaluation of model performance. This implies that the impact of data leakage cannot be ignored.

4.3 Most Influential Attributes

Using the method described in Section 3.3, we investigate the impact of misusing the value of each issue attribute that is changeable within the three investigated studies. The changeable attributes in the three studies have been marked with bold text in Table 4.

Issue localization. In the issue localization model [88] there is only one changeable attribute: *summary*. Table 5 shows the performance decrease caused by recovering the value of *summary*. In Section 4.2, the examples of Eclipse Issue#97644 and Issue#94497 demonstrate that the results are obviously different when initial rather than modified data are used. The initial *summary* of Issue#97644 is “Source lookup fails for linked resources,” while the modified *summary* is “Source lookup can make use of fully qualified file information from Ant.” The additional information of the modified *summary* includes “from Ant.” “Ant” is exactly the package name of the target file (org.eclipse.ant.internal.ui.debug.AntSourceContainer.java). The initial *summary* of Issue#94497 is “Cannot open help on Solaris,” while the modified *summary* is “[Browser] Cannot open help on Solaris.” The only difference between the two *summary* values is “Browser,” which happens to be the package name of the target file (org.eclipse.ui.internal.browser.BrowserLauncher.java). Both of the *summary* modifications add valuable information for localizing the issue.

Duplicate issue report detection. Based on the evaluation of the MAP (Table 8) and recall rate@k (Figure 3), the degree of the differences caused by recovering the values of the changeable attributes ranks as: *summary* > *component* > *product* > *priority*. In Figure 3, “mod” represents the results with modified data, “summary” stands for the result where the value of *summary* is the initial value and the values of other attributes are the modified values. The labels “product,” “component,” and “priority” indicate the same for their respective attributes. When an issue is reported, an ITS typically provides options and guides the reporter to choose the right product. Thus, the product has a high chance to be correctly assigned on the first try.¹⁵ The *priority* attribute is often assigned a default value and is rarely modified. These phenomena may explain why leaking the information of *product* and *priority* has only a

minor impact. *Component* has a higher likelihood of being changed (higher change rate and change frequency, see Table 3), meaning it is difficult to assign an issue to the correct component. Thus, the impact of misusing the value of *component* is prominent. The largest impact is caused by misusing the value of *summary*. This is illustrated by the following example. Mozilla Issue#622309 reflects the fact that a duplicate report cannot be detected due to its inappropriate *summary*. Its modified *summary* is “leading/trailing space ... displays it wrong ... in address book.” Its initial *summary* is “double quotes cause email rejection,” which is rather short and does not provide sufficient information. The comment on #622309, “Details matter: More precise summary (the problem starts with trailing space, not with the quotes) reveals that this is a dupe of bug 286760,” also reveals that the information (“trailing space”) provided in the modified *summary* is the key to detecting the duplicate. It is clear that Issue#622309 could not be detected as a duplicate due to its insufficient initial *summary*. These results demonstrate the importance of an issue report’s *summary* and *component* attributes for detecting duplicates. The quality of an issue report’s *summary* and *component* is relevant to the performance of duplicate detection, which should be of interest to developers.

Issue-fix time prediction. The difference between issue-fix time prediction reproduced with and without leakage on each attribute varies across the sub-projects of Mozilla and Eclipse (see Table 9). In particular, the recovery of *assignee* causes prominent differences (at least twice as much as the other attributes) in five sub-projects (Mozilla Core, Mozilla Bugzilla, Mozilla Thunderbird, Eclipse Platform and Eclipse CDT), while *component* impacts three projects (Mozilla Core, Mozilla Firefox and Eclipse CDT), *priority* impacts three projects (Eclipse Platform, Eclipse PDE and Eclipse JDT), and *severity* impacts three projects (Eclipse Platform, Eclipse JDT and Eclipse GEF). To summarize, the ranking of the degree of differences caused by recovering the attributes is: *assignee* > *component*, *priority*, *severity* > *OS*, *platform*. In the initial data, two thirds of issues are assigned to public ids, such as xxx-inbox@eclipse.org or xxx-triaged@eclipse.org, while in the modified data there are only one fourth of the issues were assigned to these public ids. Thus, when modified data are used to predict issue-fix time, most of the *assignees* are individual developers. This might be the reason why *assignee* has large impact. The *component* is important because, first, *component* is more likely to be changed (higher change rate and change frequency). Thus, when data are corrected, a high portion of issue reports are revised for this attribute that consequently affects the results. Second, issue-fix time is relevant to a specific

¹⁵In Mozilla, frequent renaming and merging of *product* attributes lead to the high change rate on *product*. Actually, the change rate drops to 17% after excluding these special cases.

component. For example, issues with the “*cdt-other*” component will take a long time to be resolved possibly because these issues are not relevant to the core features of CDT (so developers do not rush to address them). As discussed earlier, *os* and *platform* are the two attributes with lowest change rate and frequency. *Priority* and *severity* are modified more frequently than *os* and *platform*. Therefore, the ranking of the impact of these four attributes is consistent with their likelihood of being changed.

In conclusion, we have found that *summary*, *component*, and *assignee* are the most influential attributes in the common ITD applications (issue localization, duplicate detection and issue-fix time prediction). These attributes have a prominent impact on the prediction/recommendation models and they are often modified during the life cycle of issue reports.

5 DISCUSSION

For both researchers and practitioners who attempt to build and/or use models on ITD, it is important to recognize that a successful model has two key requirements: a) the selected model has the ability to capture patterns in software development, and b) the data is of sufficient quality to reflect the nature of the issues and the intended scenario. The selection of a proper model has been widely addressed, but improving quality of ITD has not received enough attention that it deserves.

5.1 Potential Risks

The problem brought by using leaked data has two aspects. First, such data usage is illogical. Although the models can be trained with leaked data, the model trained with leaked data cannot be used in practice as how the researchers intend, and it cannot be interpreted either, because the training data come from the future, i.e., predicting unknown via unknown. Second, data leakage has high potential risks. Among the 55 relevant studies, at least one fifth studies using ITD have data leakage. It means that the possibility of running into data leakage could be high. Meanwhile, we confirm that the data leakage can lead to biased results and make previous results difficult to reproduce. Although we did not observe that the performance of the models had a drastic change after fixing the leakages in the investigated studies, in some cases, from other perspectives, small changes may lead to totally different conclusions. For example, the investigation of issue-fix time prediction study [31] shows that, in practice, the filtering approach may not work as when it was proposed with data leakage. Therefore, the potential risks of data leakages in using ITD should not be ignored.

5.2 Recommendations for Researchers

When using ITD, researchers should carefully consider the dynamic nature of issue attributes and their intended application scenarios.

When researchers utilize ITD to target an application scenario, first, they should explicitly state the time information for each used attribute in the description of their models, second, they may consider the rules provided below to check whether their data are leaked and recover the non-leaked data through the method described in Section 3.2 if it is needed.

- An effective duplicate report detection scheme should prevent a duplicate report from being confirmed as a valid issue

that needs to be fixed. Therefore, detection should be conducted before the issue report is changed to the NEW status.

- A variety of application scenarios occur during the period of issue triage [78, 79], including issue assignment, issue-fix time prediction, priority prediction, severity prediction, configuration bugs prediction and blocking bugs prediction. As a result, they should be accomplished before issues get to the ASSIGNED status.
- Issue localization, API recommendation, package recommendation, and class recommendation happen during the period of issue resolution, which occurs before issues are marked with the RESOLVED status.

In this study, we confirmed that eleven papers have data leakage, among which, four [28, 59, 82, 88] provide the datasets that they built, while the other seven [23, 30, 46, 53, 72, 77, 84] reused those datasets. When examining the datasets, we found that the description of how these datasets were produced is insufficient and the authors may not be aware of the data leakage. Moreover, the datasets provided by previous studies are often trusted by their users. We recommend dataset providers to explain how their datasets were processed. It is also more reliable if the data processing is traceable [90]. For dataset users, it is important to understand the derivation of a dataset and how to correctly use the dataset.

5.3 Recommendations for Practitioners

As we have observed, the models based on ITD to make predictions or recommendations may perform better with higher quality issue reports. In practice, the quality of issue reports (i.e., to be accurate and complete) is often improved in the resolution process as contributors refine the reports. If practitioners want to use models to resolve issues more efficiently, they need to collect data appropriately. First, they should follow the rules summarized in Section 5.2 to obtain non-leaked data matching their application scenarios and fit the model. Second, they may select a time point when the information of an issue is more complete and accurate (or take actions to make that) to apply the model.

We also observed leakage problem resulted from leaked factors when inspecting the papers. As mentioned in Section 3.2, several models include factors (features) which can only be calculated after the intended application scenarios. Such models are impractical in predictions. When practitioners build models to conduct prediction, they should first examine whether the selected factors require information from the “future.”

Our findings related to the relevance of issue attributes may help practitioners to design better ITS. We found that leakages in the information of the issue attributes *summary*, *component*, and *assignee* have the largest impact on the model performance. This highlights the importance of designing better ITS to improve the quality of issue reports.

6 LIMITATIONS

The limitations of this study mainly exist in the operation of the proposed methods and the generalization of the results.

Operation. We recovered the values of the issue attributes at the time when the reports were submitted and considered them as matching the intended application scenarios. However, this is not

the only case where the applied data and scenarios are matched. For example, issue localization can be conducted at any time before the issue is fixed. The reason why we choose the submission time of the issues for extracting attribute values is that the data at submission time are guaranteed to exclude “future” information.

Generalization. We found that studies using leaked data may achieve biased performance for the proposed models. The ITD we used are official database dumps from Mozilla and Eclipse. We choose these two projects for two reasons: 1) the majority (42 out of 55) of the investigated papers used data from these two projects; 2) they have been using ITS for a long time. For example, the Bugzilla system was designed and first used by Mozilla. Our results may not apply to other projects (i.e., other datasets). However, the contexts of these two projects differ from each other. Mozilla serves end-users while Eclipse serves developers. The change patterns of issue attributes differ in these contexts while the results derived from these two projects are consistent, which implies that the problems caused by the changes in the studied attributes may be generalizable to other projects/datasets.

The results may not apply to the 52 other papers collected together with the three reproduced papers. However, the three papers are representative because they are chosen from different application scenarios and are referenced, reproduced and compared in subsequent papers [43, 46, 53, 63, 64, 69]. Additionally, the issue attributes used in these studies (shown in Table 3) are likely to be misused if users are not aware of the dynamic nature of these attributes, and therefore may lead to biased results as discovered in this study.

The results may not apply to other ITD-based studies (addressing other problems, except the eleven applications) that confuse the time of their application scenarios with the time of data retrieval. However, the three studies investigated in this paper represent three common and different applications (and models), yet the performance of all the proposed models decreases with leaked data. Therefore, it is reasonable to speculate that other models may demonstrate similar phenomena.

7 RELATED WORK

Although not substantial, work on risks of using software development data has been getting more and more attention. The importance of understanding the context where the software development data were produced when using them has been emphasized by Mockus [41]. The context has many aspects and can be viewed from micro and more aggregate levels. From the micro perspective, it includes when the event in the data happened, what may have been on the related developer’s conscious and subconscious mind at the time, what tools and practices were used, why the action was taken, etc. From the macro perspective, it includes ecosystems, large enterprises, or commercial and OSS hemispheres. In this paper, we focus on the time when the data were produced.

A number of studies attempted to enrich people’s understanding of context of some specific software development data and prevent possible misuses. Bird et al. analyzed the Git and revealed perils of mining Git [11], e.g., shared terms between Git and traditional version control system like SVN can have different meanings. Kalliamvakou et al. conducted similar study on GitHub and highlighted the pitfalls of mining GitHub [26], e.g., a repository is not

necessarily a project, most projects have very few commits. Negara et al. revealed the risk of studying file-based version control system (VCS) as the primary source of code evolution data [44], e.g., some code changes are not stored in VCS, thus, VCS-based code evolution research is incomplete. Lanza et al. pointed out bias in the evaluation of bug prediction techniques because the data used in evaluation may not be consistent with the actual practice [34], e.g., many bugs are not included in the bug reports.

For issue tracking data, two misuse cases have been revealed in the literature. The time when an issue’s status is labeled as RESOLVED is often used to determine the time when the issue is actually resolved [10, 17, 29]. However, Zheng et al. found that the actions of changing status are often delayed in OSS projects, and the delay could be months even years [87], e.g., developers often clean up issues that have already been fixed yet have not been closed in batch. Data users who use report status to determine issue-fix time are often unaware of such context and therefore overestimate the actual issue-fix time. Herzig et al. found that one third of issue reports in Mozilla and Apache are none of bug reports, but people may improperly assume that all the issues in ITS are bug reports, and train and test bug prediction models with the mixed reports [18]. The experiments conducted by Herzig et al. confirmed that the performance of such models is not reliable in practice.

In data mining field, the mistake of data leakage, which is caused by ignoring when the data were produced, is commonly made by data users [27]. ITD changes over time, and their users may have similar misuses. In this paper, we aim to confirm and reveal the impact of data leakage in ITD uses which has not been addressed before.

8 CONCLUSIONS

While software development data have been widely used, the problem of data misuse has not attracted enough attention it deserves. In this study, we investigated data leakage, i.e., using data from the future in ITD applications. By exploring the existing literature, we confirmed the existence of data leakage and found that it poses a significant threat to the results of the affected studies. We reproduced three studies that represent three common ITD applications with both non-leaked and leaked data and found that leaked data lead to biased results while the extent is not striking. By controlling the variables used to reproduce the studies, we found that changes in the *summary*, *component* and *assignee* attributes have the largest impact on the biased results. We made recommendations for researchers and practitioners regarding how to avoid data leakage and achieve better results in practice. We expect our work to highlight pitfalls of data leakage and lead to more accurate and realistic predictions in practice and more relevant methods in research.

We provide scripts and datasets used in this study at github.com/fayekia/FSE_2018.

ACKNOWLEDGMENTS

This work is supported by the National key R&D Program of China Grant 2018YFB10044201, the National Natural Science Foundation of China Grants 61432001 and 61690200.

REFERENCES

- [1] Karan Aggarwal, Tanner Rutgers, Finbarr Timbers, Abram Hindle, Russ Greiner, and Eleni Stroulia. 2015. Detecting duplicate bug reports with software engineering domain knowledge. In *Software Analysis, Evolution and Reengineering (SANER)*, 2015 IEEE 22nd International Conference on. IEEE, 211–220.
- [2] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Leong. 2009. Diversifying search results. In *Proceedings of the second ACM international conference on web search and data mining*. ACM, 5–14.
- [3] Rafi Almhana, Wiem Mkaouer, Marouane Kessentini, and Ali Ouni. 2016. Recommending relevant classes for bug reports using multi-objective search. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 286–295.
- [4] P. Anbalagan and M. Vouk. 2009. On predicting the time taken to correct bug reports in open source projects. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. 523–526.
- [5] Sigrún Andradóttir, Daniel P Heyman, and Teunis J Ott. 1993. Variance reduction through smoothing and control variates for Markov chain simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 3, 3 (1993), 167–189.
- [6] John Anvik. 2006. Automating bug report assignment. In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*. ACM, New York, NY, USA, 937–940.
- [7] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*. ACM, New York, NY, USA, 361–370.
- [8] John Anvik and Gail C. Murphy. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.* 20, 3, Article 10 (Aug. 2011), 35 pages.
- [9] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Duplicate bug reports considered harmful... really?. In *Software maintenance, 2008. ICSM 2008. IEEE international conference on*. IEEE, 337–345.
- [10] Pamela Bhattacharya and Iulian Neamtiu. 2011. Bug-fix Time Prediction Models: Can We Do Better?. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*. ACM, New York, NY, USA, 207–210.
- [11] Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. 2009. Fair and Balanced? Bias in Bug-fix Datasets. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09)*. ACM, New York, NY, USA, 121–130.
- [12] Andrew P Bradley. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern recognition* 30, 7 (1997), 1145–1159.
- [13] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2012. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering* 1, 1 (2012), 1–182.
- [14] Carlos Eduardo Albuquerque da Cunha, Yguaratã Cerqueira Cavalcanti, Paulo Anselmo da Mota Silveira Neto, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. 2010. A Visual Bug Report Analysis and Search Tool.. In *SEKE*. 742–747.
- [15] Shin Fujiwara, Hideaki Hata, Akito Monden, and Kenichi Matsumoto. 2015. Bug report recommendation for code inspection. In *Software Analytics (SWAN)*, 2015 IEEE 1st International Workshop on. IEEE, 9–12.
- [16] Daniel M German and Julius Davies. 2011. Apples vs. oranges?: An exploration of the challenges of comparing the source code of two software systems. In *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 246–249.
- [17] Emanuel Giger, Martin Pinzger, and Harald Gall. 2010. Predicting the Fix Time of Bugs. In *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering (RSSE '10)*. ACM, New York, NY, USA, 52–56.
- [18] K. Herzig, S. Just, and A. Zeller. 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Software Engineering (ICSE)*, 2013 35th International Conference on. 392–401.
- [19] Abram Hindle, Anahita Alipour, and Eleni Stroulia. 2016. A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering* 21, 2 (2016), 368–410.
- [20] Yukinao Hirata and Osamu Mizuno. 2011. Do comments explain codes adequately?: investigation by text filtering. In *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 242–245.
- [21] Pieter Hooimeijer and Westley Weimer. 2007. Modeling Bug Report Quality. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*. ACM, New York, NY, USA, 34–43.
- [22] Qiao Huang, David Lo, Xin Xia, Qingye Wang, and Shanping Li. 2017. Which packages would be affected by this bug report? (2017).
- [23] Da Huo, Tao Ding, Collin McMillan, and Malcom Gethers. 2014. An Empirical Study of the Effects of Expert Knowledge on Bug Reports.. In *ICSME*. 1–10.
- [24] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE '09)*. ACM, New York, NY, USA, 111–120.
- [25] Leon Wu Boyi Xie Gail Kaiser and Rebecca Passonneau. 2011. BugMiner: Software reliability analysis via data mining of bug reports. *delta* 12, 10 (2011), 09–0500.
- [26] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel German, and Daniela Damian. May 31–June 1, 2014. The Promises and Perils of Mining GitHub. In *MSR'2014*. 92–101.
- [27] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. 2012. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6, 4 (2012), 15.
- [28] Dongsun Kim, Yida Tao, Sunghun Kim, and A. Zeller. 2013. Where Should We Fix This Bug? A Two-Phase Recommendation Model. *Software Engineering, IEEE Transactions on* 39, 11 (Nov 2013), 1597–1610.
- [29] Sunghun Kim and E. James Whitehead, Jr. 2006. How long did it take to fix bugs?. In *Proceedings of the 2006 international workshop on Mining software repositories (MSR '06)*. ACM, New York, NY, USA, 173–174.
- [30] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2015. Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports (N). In *Automated Software Engineering (ASE)*, 2015 30th IEEE/ACM International Conference on. IEEE, 476–481.
- [31] Ahmed Lamkanfi and Serge Demeyer. 2012. Filtering bug reports for fix-time analysis. In *Software Maintenance and Reengineering (CSMR)*, 2012 16th European Conference on. IEEE, 379–384.
- [32] Ahmed Lamkanfi and Serge Demeyer. 2013. Predicting reassignments of bug reports—an exploratory investigation. In *Software Maintenance and Reengineering (CSMR)*, 2013 17th European Conference on. IEEE, 327–330.
- [33] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. 2010. Predicting the severity of a reported bug. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 1–10.
- [34] Michele Lanza, Andrea Mocci, and Luca Ponzanelli. 2016. The tragedy of defect prediction, prince of empirical software engineering research. *IEEE Software* 6 (2016), 102–105.
- [35] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 308–311.
- [36] Meng-Jie Lin, Cheng-Zen Yang, Chao-Yuan Lee, and Chun-Chang Chen. 2016. Enhancements for duplication detection in bug reports with manifold correlation features. *Journal of Systems and Software* 121 (2016), 223–233.
- [37] Kaiping Liu, Hee Beng Kuan Tan, and Mahinthan Chandramohan. 2012. Has this bug been reported?. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 28.
- [38] Kaiping Liu, Hee Beng Kuan Tan, and Hongyu Zhang. 2013. Has this bug been reported?. In *Reverse Engineering (WCRE)*, 2013 20th Working Conference on. IEEE, 82–91.
- [39] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. 2010. Bug localization using latent dirichlet allocation. *Information and Software Technology* 52, 9 (2010), 972–990.
- [40] Dominique Matter, Adrian Kuhn, and Oscar Nierstrasz. 2009. Assigning bug reports using a vocabulary-based expertise model of developers. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 131–140.
- [41] Audris Mockus. 2014. Engineering big data solutions. In *on Future of Software Engineering*. 85–99.
- [42] Laura Moreno, Wathsala Bandara, Sonia Haiduc, and Andrian Marcus. 2013. On the relationship between the vocabulary of bug reports and source code. In *Software Maintenance (ICSM)*, 2013 29th IEEE International Conference on. IEEE, 452–455.
- [43] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in software engineering research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 466–476.
- [44] Stas Negara, Mohsen Vakilian, Nicholas Chen, Ralph E Johnson, and Danny Dig. 2012. Is it dangerous to use version control histories to study source code evolution?. In *ECOOP*, Vol. 12. Springer, 79–103.
- [45] Anh Tuan Nguyen, Tung Thanh Nguyen, J. Al-Kofahi, Hung Viet Nguyen, and T.N. Nguyen. 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In *Automated Software Engineering (ASE)*, 2011 26th IEEE/ACM International Conference on. 263–272.
- [46] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Automated Software Engineering (ASE)*, 2012 Proceedings of the 27th IEEE/ACM International Conference on. IEEE, 70–79.
- [47] Robert Nisbet, Gary Miner, and John Elder IV. 2009. *Handbook of statistical analysis and data mining applications*. Academic Press.
- [48] Lucas D Panjer. 2007. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on mining software repositories*. IEEE Computer Society, 29.
- [49] Jinwoo Park, Muwoong Lee, Jinhan Kim, Seungwon Hwang, and Sunghun Kim. 2011. Costrage: A cost-aware triage algorithm for bug reporting systems. In

- Proceedings of the National Conference on Artificial Intelligence*. 139.
- [50] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. 2004. Simple BM25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, 42–49.
 - [51] Saharon Rosset, Claudia Perlich, Grzegorz Świrszcz, Prem Melville, and Yan Liu. 2010. Medical data mining: insights from winning two competitions. *Data Mining and Knowledge Discovery* 20, 3 (2010), 439–468.
 - [52] Reuven Y Rubinstein and Ruth Marcus. 1985. Efficiency of multivariate control variates in Monte Carlo simulation. *Operations Research* 33, 3 (1985), 661–677.
 - [53] R.K. Saha, M. Lease, S. Khurshid, and D.E. Perry. 2013. Improving bug localization using structured information retrieval. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. 345–355.
 - [54] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M Ibrahim, Masao Ohira, Bram Adams, Ahmed E Hassan, and Ken-ichi Matsumoto. 2013. Studying re-opened bugs in open source software. *Empirical Software Engineering* 18, 5 (2013), 1005–1042.
 - [55] Ramin Shokripour, John Anvik, Zarinah M Kasirun, and Sima Zamani. 2013. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2–11.
 - [56] Ramin Shokripour, John Anvik, Zarinah M Kasirun, and Sima Zamani. 2015. A time-based approach to automatic bug report assignment. *Journal of Systems and Management* 102 (2015), 109–122.
 - [57] Dag IK Sjøberg, Jo Erskine Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, N-K Liborg, and Anette C Rekdal. 2005. A survey of controlled experiments in software engineering. *IEEE transactions on software engineering* 31, 9 (2005), 733–753.
 - [58] Yoonki Song, Xiaoyin Wang, Tao Xie, Lu Zhang, and Hong Mei. 2010. JDF: detecting duplicate bug reports in Jazz. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—Volume 2*. ACM, 315–316.
 - [59] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 253–262.
 - [60] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—Volume 1*. ACM, 45–54.
 - [61] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar Al-Kofahi, and Tien N Nguyen. 2011. Fuzzy set-based automatic bug triaging: NIER track. In *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 884–887.
 - [62] Ferdian Thung, Pavneet Singh Kochhar, and David Lo. 2014. DupFinder: integrated tool support for duplicate bug report detection. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 871–874.
 - [63] Ferdian Thung, David Lo, Lingxiao Jiang, Foyzur Rahman, Premkumar T Devanbu, et al. 2012. When would this bug get reported?. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 420–429.
 - [64] Ferdian Thung, Shaowei Wang, David Lo, and Lingxiao Jiang. 2012. An empirical study of bugs in machine learning systems. In *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*. IEEE, 271–280.
 - [65] Ferdian Thung, Shaowei Wang, David Lo, and Julia Lawall. 2013. Automatic recommendation of API methods from feature requests. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 290–300.
 - [66] Yuan Tian, David Lo, and Chengnian Sun. 2012. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, 215–224.
 - [67] Y. Tian, D. Lo, and C. Sun. 2013. DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. 200–209.
 - [68] Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. 2015. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering* 20, 5 (2015), 1354–1383.
 - [69] Yuan Tian, Chengnian Sun, and David Lo. 2012. Improved duplicate bug report identification. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. IEEE, 385–390.
 - [70] Yuan Tian, Dinusha Wijedasa, David Lo, and Claire Le Goues. 2016. Learning to rank for bug report assignee recommendation. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. IEEE, 1–10.
 - [71] Harold Valdivia Garcia and Emad Shihab. 2014. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 72–81.
 - [72] Shaowei Wang and David Lo. 2014. Version history, similar report, and structure: Putting them together for improved bug localization. In *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 53–63.
 - [73] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering (ICSE '08)*. ACM, New York, NY, USA, 461–470.
 - [74] Xinlei Oscar Wang, Eilwoo Baik, and Premkumar T Devanbu. 2011. Operating system compatibility analysis of Eclipse and Netbeans based on bug data. In *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 230–233.
 - [75] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. May 2007. How Long Will It Take to Fix This Bug? 20–26.
 - [76] Wei Wen, Tingting Yu, and Jane Huffman Hayes. 2016. Colua: Automatically predicting configuration bug reports and extracting configuration options. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE, 150–161.
 - [77] Chu-Pan Wong, Yingfei Xiong, Hongyu Zhang, Dan Hao, Lu Zhang, and Hong Mei. 2014. Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis. In *ICSME*. Citeseer, 181–190.
 - [78] Jialiang Xie, Qimu Zheng, Minghui Zhou, and Audris Mockus. 2014. Product Assignment Recommender. In *ICSE'2014 Research Demonstration*. Hyderabad, India, 556–559.
 - [79] Jialiang Xie, Minghui Zhou, and Audris Mockus. 2013. Impact of triage: a study of mozilla and gnome. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*. IEEE, 247–250.
 - [80] Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. 2012. Developer prioritization in bug repositories. In *Software Engineering (ICSE), 2012 34th International Conference on*. 25–35.
 - [81] Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. 2016. Combining word embedding with information retrieval to recommend similar bug reports. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE, 127–137.
 - [82] Xin Ye, Razvan Bunescu, and Chang Liu. 2014. Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 689–699.
 - [83] Xin Ye, Razvan Bunescu, and Chang Liu. 2016. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. *IEEE Transactions on Software Engineering* 42, 4 (2016), 379–402.
 - [84] Klaus Changsun Youm, June Ahn, and Eunseok Lee. 2017. Improved bug localization based on code change histories and bug reports. *Information and Software Technology* 82 (2017), 177–192.
 - [85] Hugo Zaragoza, Nick Craswell, Michael J Taylor, Suchi Saria, and Stephen E Robertson. 2004. Microsoft Cambridge at TREC 13: Web and Hard Tracks. In *TREC*, Vol. 4. 1–1.
 - [86] Wen Zhang, Song Wang, and Qing Wang. 2016. KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. *Information and Software Technology* 70 (2016), 68–84.
 - [87] Qimu Zheng, Audris Mockus, and Minghui Zhou. 2015. A method to identify and correct problematic software activity data: exploiting capacity constraints and data redundancies. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 637–648.
 - [88] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 14–24.
 - [89] Minghui Zhou and Audris Mockus. 2015. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering* 41, 1 (2015), 82–99.
 - [90] Jiaxin Zhu, Minghui Zhou, and Hong Mei. 2016. Multi-extract and multi-level dataset of mozilla issue tracking history. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 472–475.