A Case Study of Bias in Bug-Fix Datasets

Thanh H. D. Nguyen, Bram Adams, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University
Kingston, Ontario, Canada
Email: {thanhnguyen,bram,ahmed}@cs.queensu.ca

***Abstract*—Software quality researchers build software quality models by recovering traceability links between bug reports in issue tracking repositories and source code files. However, all too often the data stored in issue tracking repositories is not explicitly tagged or linked to source code. Researchers have to resort to heuristics to tag the data (e.g., to determine if an issue is a bug report or a work item), or to link a piece of code to a particular issue or bug.**

Recent studies by Bird et al. and by Antoniol et al. suggest that software models based on imperfect datasets with missing links to the code and incorrect tagging of issues, exhibit biases that compromise the validity and generality of the quality models built on top of the datasets. In this study, we verify the effects of such biases for a commercial project that enforces strict development guidelines and rules on the quality of the data in its issue tracking repository. Our results show that even in such a perfect setting, with a near-ideal dataset, biases do exist – leading us to conjecture that biases are more likely a symptom of the underlying software development process instead of being due to the used heuristics.

***Keywords*-sample; bias; bug-fix; prediction; data quality;**

I. Introduction

Software repositories allow researchers to closely study and examine the different factors that influence and impact the quality of a software system [1]–[3]. A good understanding of these factors helps improve the quality of future releases of a software system. Using these factors, practitioners can allocate testing and code-review efforts in large projects, enabling cost-effective and timely quality control processes.

Much of the information stored in software repositories must be processed and cleaned for further analysis. Such processing and cleaning is often done using assumptions and heuristics. One popular assumption is that most issue tracking systems (e.g., Bugzilla[1]) contain just bug reports while in reality they contain other items such as tasks or enhancements. On the other hand, heuristics, such as those described in [4] and [5], link bug reports to the location of the bug in the code by locating bug IDs in change messages in a source control repository.

The quality of these assumptions and heuristics represents a threat to the validity of results derived from software repositories. Recent studies by Antoniol et al. [6] and Bird et al. [7] have casted doubts on the quality of data produced using such heuristics. These prior studies have identified two types of biases:

- **Linkage bias:** Bird et al. [7] note that when considering links between bug reports and actual code, there exists bias in the severity level of bugs and the experience of developers fixing bugs. The higher the severity of a bug, the less chance it will be linked to source code. This implies that quality models will be more biased towards the lower severity bugs. Bird et al. also found that there are more bugs linked by experienced developers than by less experienced developers. A quality model built using only the linked bugs, will be biased towards the behaviour of more experienced developers.
- **Tagging bias:** Antoniol et al. [6] note that many of the issues in an issue tracking system do not actually represent bug reports. Instead developers often use issue tracking systems to track other issues such as tasks, decisions, and enhancements. Therefore, using such data might lead to incorrect bug counts for the different parts of a software system.

While prior studies have shown that the above biases exist, they never explored if such biases are due to the software development process in general (e.g., it might be the case that experienced developers are by definition more likely to fix bugs over junior developers), or if they are due to the used heuristics. To answer such a question, we would require a perfect dataset where such linkages and tagging of issues was done by professional developers and with great attention to detail. While such a perfect dataset might not exist, we believe that a near-ideal dataset does exist. This dataset is derived from the IBM Jazz™[2] project. The IBM Jazz project follows a disciplined software development process with careful attention to continuously maintaining the linkages between issues and code changes through automated tool support. The strong discipline and the automated tool support gives us strong confidence that this project represents a near-ideal real-life case of high quality linkage dataset that is correctly tagged.

Using this near-ideal dataset we re-examine the aforemen-

[1]http://www.bugzilla.org

[2]http://www.jazz.net. IBM and Jazz are trademarks of IBM Corporation in the US, other countries, or both.

tioned biases. We find that even in such an ideal setting, biases do exist in the linked dataset, indicating that such biases are more likely due to the software development process rather than being a side-effect of the linking heuristics. We also find that, even under tagging bias, bug prediction models, such as those in [5], [8], [9], will still perform almost as if there is no bias. Our results suggest that these biases may exist in software data as properties of the software process itself and that these biases should not stop researchers from using the dataset. However, as our results are derived from a single dataset, we encourage other researchers to replicate our study to verify the generality of the findings.

This paper provides the following two contributions:

- We argue that linkage bias is not a symptom of imperfect linkage but more likely a property of the software process.
- We suggest that tagging bias does not significantly affect bug prediction models.

Paper Organization. The next section describes the related work and the background on bias in software research. Section III introduces our data collection and analysis techniques. Section IV presents our results. Section V discusses the implications of our results on software quality models. Section VI discusses the threats to the validity. Section VII concludes the paper.

II. Background and Related Work

Scientific studies build models of and theories about the nature of the subject under study. Such scientific models and theories have to be validated by evidence. Evidence consists of facts that are collected by the researchers from the subject under study. For example, if a biologist wants to study the behaviour of a certain kind of birds, he or she will have to find and observe the behaviour of some individual birds. The problem is that the biologist cannot observe all the birds of a specific kind, but hopes that the studied birds are representative of all the birds of that kind.

What if the birds that the researcher was able to find behave differently from their siblings because of local adaptation? In that case, the behavioural model built by the researcher will not be representative for all the birds of that kind. This problem is called sampling bias. A famous example of sampling bias in social science is the 1936 U.S. presidential election poll [10]. Prior to the election date, the Literary Digest magazine conducted a survey via postage mail and concluded that the Republican party's candidate was going to win. However, the magazine collected its mailing list through the telephone book and the automobile register list. In the 1930s, telephones and automobiles were only affordable by richer Americans, who favoured the Republican party at the time. The Democrat party's candidate won the election by the widest margin in history.

Software quality studies will probably not be as inaccurate as the Literary Digest. However, the question of sample bias is a valid concern that should be addressed. There are not many studies about data quality in software engineering literature. The need for a systematic review of data quality in software engineering is outlined by Liebchen and Shepperd [11], [12]. They surveyed 552 research papers in major software engineering conferences and found that only 23 actually reported the quality of the data. Their report cast doubt about the quality of data in software engineering research. Mockus [13] stated that the quality of the data is more important than the choice of analysis method. He emphasized missing or invalid data as a common source of sampling bias in software engineering. He outlined techniques to deal with missing data.

As mentioned in the introduction, we concentrate on two sources of sampling bias in this study, as reported by Bird et al. [7] and Antoniol et al. [6].

A. Linkage bias

We call the population of all bug fixes B_f. Software quality researchers intend to build their models based on this dataset, but, in the majority of the cases, researchers can only recover B_{fl}. B_{fl} are the bug fixes that would be linked to an actual change set in the source control system. Since the bug tracking system, such as BugZilla, and the source control system, such as CVS or SVN, are standalone systems that are usually not linked together, researchers have to rely on heuristics to detect linkages between a change set and a bug. For example, in some projects such as Eclipse, developers put the bug report number in the comment field of the change set that fixes the bug. Hence, a popular heuristic is to scan the comments of the change sets and detect the bug report numbers [4], [5]. Unfortunately, such heuristics are imperfect. Developers might have forgotten to enter the bug report number.

Because of the imperfection of the traceability heuristic, the linked bugs, B_{fl}, are normally only a fraction of all fixed bugs, B_f. Thus, models built on B_{fl} can be biased. This is the main thesis of Bird et al. [7]. They examined if there is bias between B_{fl} and B_f along the following features of bugs:

- **F1 Severity:** There is a difference in the distribution of severity levels between B_{fl} and B_f.
- **F2 Experience:** Bugs in B_{fl} are fixed by more experienced people than those who fix bugs in B_f.
- **F3 Maturity:** Bugs that are linked are more likely to have been closed later in the project than unlinked bugs.
- **F4 Release pressure:** Bugs that are closed near a project release date are less likely to be linked.
- **F5 Collaboration:** Bugs that are linked will have more people or events associated with them than bugs that are not.

Table I
THE IBM JAZZ DATASET.

Attribute	Value
Time	14 months, 5 iterations
# Issues	23,025
# Change sets	5,780
# Issue reporters	288
# Developers	141
# Location	22

Bird et al. [7] found that there is bias in F1 and F2. In this paper, we will replicate their study to confirm if the biases caused by missing linkages also exist in a project that strictly enforces linkages. We call this type of bias ***linkage bias***. We note that Bird et al. also identify verifiability as another potential bias feature. However, all resolved bug reports in the system we examine must be either approved, verified, or peer reviewed. Hence, we cannot test for bias in this feature, because most bug reports are verified one way or the other.

B. Tagging bias

The second potential threat of sampling bias is the type of bug report. The purpose of bug tracking systems such as Bugzilla is to report unexpected behaviour of a software system. However, as bug tracking systems become more popular, especially in open-source software (OSS) projects, developers start to use Bugzilla not only to track bugs, but also to specify tasks, enhancements, or even requirements. Antoniol et al. [6] reported that most bug reports are not really bugs. Through manual inspection, they found that, in a large project such as Eclipse, the number of actual bugs is only about a third of all bug reports. This poses a challenge, because bug prediction studies [1], [5], [14] assume that all bug reports are actually defects. Could this be the reason why generalizing bug prediction models from one project to other projects does not work [3]? In this paper, we examine the existence of bias on the features of bugs, mentioned above, between all bug reports and reports that are actual defects. We call this type of bias ***tagging bias***.

III. CASE STUDY SETUP

A. Data collection

Our study uses software quality data from a commercial software project, i.e., the IBM Jazz software project, that develops integrated development environments (IDE). We choose to study this particular team because, as explained before, the team strictly enforces the linkages between the bugs and the change sets. When developers check in a change set, they have to explicitly specify the bug/issue that the change set is supposed to fix. This practice (theoretically) eliminates linkage bias on this dataset. Secondly, their bug reporting system tracks the different types of reports, such as tasks, enhancements, or actual bugs. This capability of the bug reporting system (theoretically) eliminates tagging bias in this dataset.

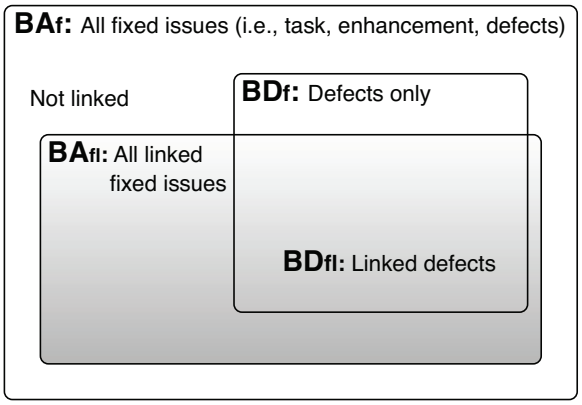

Figure 1. An overview of the IBM Jazz dataset that we use to examine the existence of biases. To check for linkage bias, we compare the distribution of each feature between BA_f vs. BA_{fl} and BD_f vs. BD_{fl}. To check for tagging bias, we compare the distribution of each feature between BA_f vs. BD_f.

We are not claiming that this dataset is perfect, because a developer can still misclassify a task as a bug. He or she can also link the wrong bug to the committing change set. However, we claim that the quality of this dataset is as ideal as one can get from a software repository, because of two reasons. The first reason is that Jazz is a commercial project with a stable base of developers. Each developer has responsibility to his or her team and reports to his or her team lead. Hence, we believe that mistakes in linking and classifying bug reports should be minimal. The second, and more important reason, is that the bug report system and the source control system is part of the software product they are building. The developers have a motivation to customize and utilize their tools. Because of these two reasons, we believe that this dataset does not suffer from linkage and tagging bias. Therefore, if we still determine that a feature is biased in this dataset, the bias is most likely a property of the software development process itself instead of being caused by inexact heuristics.

The IBM Jazz team has about 200 developers located in Canada, the United States, Europe and Asia. Table I shows basic information about the team. The team follows an agile development methodology. Each iteration takes about 12 weeks. Each iteration may result in one or two deliveries, either in the middle and/or at the end of the iteration. We are able to collect data of five iterations, which span about 14 months. All five iterations belong to the same product version.

In total, there are 23,025 bug reports over the five iterations, of which, 13,367 are fixed. This is the dataset that we will use in our study. Figure 1 shows how we divided the data such that we can test the effect of the two biases.

Table II
NUMBER OF FIXED ISSUES IN EACH DATASET. THE RELATIONSHIP AMONG THE DATASETS IS PRESENTED IN FIGURE 1.

Dataset	Linkage	Notation	# Issues
All fixed issues (defect, task, enhanc.)	All	BA_f	13,367
	Linked	BA_{fl}	10,960
Defects only	All	BD_f	9,462
	Linked	BD_{fl}	8,038

Table II shows the amount of data in each set. To avoid confusion, we call the set of all fixed bug reports "all issues" or BA_f, to distinguish it from the set of actual bugs that we will call "defects only" or BD_f. We will call the set of issues that are linked as BA_{fl} and the set of linked defects as BD_{fl}. In total, there are 13,367 bug reports (BA_f). About 82% of which, i.e., 10,960 issues, can be linked to source code (BA_{fl}). Among all bug reports are 9,462 real defects (BD_f), out of which, 8,038 (85%) are linked to source code ($BD_f l$).

B. Analysis Techniques

To determine the effects of linkage bias, we will compare the distribution of the five features of Bird et al. (Section II) between the linked issues/defects and the entire population of all issues/defects. In other words, we will compare between BA_f and BA_{fl}, and between BD_f and BD_{fl}. To determine the effects of tagging bias, we will compare the distribution of each feature between all issues, i.e., BA_f and defects only, i.e., BD_f.

We will use two statistical tests in our analysis: Fisher's exact test [15] and a two-sample Kolmogorov-Smirnov (K-S) test. Both tests have the same purpose: to compare the distribution of each feature between two datasets. The tests will tell us if two datasets come from the same population, i.e., there is no statistically significant difference between their distributions. The null hypothesis in both tests is that the two datasets have the same distribution. The tests will give us a statistic and a p-value. If the p-value is smaller than a certain threshold, we can reject the null hypothesis, which means that the datasets do not have the same distribution. In both tests, we will use a p-value threshold of 0.01.

The difference between Fisher and K-S is that Fisher is used for nominal data, such as the severity level, while K-S is used for ordinal and continuous data, such as developer experience. An alternative for Fisher's exact test is Pearson's χ^2 test, which is much faster but computes an approximation for Fisher's. We were able to run Fisher's test on the largest dataset, BA_f, in under 15 minutes, so we opt to use Fisher's test for accuracy. An alternative for the K-S test is the two-sample t-test. This t-test requires the data distribution to be normal. As we will show through the box plots in the next section, our data is not normally distributed. Hence, we decided to use the K-S test.

Table III
SEVERITY (FEATURE 1): DISTRIBUTION OF SEVERITY IN BA_f AND BD_f MEASURED AS THE PERCENTAGE OF ISSUES/DEFECTS IN EACH SEVERITY LEVEL OVER THE TOTAL POPULATION. THE LAST TWO COLUMNS SHOW THE P-VALUE OF THE FISHER TEST BETWEEN THE DATASETS.

Dataset	Blocker	Critical	Major	Normal	Minor	Fisher's p	
BA_f	1.2	3.2	9.2	83.9	2.6	0.72	<0.001
BA_{fl}	1.2	3.2	9.6	83.3	2.8		
BD_f	1.5	4.0	11.4	80.1	3.0	0.94	
BD_{fl}	1.4	4.0	11.6	79.9	3.2		

IV. ANALYSIS AND RESULTS

In this section, we analyze the IBM Jazz dataset for each bug feature, as introduced in Section II.

A. Feature 1: Severity

Motivation. In the original study, Bird et al. [7] reported that the severity of linked bugs B_{fl} is biased toward less critical bugs. Software quality models that are built on B_{fl} will not be representative for all bugs, but rather will be focused on less severe bugs.

Approach and Finding. Table III shows the distribution of severity in the Jazz data. The table shows the percentage of issues in each severity level over the overall population. For the effect of **linkage bias**, we can see that when we use all bug reports, there are small differences between the distribution of the Major, Normal and Minor severity levels for the linked BA_{fl} and all issues BA_f. When we only use the real defects, BD_f vs. BD_{fl}, the differences are smaller. In order to confirm the difference, we run Fisher's exact test to determine if the two populations should come from the same population. The test shows that in either case the difference is not large enough to be statistically significant (second to last column on Table III). Thus we cannot reject the null hypothesis that there is no difference. For the effect of **tagging bias**, we can see that there is definitely a difference between BA_f and BD_f. Fisher's exact test confirms the statistical significance of the difference (last column on Table III).

> *We cannot find evidence of linkage bias for the severity feature. However, there is tagging bias for the severity feature.*

B. Feature 2: Experience

Motivation. Experience is another feature of bugs that was reported to be biased. Experience is defined as the number of bugs a developer has fixed before fixing that particular bug. Bird et al. [7] found that linked bugs are normally fixed by more experienced developers compared to all fixed bugs. This implies that if we build quality models

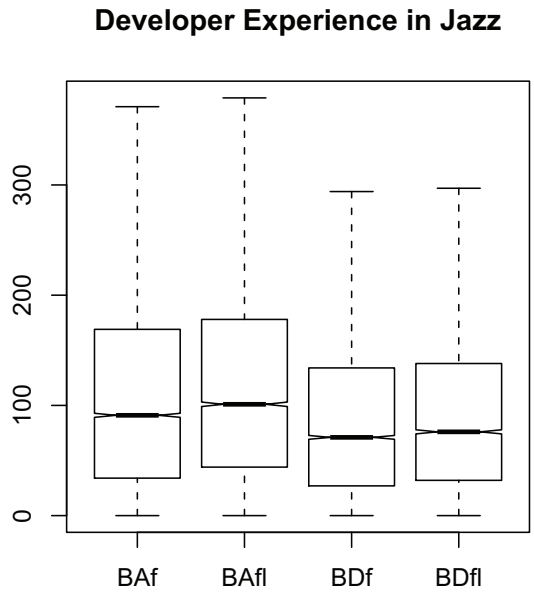

Figure 2. Developer experience in Jazz (feature 2). Experience is defined as the number of previously fixed bugs.

using the linked bugs, the model will be biased toward the behaviour of experienced developers.

Approach and Finding. In order to test the theory of Bird et al., we calculate the experience of developers in Jazz using the same method in Bird et al. [7]. For each bug, we count the number of bugs that the developer who fixed the bug has fixed before. Figure 2 shows the experience of bug fixers. As we can see in both datasets, linked issues BA_{fl} and defects BD_{fl} are fixed by more experienced developers compared to BA_f and BD_f. To make sure that the differences are statistically significant, we run K-S tests between BA_f and BA_{fl}, and between BD_f and BD_{fl}. Both tests indicate the existence of bias at $p<0.001$. Hence, there is a clear effect of **linkage bias** on experience. This is similar to what Bird et al. [7] found.

We can also see that there is a difference between BA_f and BD_f. It appears that developers who fix defects only are less experienced than developers who resolve all issues. A K-S test confirms this difference. This means that **tagging bias** exists for the experience feature.

We observe both linkage and tagging biases for the experience feature.

Table IV
MATURITY (FEATURE 3): PERCENTAGE OF LINKED BUGS OVER ALL BUGS OVER FIVE MILESTONES.

Data	M1	M2	M3	M4	M5
BA_f	78.83	81.88	80.02	83.67	83.75
BD_f	84.05	84.70	83.67	85.09	86.26

C. Feature 3: Maturity

Motivation. Providing linkages along with the committed change sets requires discipline from the developers. There is a possibility that the percentage of linked bugs relative to all bugs increases as the project matures. Bird et al. [7] tested this theory, but found no evidence to support it. In the Jazz project, the linkages are automatically enforced by tools. Regardless of maturity, all change sets have to be linked.

Approach and Finding. We look for evidences of bias by calculating the portion of linked over all bugs along the five milestones. Table IV shows the results. We can observe the increase in the percentage of linked bugs as the project matures. For all issues, BA_f, the increase from M1 to M5 is about 5%. For defect only, BD_f, the increase is about 3%. This is evidence of the **linkage bias** for the maturity feature. We can also observe that the percentage of linked bug is higher when we only consider the defects, BD_f, compare to when we consider all issues, BA_f. This shows the existence of **tagging bias**.

In contrast to Bird et al. [7], our data shows linkage and tagging biases for the maturity feature.

D. Feature 4: Release pressure

Motivation. As the release date approaches, team leads will be more vigilant about newly committed changes because changes increase the chance the build will break. They might even freeze the code near the release date. Thus, developers who commit code near the end of a release may be more careful about linking their changes to a bug. Thus the closer to the release date, the more fixed bugs should be linked compared to the beginning of the iteration. However, Bird et al. [7] found no evidence of this in the projects they studied.

Approach and Finding. To test the existence of **linkage bias**, we calculate the time from the resolution date of each bug to the nearest release. Then, we compare the distribution of these times for linked bugs and all bugs.

Figure 3 shows the box plot of days before the release date of each bug. We can observe that there is a small difference between the linked issues/defects and all issues/defects, i.e., BA_f vs. BA_{fl} and BD_f vs. BD_{fl}. A K-S test shows that these differences are statistically insignificant ($p>0.01$).

As for **tagging bias**, we can observe that there is also not much difference between BA_f and BD_f. A K-S test confirms that the difference is statistically insignificant.

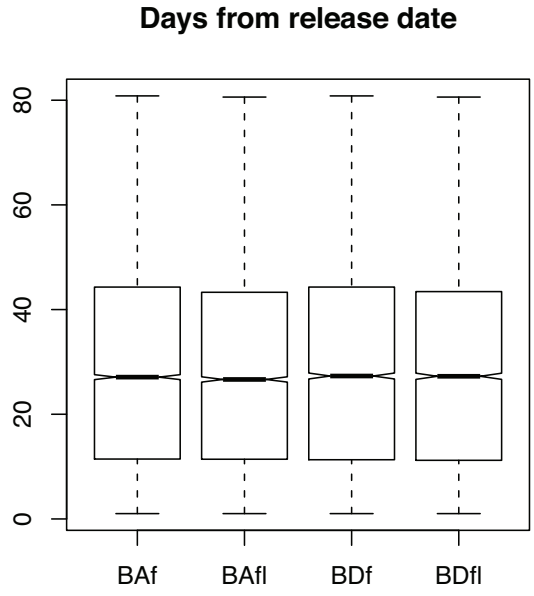

Figure 3. Release pressure (feature 4): Days before release date in Jazz.

Similar to Bird et al. [7], we find no evidence in the Jazz project of linkage and tagging biases for the release pressure feature.

E. Feature 5: Collaboration

Motivation. Some bugs require collaboration from many developers. There is a possibility that if a bug requires collaboration, the owner of the bug will be more inclined to add references to the commit log when he/she checks in the code. However, Bird et al. [7] did not find any evidence of this.

Approach. To test this potential bias, we count the number of people that were involved during the lifetime of each bug. Involvement in a bug is defined as creating, owning (fixing), commenting on or subscribing to the bug report. While there is only one creator for each bug, there might be many owners for a bug, because the ownership of a bug can be changed during the bug's lifetime.

Finding. Figure 4 shows box plots of the all and linked issues/defects in both datasets. Regarding the effect of **linkage bias**, we observe that in both datasets, i.e., all bugs BA_f and defect only BD_f, there is no difference in the number of people involved. Thus there is no evidence of linkage bias. K-S tests confirm that the difference is statistically insignificant. As for **tagging bias**, we observe no difference between BA_f and BD_f. The K-S test confirms that there is no statistically significant difference.

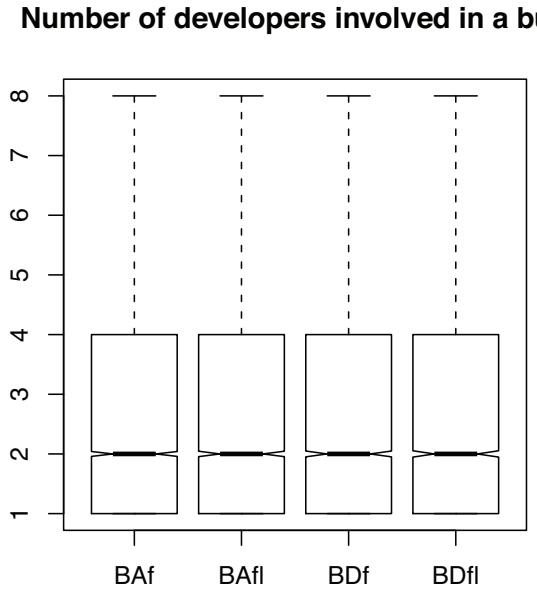

Figure 4. Collaboration (feature 5): The number of developers involved in a bug. Involvement is defined as creating, fixing, commenting on or subscribing to a bug report.

Table V
SUMMARY OF LINKAGE BIAS FOR THE FIVE STUDIED FEATURES. THE POSSIBLE IMPLICATIONS OF THE RESULTS ARE PRESENTED IN TABLE VI.

Feature	**Imperfect linkage [7]**	**Ideal linkage**	
		All (BA_f)	**Defect only (BD_f)**
Severity	Yes	No	No
Experience	Yes	Yes	Yes
Maturity	No	Yes	Yes
Release pressure	No	No	No
Collaboration	No	No	No

Similar to Bird et al. [7], we cannot find evidence of linkage or tagging biases for the collaboration feature.

V. DISCUSSION

A. Summary of results

Table V summarizes the effect of linkage bias for each studied feature. The imperfect linkage biases are the findings of Bird et al. [7] based on imperfect OSS data. The ideal linkage data are based on our study on the Jazz project. Table VII summarizes the effect of tagging bias on the five studied features.

Table VI
POSSIBLE IMPLICATIONS OF THE DIFFERENCES IN BIASES.

Imperfect linkage	Ideal linkage	Conclusion
Bias	Not bias	We confirm that when all bugs are linked properly, the bias will not exist.
Bias	Bias	We suggest that the bias is a feature of the software quality process itself; not a feature of the imperfect linked data.
Not bias	Bias	
Not bias	Not bias	We confirm that it is unlikely that the feature contains bias.

Table VII
EFFECT OF TAGGING BIAS ON AN ISSUE'S FEATURE.

Feature	Bias
Severity	Yes
Experience	Yes
Maturity	Yes
Release pressure	No
Collaboration	No

B. Does linkage bias matter?

As mentioned in Section II, research in software quality often has to rely on data from OSS projects such as Eclipse, Mozilla, or Apache. The effect of linkage bias on OSS data, as reported by Bird et al. [7], causes serious concerns over the validity of software quality studies, because heuristics to recover the linkage in OSS projects are not perfect. In their study, Bird et al. [7] state that they do not have truly unbiased data. We argue in Section III-A that the Jazz data, conversely, is considerably unbiased. This enables us to determine if linkage bias also exists in an unbiased dataset.

As we can see from Table V, of the two features that Bird et al. [7] identify as affected by linkage bias, we find that severity is not affected in our dataset. This implies that limitation of linkage heuristics is a plausible explanation for the bias in severity. However, developer experience still exhibits bias behaviour. So even in a near-ideal dataset that does not suffer from the linkage problems, developer experience is still biased. This suggests that developer experience bias is not caused by limitation of linkage heuristics, but is likely a property of the software development process followed by the Jazz team itself.

Of the three features that were found not to be affected in Bird et al.'s case study [7] of seven projects, maturity exhibits bias in the Jazz project. This shows that bias might exist in a near-ideal dataset even though it does not in imperfect datasets. The observed maturity bias is not caused by limitation of linkage heuristics but is likely a property of the Jazz team's software process.

We believe that the inconsistencies between our findings and Bird et al.'s findings provide evidences that linkage bias is not a property of missing traceability, but a fundamental property of the software process itself. For instance, the experience feature still exhibits bias behaviour even on an unbiased dataset. Maturity, on the other hand, exhibits bias behaviour in an unbiased dataset, but did not on biased datasets.

We must note that our findings are based on a single case study. So we cannot claim conclusively that linkage bias does not matter to software quality data. We believe future studies are required to verify our claims. However, determine whether bias is harmful or not is, in our opinion, a difficult problem. The first reason is that near-ideal dataset such as Jazz are very rare at the moment. The second reason is that determining the effect of linkage bias is inherently difficult. To detect the effect of bias, we have to compare the distribution of the features between all bugs and the linked bugs, which is a subset of all bugs. When the data is imperfect, there will be less linked bug. So we might be able to see the difference in distribution. However, when the data is ideal, the number of linked bugs is almost the same as the population of all bugs. So comparing the distribution of a feature in this case would be hard to reveal bias. For example, the percentages of linked bugs in the seven case studies by Bird et al. [7] range from 8.12% to 54.90%. In our near-ideal case study, this percentage is 81.99% for all issues ($|BA_{fl}|/|BA_f|$) and 84.95% for defects only ($|BD_{fl}|/|BD_f|$). In game theory terminology, this is a no-win situation, colloquially known as the Catch-22 problem. When there is a linkage problem, we can see bias. To determine the effects of linkage bias, we have to use a dataset that does not have linkage problems. However, when there is no linkage problem, the linked bugs correspond to almost all of the bugs. Hence analyzing bias in a dataset is hard. This dilemma implies that it is difficult to argue if linkage biases affect software quality data.

C. Why are there fixed bugs that are not linked?

Our results show that there are biases in developer experience and maturity. To understand the existence of these biases, we want to understand why there are fixed bugs that resulted in no code change in the first place.

To find the answer we manually inspect a random subset of fixed but not-linked issues following the principles of the grounded theory methodology [16]. Grounded theory is a common technique to inductively extract quantitative data and develop theory from unstructured data. We first randomly pick 50 fixed but not-linked bugs, which is about 2% of the population. We investigate the first 20 bugs' description and comments to determine how each bug was fixed. We code the reasons. Then, we inductively group the related reasons into bigger categories. This results in five categories as shown in Table VIII. Then we classify the rest of the 50 bugs using the five categories.

Table VIII
REASONS WHY FIXED BUT NOT-LINKED BUGS RESOLVED

Reason	Percentage
Fixed somewhere else	50%
Fixed by changes outside of the repository	18%
Cannot be determined	14%
General communication	10%
Invalid bug	4%

Table VIII shows the five categories and the percentage of not-linked bugs in each category. In about 50% of the cases, the bug seems to be fixed as a side effect of other related fixes. For example, one bug reported that there were some uninitialized objects. The bug owner closed the bug with a message indicating that the issue should have been fixed when they refactored the code in the latest development stream. In some cases, the owner also recorded the related bug that is linked to the change. The Jazz system actually has a resolution state called Fixed Upstream. We do not include fixed upstream bugs in our analysis because they are most likely not linked. However, the bugs in this category (fixed somewhere else) are only marked as fixed with no linking to a particular code change. Perhaps, it is because the fix is a side effect or the owner cannot identify the upstream bug.

18% of the not-linked bugs appear to produce changes. However, the changes are outside of the repository. For example, the team maintains a wiki that contains documentation such as tutorials or instructions. In some instances, the fix to a bug is just updating the work-around instruction on the wiki.

About 10% of the bugs represent general communication such as questions, announcements, or reminders. For example, one bug simply describes performance test results on a new database system. In another example, the reporter just asks for opinions about naming a certain user interface element.

The remaining 4% are invalid bugs which means that they should not be filed in the first place. The existence of invalid bugs is interesting because the Jazz system actually has a resolution state called Invalid. So bugs in this category should not have been marked as fixed, and hence should not be in our data. This demonstrates that even in a near-ideal dataset, tagging mistakes still exist.

Unfortunately, we cannot determine the reason for 14% of the not-linked bugs because there is no comment indicating the final status of the bug.

Interestingly, the high percentage of fixed somewhere else bugs among the bugs that are not-linked may help explain the observed maturity bias. Those bugs occur when there are other changes that have already fixed the bug. The larger a change is, the higher the chance that it fixes other bugs as well. At the beginning of the project, there are larger changes to the system compared to later on. Thus, the number of fixed somewhere else bugs is higher at the beginning of the project compared to at the end. This however requires further investigation.

D. Does tagging bias matter?

As for tagging bias, we show in Table VII that it exists for the severity, experience, and maturity features. This is an indicator that tagging bias does affect software models. If we build a software quality model using this data, then the model will likely be biased. However, does such bias make bug prediction studies inaccurate?

With only a single case study, we cannot provide a definitive answer to the question. However, we can examine the possible effects of tagging bias on a class of recent bug prediction studies [5], [8], [9]. In these studies, researchers use the number of defects associated with, for example, a class or a package to build a prediction model, such that they can predict high-risk classes or packages. These models explore correlation between some factors of the class or package and the number of defects. For example, Nagappan et al. [9] used in-process testing metrics to predict defects. However, as Antoniol et al. [6] pointed out, not all bug reports are defects. So if the data is from an OSS project and it is using, as in most cases, Bugzilla, the model will suffer from tagging bias because Bugzilla cannot distinguish between defects and tasks or enhancements. But to what extent will the model suffer?

Assume that there is no linkage bias, i.e., we can link bugs to the source code with maximum accuracy, and that we are lucky enough to be able to separate defects from other kinds of issues, as if we are using the Jazz data. Table IX shows the number of classes and packages, in Jazz, that are associated with defects (the **Defect** column) and those that are associated with all other issues (the **Other** column). We can observe that the number of unique classes and packages that are not defect-related are only 31% (3,940/12,711) and 12% (116/958) respectively because the number of classes and packages that were changed due to both defects and other issues is very high (the **Shared** column). We can measure the extent to which tagging bias affects these models by correlating the defect count when considering all linked issues, i.e., BA_{fl} as defect, and the defect count when we only consider the actual linked defects BD_{fl}. In this case, BA_{fl} is the biased data because it is equivalent to just using every bug report in Bugzilla. On the other hand, BD_{fl} is the non-biased data containing only real bugs. If the correlation is high, it means that if the risk prediction use the biased BA_{fl}, the prediction should be similar to if they use the non-biased BD_{fl}. On the other hand, if the correlation is low, it means that the prediction would have been very inaccurate because of tagging bias.

As typically done by bug prediction studies [5], [8], [9], we run both the Pearson's product-moment correlation test (Pearson) and the Spearman rank correlation test (Spearman)

Table IX
THE NUMBER OF CLASSES AND PACKAGES THAT ARE ASSOCIATED WITH DEFECTS VS. ALL OTHER ISSUES.

	Defect	**Shared**	**Other**	**Total**
Classes	3,726	5045	3,940	**12,711**
Packages	454	388	116	**958**

to compare the predicted number of defects to the actual number of defects. Pearson is a parametric correlation test while Spearman is non-parametric. Both Pearson and Spearman return a coefficient between -1 and 1. Values of 1 or -1 mean that there is a perfect positive or negative correlation. 0 means that there is no correlation.

On our data, for the packages, Spearman correlation returns a coefficient of .94 and Pearson yields .97. For the classes, the correlations are .85 and .92 respectively. The high correlations mean that the bias defect counts are almost the same as the non-bias counts. This provides the answer to our question: there are almost no differences in defect counts, even when we suffer from tagging bias. Since defect count is the only metric used by many bug prediction studies [5], [8], [9], we conjecture that tagging bias does not significantly affect the result of such studies. However, this paper is based only a single case study. We believe that further investigations, once more near-ideal datasets are available, are required to confirm our claims.

VI. THREATS TO VALIDITY

As in any empirical software engineering study, our findings are subject to certain threats to validity. We believe that the largest threat is the external validity of the study, because of the sample size. We only conducted a case study on one software project. Also, the IBM Jazz data we have access to only comprises five milestones within one version of the software. This is comparatively smaller than the data that was studied by Bird et al. [7] and Antoniol et al. [6]. Unfortunately, near-ideal datasets such as Jazz are very rare at this moment. We hope that in the future, when the adoption of integrated issue and source control tools such as Jazz become widespread, we can collect more near-ideal data to strengthen the external validity of our findings.

A possible threat to the internal validity of our study is the use of grounded theory in Section V-C. Because grounded theory method requires the authors to judge and classify the bug, it can be subjective. To counter this threat, researchers can use independent investigators to perform part of the classification and report the inter-agreement between the authors and the independent investigators. However, the non-disclosure agreement prevents us from showing the data to a third party. Thus we cannot perform this step in this study.

VII. CONCLUSION

In this paper, we examine the linkage and tagging biases on bug features. Our results indicate that biases exist even in a near-ideal dataset. This suggests that biases are most likely properties of the software process itself, and are not caused by imperfect linkage as previously thought. This is both good news and bad news. The good news is that we should be able to continue studying OSS projects because even though biases exist in the datasets, they are more likely properties of the project itself. The bad news is that researchers should be careful about checking for biases and verifying the reasons for such biases as they may vary from project to project. As for tagging bias, our study shows that it does exist. However, we show that even under tagging bias, software quality models can produce very similar results. We note, however, that this study is only a single data point. We encourage future studies to replicate and verify our claims.

We believe that more research is needed to examine the effect of bias in software quality data. For example, although we show that bug prediction models may not be affected by tagging bias, it is not clear what is the impact of bias on models that predict the resolution time or analyze the flow of bugs? The effect of tagging bias in those cases is still unknown.

VIII. ACKNOWLEDGEMENTS

We thank all members of the IBM's Jazz project who helped us during the study, in particular, Harold Ossher, Jean-Michel Lemieux, Li-te Cheng, Kate Ehrlich, and Tim Klinger.

We also thank the reviewers for their comments and feedback.

REFERENCES

[1] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting faults from cached history," in *Proceedings of the 29th International Conference on Software Engineering (ICSE 07)*. IEEE Computer Society, 2007, pp. 489–498.

[2] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, 2002.

[3] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (FSE 09)*. Amsterdam, The Netherlands: ACM, 2009, pp. 91–100.

[4] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *Proceedings of the International Conference on Software Maintenance (ICSM 03)*. IEEE Computer Society, 2003, p. 23.

[5] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering (PROMISE 07)*. IEEE Computer Society, 2007.

[6] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guhneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds (CASCON 08)*. Ontario, Canada: ACM, 2008, pp. 304–318.

[7] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," 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 (FSE 09)*. Amsterdam, The Netherlands: ACM, 2009, pp. 121–130.

[8] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM 07)*. IEEE Computer Society, 2007, pp. 364–373.

[9] N. Nagappan, L. Williams, M. Vouk, and J. Osborne, "Early estimation of software quality using in-process testing metrics: a controlled case study," in *Proceedings of the 3rd workshop on Software quality*. St. Louis, Missouri: ACM, 2005, pp. 1–7.

[10] M. Wheeler, *Lies, Damn Lies, and Statistics: The Manipulation of Public Opinion in America*. New York, NY: Liveright, 1976.

[11] G. Liebchen, B. Twala, M. Shepperd, M. Cartwright, and M. Stephens, "Filtering, robust filtering, polishing: Techniques for addressing quality in software data," in *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM 07)*. IEEE Computer Society, 2007, pp. 99–106.

[12] G. A. Liebchen and M. Shepperd, "Data sets and data quality in software engineering," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE 08)*. Leipzig, Germany: ACM, 2008, pp. 39–44.

[13] A. Mockus, *Guide to Advanced Empirical Software Engineering*, 1st ed. Springer London, 2008, ch. Missing Data in Software Engineering, pp. 185–200.

[14] A. E. Hassan, "Predicting faults using the complexity of code changes," *Proceedings of the 31st International Conference on Software Engineering (ICSE 09)*, pp. 78–88, 2009.

[15] R. A. Fisher, "On the interpretation of 2 from contingency tables, and the calculation of p," *Journal of the Royal Statistical Society*, vol. 85, no. 1, pp. 87–94, 1922.

[16] A. Strauss and J. Corbin, *Basics of qualitative research: Grounded theory procedures and techniques*. Newbury Park, CA: Sage, 1990.