# When Process Data Quality Affects the Number of Bugs: Correlations in Software Engineering Datasets

Adrian Bachmann and Abraham Bernstein
Department of Informatics
University of Zurich, Switzerland
{bachmann,bernstein}@ifi.uzh.ch

***Abstract*—Software engineering process information extracted from version control systems and bug tracking databases are widely used in empirical software engineering. In prior work, we showed that these data are plagued by quality deficiencies, which vary in its characteristics across projects. In addition, we showed that those deficiencies in the form of bias do impact the results of studies in empirical software engineering. While these findings affect software engineering researchers the impact on practitioners has not yet been substantiated. In this paper we, therefore, explore (i) if the process data quality and characteristics have an influence on the bug fixing process and (ii) if the process quality as measured by the process data has an influence on the product (i.e., software) quality. Specifically, we analyze six Open Source as well as two Closed Source projects and show that process data quality and characteristics have an impact on the bug fixing process: the high rate of empty commit messages in Eclipse, for example, correlates with the bug report quality. We also show that the product quality – measured by number of bugs reported – is affected by process data quality measures. These findings have the potential to prompt practitioners to increase the quality of their software process and its associated data quality.**

***Keywords*-case study; process quality; product quality; correlation; mining software repositories**

## I. Introduction

Empirical software engineering researchers make use of software process data such as bug reports and version control log files, e.g., to predict the number and locale of bugs in future software releases (e.g., [1], [2], [3], [4]). Unfortunately, the process data characteristics and quality may have a major impact on the results of studies that rely on such data. In prior work we, therefore, introduced process data quality and characteristics measures, which make an evaluation of process data possible. Based on these measures we surveyed five Open Source Software (OSS) projects and one Closed Source Software (CSS) project and showed that such process data is plagued by quality issues across all surveyed projects [5]. In addition to the quality issues in these process datasets, we showed that there are vast differences in the data characteristics of these projects often used in empirical software engineering studies.

These findings give rise to one main question: Why should one care about data quality and characteristics issues? In a first step, this lead to the following two research questions:

RQ 1: *Do quality issues in software engineering datasets have an impact on studies that rely on such datasets?*

RQ 2: *Do data characteristics, which differ from project to project, have an impact on studies that rely on such datasets?*

We tried to answer both questions in prior work. In [6] and [7] we investigated historical data from several software projects, and found strong evidence of systematic bias. We then investigated potential effects of "unfair" or "imbalanced" datasets on the performance of prediction techniques. Our experiments suggested that bug-feature bias affects the performance of the award-winning BugCache defect prediction algorithm and that this type of bias is a serious problem.
We also discussed the impact of varying data characteristics across projects [5] and concluded that, e.g., the nature of used software engineering processes, the use of process support tools, and kind of testing (e.g., professionalized testing vs. beta user testing) result in differing data characteristics. Such differences may raise threats in generalizability of research results, algorithms, tools, and heuristics across projects' borders.

Whereas RQ1 and RQ2 addressed the "why we should care about data quality and characteristics issues in empirical software engineering research", this paper would like to highlight why practitioners – software engineers and software project managers – should care about these issues. Therefore, we analyze the interplay of data quality and characteristics and its influence on the product (i.e., software) quality. Primarily, we answer the following two research questions:

RQ 3: *Do software engineering data quality and characteristics issues have an impact on the bug fixing process?*

RQ 4: *Does software engineering process quality have an immediate (or time-shifted) impact on product quality?*

Basically, we analyze several research hypotheses based on our two new research questions (RQ3 and RQ4). To test our hypotheses, we selected six OSS projects, which are widely used in empirical software engineering studies. In addition, we used the data of two CSS projects. This allows a first comparison of results across different software engineering processes, project communication habits, and testing approaches.

After the customary discussion of related work, this paper first explorers briefly the theoretical background of our study (Section III). We then present what software projects we selected, show what kind of data we used, and how we prepared them for this study (Section IV). In Section V, we

introduce our research hypotheses and provide a short contextual discussion. Also, we test our hypotheses by calculating Kendall tau rank correlations between the measures and the number of defects as software quality measure and provide our main contributions:

1) We find that process data quality and characteristics measures affect each other and correlate over time. For instance, we find—non surprisingly—that the rate of empty commit messages has an negative qualitative impact on the bug fixing process (RQ3).
2) We evaluate the correlation between process quality and product quality. In other words, we analyze immediate and time-shifted data correlations between process data quality measures and product quality measured by number of bugs reported and find that product quality is affected by part of our data quality measures (RQ4).

We close with threats to validity and the usual discussion and conclusions.

## II. Related Work

We discuss related work on data extraction and integration, data quality in software engineering, and work on data quality effects in empirical software engineering. To our knowledge, no studies on correlations between process data quality and characteristics and software product quality have been published so far.

### A. Process Data Extraction and Integration

Bug reports and version control log files are a valuable source of information about the history/evolution of a software project and are widely used in empirical software engineering research. The integration of these two data sources, e.g., by scanning trough commit log messages for bug report links, provides even more information. Therefore, several researchers provided approaches and algorithms to extract and integrate/link these data for applications which rely on such data.

Fischer *et al.* [8] presented a Release History Database (RHDB) which contains the version control log and the bug report information. To link the change log and the bug tracking database, Fischer *et al.* searched for change log messages which match to a given regular expression. Later, they improved this linking algorithm and built in a file-module verification [9].

A similar approach to link the change log with the bug tracking database was chosen by other researchers. All of them used regular expressions to find bug report link candidates in the change log file (e.g., [10], [2], [11], [12], [13], [11]).

In [14], we improved this well established linking approach to get a better linking rate and verified links at the same time. In addition, we presented a step-by-step guidance to retrieve, parse, convert and link the data sources. The datasets used in this study are prepared in this way.

### B. Quality in Software Engineering

Quality of software and source code is a widely explored field in research. But only a few publications focus on process data quality and characteristics. Due to space considerations we focus our discussion on recent work on process data quality and omit a detailed discussion of publications concerning software and source code quality.

Bettenburg *et al.* [15], [16], [17] provide an analysis of bug report quality. They investigated the attributes of a good bug report surveying developers and used it to develop a computational model of a bug report quality. Using the resulting model, the current bug report quality is displayed whilst typing. Hooimeijer *et al.* [18] also analyzed the quality of defect reports and tried to predict whether the defect report will be closed within a given amount of time.

Koru and Tian [19] surveyed members of 52 different medium to large size OSS projects to gain knowledge about the defect handling practices. They found that defect-handling processes varied among projects. Some projects are disciplined and require recording all bugs found; others are more lax. Some of the surveyed projects record defects only in source code while others record defects also in documents. This variation in bug handling practices may raise threats concerning cross project re-use of empirical work.

Chen *et al.* [20] studied the change logs of three OSS projects and analyzed the quality of these log files.

In [5] we defined process data quality and characteristics measures in order to provide a deeper insight into the quality and characteristics of these often-used process data. Then, we computed this measures for five OSS projects and one CSS project and discussed the issues arose from these observation. We showed that there are vast differences between the projects, particularly with respect to the quality in the linking rate between bug reports and commit log messages.

Paulson *et al.* [21] and Yu *et al.* [22] analyzed differences between OSS and CSS projects. Paulson *et al.* hypothesized that OSS has a higher quality. In addition, they provided five hypothesis analyzing them with the data of three CSS and OSS projects. They concluded that OSS projects foster more creativity and CSS projects are generally less defective since defects are found and fixed more rapidly. On the other hand, Yu *et al.* analyzed the average fault (defect) hidden time, average fault pending time, and average fault correction time in CSS projects and OSS projects. They concluded that defects are fixed more rapidly in OSS projects.

### C. Data Quality Effects on Empirical Software Engineering

Today, data quality issues and effects are not widely explored in empirical software engineering. Liebchen and Shepperd [23] surveyed hundreds of empirical software engineering papers to analyze how studies manage data quality issues. They found only 23 studies discussing data quality. Only four of the 23 suggested that data quality might impact analysis, but made no suggestion how to deal with it.

Unfortunately, effects of poor data quality on empirical software engineering is even less explored. In [6] and [7] we

investigated historical data from several software projects, and found strong evidence of systematic bias. Basically, we described two kinds of bias: bug feature and commit feature bias. Ideally, all bug-fixing commits are linked to bug reports; then empirical research would consider all type of fixed bug reports. However only some of the fixed bugs have links to the bug-fixing commits. This raises the possibility of two types of bias: bug feature bias, where only certain types of bugs are linked, or commit feature bias, whereby only certain types bug-fixing repairs are linked. With our experiments, we investigated potential effects of "unfair, imbalanced" datasets on the performance of prediction techniques, showed that bug-feature bias affects the performance of the award-winning BUGCACHE defect prediction algorithm.

Summarizing, the studies analyzing data quality and characteristics and show the effects on different types of outcomes. A study that shows the relationship between process data quality and product quality as investigated in this paper, however, seems to be missing.

## III. EVALUATION METHODS AND THEORY

In this section, we introduce the evaluation methods we used for this study. Additionally, we briefly discuss how we measured correlations in the datasets and how we defined process and product quality.

### A. Measurement of Process Quality

Current publications (e.g., [24], [25], [26]) present several approaches to measure and ensure quality in processes. For this study, we use a statistical approach that measures the quality of data provided by tools and systems used in these processes (such as bug tracking databases and version control systems). In particular, we make use of software engineering process data quality and characteristics measures introduced in [5]. Tables I and II summarize these measures including their definition.[1] Based on these measures, we are able to analyze the process characteristics and quality, compare the values across projects, and calculate correlation metrics.

TABLE I
PROCESS DATA QUALITY MEASURES

| Quality measure | Definition |
|---|---|
| Rate of fixed bug reports | #fixed bug reports / #bug reports |
| Rate of duplicate bug reports | #duplicate bug reports / #bug reports |
| Rate of invalid bug reports | #invalid bug reports / #bug reports |
| Rate of empty commit messages | #empty commit messages / #commit messages |
| Rate of commit messages with bug report links (w/o empty) | #commit messages with bug report links / #commit messages (w/o empty) |
| Rate of linked bug reports | #linked bug reports / #bug reports |
| Rate of linked bug reports (only fixed bug reports) | #linked bug reports / #fixed bug reports |

TABLE II
PROCESS DATA CHARACTERISTICS MEASURES

| Characteristics measure | Definition |
|---|---|
| Average status changes per bug report | #bug report status changes / #bug reports |
| Average comments per bug report | #bug report comments / #bug reports |
| Average attachments per bug report | #bug report attachments / #bug reports |
| Average bug reports per bug reporter | #bug reports / #bug reporters |
| Average length of commit messages (w/o empty) | sum of all message lengths / #commit messages (w/o empty) |
| Average number of bug report status changers per developer | #bug report status changers / #developers |
| Average bug reporters per developer | #bug reporters / #developers |
| Average bug reports per developer | #bug reports / #developers |
| Average fixed bug reports per developer | #fixed bug reports / #developers |
| Average bug report links per linked bug report | #bug report links / #linked bug reports |
| Average commits per bug report (all bug reports) | #commits / #bug reports |
| Average commits per bug report (only fixed bug reports) | #commits / #fixed bug reports |
| Average commits per developer | #commits / #developers |

### B. Measurement of Software Product Quality

The evaluation and measurement of software product quality is a widely explored field with many approaches. Already in 1978, for instance, Cavano and McCall presented a framework to measure software quality [27]. They defined the following software quality dimensions and factors:

- Product revision: maintainability, flexibility, and testability
- Product transition: portability, reusability, and interoperability
- Product operations: correctness, reliability, efficiency, integrity, and usability

We acknowledge that software engineering changed over the past few years and, therefore, many approaches for software quality management and measurement were newly developed. Nonetheless, we believe, that this framework still defines the most important factors for software quality factors although the labels changed somehow.

In our study, we mainly focus on the product operations aspect: i.e., quality factors affecting the users. These are typically reported as bugs, whenever they exhibit unwanted behavior (e.g., code is wrong, to slow, crashes, etc.). Luckily, it is very easy to evaluate the number of bugs, if we have access to a bug tracking database.

Measuring product quality by number of bugs, we asked ourselves what kind of bugs we should consider: all bugs or

[1] We refer to [5] for a full discussion of the measures.

only post-release bugs. Based on RQ4, we should consider all bugs which were released and available to the public in any release of the software. Irrelevant of the version in which version a bug was found: it was implemented and released without being noticed by the developers. Consequently, we define the product quality in this study as the number of bugs reported over time (pre- and post-release), which is common practice (see [24]).

### C. Evaluation of Correlations in Process and Product Quality

To evaluate the relation between each measure and the number of bugs, we computed the correlations between them for each week in a project. As correlation we used the Kendall tau ($\tau$) rank correlation coefficient [28]. In contrast to other correlation coefficient metrics (e.g., Spearman or Pearson), this correlation coefficient has *substantial advantages*. Basically, the Kendall tau rank correlation coefficient

1) makes no assumptions about the particular nature of the relationship between the variables (linear relationship not required),
2) does not require a normal distribution of the data,
3) does not require equidistance of the values, and
4) has a high robustness against outliers.

In addition, it allows an easy interpretation of values which lie between -1 and 1. Please note that *the correlation values of Kendall tau cannot be meaningfully compared to other rank correlation coefficients values* such as Spearman's $\rho$ as it usually generates lower correlation values. Since the interpretation of the Kendall tau rank correlation values is not standardized, we used the following interpretation schema [29]:

| | |
|---|---|
| $0.1 \leq \lvert\tau\rvert < 0.3$ | weak correlation |
| $0.3 \leq \lvert\tau\rvert < 0.5$ | moderate correlation |
| $0.5 \leq \lvert\tau\rvert \leq 1.0$ | strong correlation |

To test the $\tau$ correlation values for significance we calculated the two-sided $p$-values (t-test).

### D. Time-Shifted Correlations

Some effects may appear only with time delays. Bugs are usually discovered some time after the introducing commit (see Section V). Hence, we computed not only the correlation between the measures and the number of bugs within the same week but also calculated time-shifted correlations to uncover time-shifting effects. Essentially, we calculated the correlation between the measures at time $t = 0$ and the number of bugs reported at time $t \pm 0..50$ weeks. For all time constrained values we used the week of reporting or committing as time stamp.

### E. Correlations and Causality

It is important to note that all our explorations are based on correlations. Hence, we cannot make definitive statements on causality.

## IV. Used Datasets

For our experiments we selected eight medium to large scale projects with a long project history:

- Apache HTTP web server (OSS)
- Eclipse IDE (OSS)
- GNOME desktop suite (OSS)
- NetBeans IDE (OSS)
- OpenOffice.org productivity suite (OSS)
- Mozilla project (OSS)
- Banking system #1 (CSS)
- Banking system #2 (CSS)

The selected OSS projects are widely-used and well-known in empirical software engineering research. Many studies in this area rely on them. In addition, we selected two CSS datasets provided by the Zurich Cantonal Bank[2]. These two CSS datasets allow a slight insight into the commercial practices/processes and the resulting data quality and characteristics. For all OSS projects and one of the CSS projects we analyzed the data quality and characteristics presented in [5]. Therefore, due to space considerations, we omit a full discussion of the datasets. Unfortunately, due to security and confidentiality concerns, we are not allowed to publish detailed information about the CSS datasets. But both CSS projects are medium-scale banking software systems with many users or systems involved and have various releases over many years.

All considered OSS projects make use of BugZilla[3] or IssueZilla[4] as bug tracker and SVN[5] or CVS[6] as version control system. Both systems – bug tracker and version control system – allow free and open access to their contents.
The CSS projects, on the other hand, make use of HP Quality Center[7] as bug tracker and SVN as version control system. In contrast to the OSS projects, only a few people have the permission to access and modify the version control and bug data. Therefore, this data is not available to the public.

To test our hypotheses, we needed to obtain the process data from the source systems introduced above. Therefore, we retrieved, processed and linked these data as presented in [14]:

1) We downloaded or dumped all bug report information from the bug tracking database.
2) We downloaded the SVN or CVS log file.
3) We parsed the bug report information and SVN/CVS log data.
4) We linked these two data sources by scanning through all the commit log messages for bug report numbers.
5) We verified the caught bug report numbers for validity based on our heuristic.

Following this procedure, we created a process dataset for every software project introduced above. Table III lists some

[2]See http://www.zkb.ch
[3]See http://www.bugzilla.org/
[4]IssueZilla is no longer available online.
[5]See http://subversion.tigris.org/
[6]See http://www.nongnu.org/cvs/
[7]See http://www.hp.com

TABLE III
DETAILS OF SOFTWARE PROJECTS INVESTIGATED

| | Apache | Eclipse | GNOME | NetBeans | OpenOffice | Mozilla | BS #1 | BS #2 |
|---|---|---|---|---|---|---|---|---|
| **considered time period** | 2002-03-18 to 2008-04-30 | 2001-10-11 to 2008-02-29 | 2000-05-18 to 2008-09-30 | 2000-06-05 to 2008-04-30 | 2000-10-21 to 2008-04-30 | 1998-10-01 to 2009-08-31 | 2005-03-18 to 2008-02-29 | 2007-04-03 to 2009-07-30 |
| **# weeks** | 201 | 333 | 436 | 412 | 392 | 569 | 154 | 123 |
| **# bug reports (entries in the bug database)** | 4 997 | 215 298 | 492 107 | 127 421 | 88 837 | 495 985 | 7 843 | 640 |
| **# fixed bug reports**[8] | 1 439 | 112 309 | 113 303 | 66 786 | 34 586 | 158 386 | 4 449 | 304 |
| **# bug report duplicates** | 619 | 28 052 | 144 020 | 18 890 | 14 319 | 129 069 | 108 | 13 |
| **# bug report activities** | 19 152 | 1 412 467 | 1 973 620 | 923 764 | 684 988 | 5 384 816 | 114 109 | 8 108 |
| **# bug report comments** | 17 900 | 929 056 | 1 266 172 | 568 788 | 579 747 | 3 672 984 | 18 315 | 1 454 |
| **# bug report attachments** | 1 586 | 89 250 | N/A | 60 317 | 53 219 | 389 868 | 8 606 | 208 |
| **# bug reporters** | 3 510 | 18 836 | 158 561 | 11 410 | 19 707 | 122 325 | 133 | 39 |
| **# commit log revisions (transactions)** | 16 546 | 221 156 | 655 668 | 378 284 | 106 710 | 220 460 | 24 045 | 22 652 |
| **# developers (committers)** | 75 | 187 | 1 503 | 648 | 122 | 651 | 51 | 49 |

basic statistics for each of the project datasets and shows the time-periods we considered.

## V. RESEARCH HYPOTHESES AND RESULTS

In this section, we introduce our research hypotheses, provide a contextual discussion and test the hypotheses on our eight datasets.

### A. Interplay of Data Quality and Data Characteristics

In our prior work, we showed that the quality and characteristics of process data varies across projects [5]. Supplementary and detailed analysis showed that the quality and characteristics measures vary over time. Figure 1 shows the quality measures for the Eclipse project over the considered time period.

Due to similar value changes over time in many of the characteristics and quality measures, we assume that the phenomena they measure are connected. For example, empty commit messages (a data characteristic) are undesirable due to missing information about commit's reason and may influence the quality in bug reports (a quality characteristic).

HYPOTHESIS 1. *Empty commit messages have an impact on the bug report quality.*

The problem of empty commit messages arises mainly in the Eclipse dataset where about 20% of all commit messages are empty. In all other projects, this rate is far below 1% [5], e.g., due to commit message rules & regulations in these projects. Therefore, we are only able to test this hypothesis on the Eclipse project. Table IV lists the $\tau$ correlation coefficient values between the "Rate of empty commit messages" measure and the other quality measures. It also shows the two-sided $p$-values.

TABLE IV
CORRELATIONS FOR "RATE OF EMPTY MESSAGES" IN ECLIPSE

| Quality measure | $\tau$ value | $p$ value ($\tau$) | | $\rho$ value |
|---|---|---|---|---|
| Rate of fixed bug reports | $-0.096$ | $<8.652{\cdot}10^{-3}$ | | $-0.142$ |
| Rate of duplicate bug reports | $0.481$ | $<2.22{\cdot}10^{-16}$ | | $0.672$ |
| Rate of invalid bug reports | $0.481$ | $<2.22{\cdot}10^{-16}$ | | $0.671$ |
| Rate of commit messages with bug report links (w/o empty) | $-0.228$ | $<5.397{\cdot}10^{-10}$ | | $-0.337$ |
| Rate of linked bug reports | $0.475$ | $<2.22{\cdot}10^{-16}$ | | $0.671$ |
| Rate of linked bug reports (only fixed bug reports) | $0.488$ | $<2.22{\cdot}10^{-16}$ | | $0.691$ |

As already mentioned in Sub-Section III-C, the Kendall $\tau$ rank correlation values are usually lower than other correlation values. For illustration purposes, Table IV contains also the Spearman's $\rho$ correlation values (right part of the table).

The $\tau$ values in the table show a moderate to strong positive correlation between the "Rate of empty messages" and the proportion of duplicate, invalid and linked bug reports. Please note, that the "Rate of fixed bug reports" does not exhibit any correlation. Based on the correlation values, we can conclude that the more empty messages the more duplicate and invalid bug reports we have. On the other hand, we seem to have also a beneficial effect, due the positive correlation to linked bug reports. Because we were able to test this hypothesis with Eclipse data only, we acknowledge possible generalization threats.

[8]We define "fixed" bug reports as bug reports that have at least one associated fixing activity (i.e., status change to "fixed").

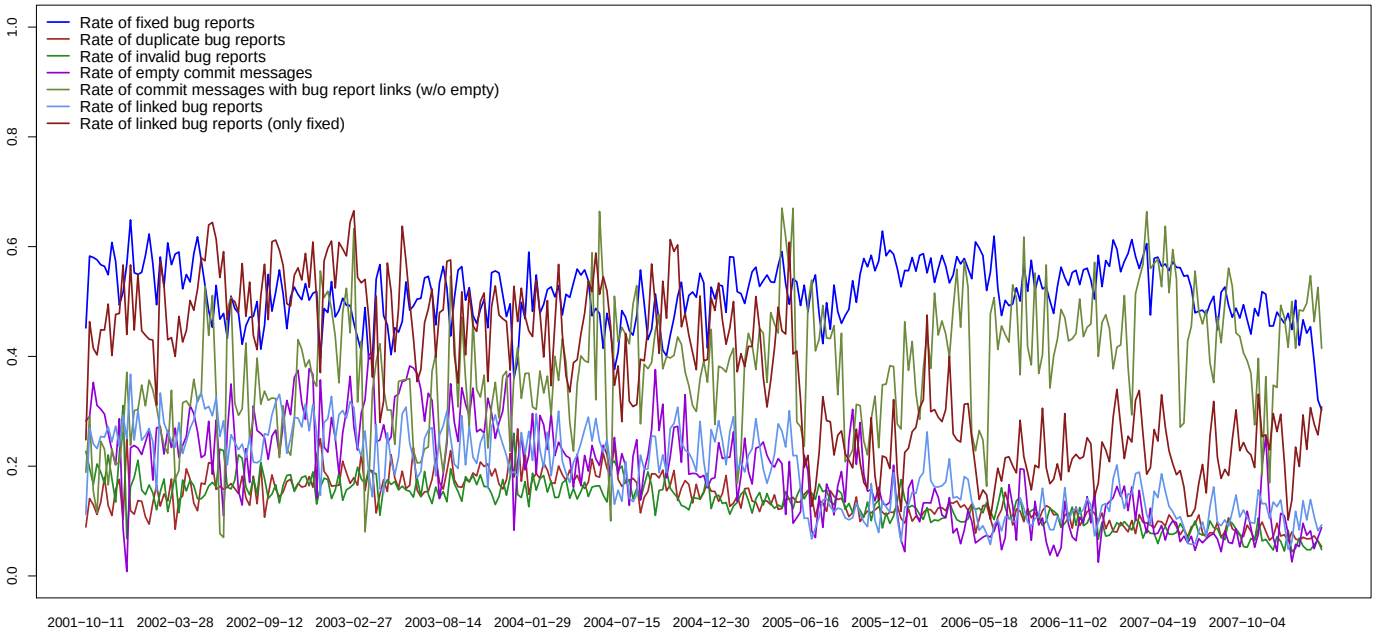


Fig. 1. History of data quality measure values for Eclipse (weekly frames)

The results of empirical software engineering studies rely on completeness in datasets. Therefore, it is essential at which rate bug reports are mentioned in the datasets. As a common practice, we scan trough commit messages for valid bug report numbers and then link commit messages with bug reports (see Section IV). Unfortunately, as already shown in previous work [5], only a fraction of fixed bug reports are linked. Such linking has to be established by developers, which may e.g., miss the linking. Therefore, we theorize the higher the ratio between number of commits and bug reports, the higher the chance (and probability) to get linked bug reports.

HYPOTHESIS 2. *The higher the ratio between number of commits and bug reports, the higher the chance to have linked bug reports.*

We can observe a strong correlation between "Average commits per bug report" and "Rate of linked bug reports" in Eclipse and Mozilla, a weak correlation in GNOME, NetBeans, and BS #2, as well as almost no correlation in Apache, OpenOffice and BS #1 (Table V). Therefore, we have positive correlations supporting our hypothesis in four datasets. Three projects do not support the hypothesis and for NetBeans we have a negative correlation which also contradicts our hypothesis.

With our hypothesis, we theorize that bug report linking varies because developers link bug reports sometimes and sometimes not. In other words: we assume inconsistent developers that link with some probability. Missing correlations in three projects may be an indicator for strict habits of developers in these projects. Hence, some bugs get linked and some bugs not (e.g., consistently based on bug report features), which may be an indicator of bug-feature bias [7].

Usually, bugs are reported by users who discover an unwanted behavior of the system during its use. Particularly after new releases, more bugs are found and get reported. This leads also to a varying number of bug reports over time to take care of by developers. Given that developer work time is a limited resource, we hypothesize that when the workload – measured by number of bug reports to fix – goes up, developers will only be able to attend a smaller fraction of them – measured by bug reports that get fixed and linked to source code commits.

TABLE V
CORRELATIONS BETWEEN "AVERAGE COMMITS PER BUG REPORT" AND "RATE OF LINKED BUG REPORTS"

| **Project** | $\tau$ **value** | $p$ **value** |
|---|---|---|
| Apache | $-0.028$ | $<5.818 \cdot 10^{-1}$ |
| Eclipse | $0.574$ | $<2.22 \cdot 10^{-16}$ |
| GNOME | $0.239$ | $<2.22 \cdot 10^{-16}$ |
| NetBeans | $-0.103$ | $<1.864 \cdot 10^{-3}$ |
| OpenOffice | $-0.063$ | $<6.317 \cdot 10^{-2}$ |
| Mozilla | $0.529$ | $<2.22 \cdot 10^{-16}$ |
| BS #1 | $-0.003$ | $<9.607 \cdot 10^{-1}$ |
| BS #2 | $0.181$ | $<7.269 \cdot 10^{-3}$ |

HYPOTHESIS 3. *The higher developer workload, the fewer bug reports get fixed and linked to source code commits.*

Mozilla and Gnome have moderate negative correlations between developer workload and fixing rate. This means, that in these projects the more bugs a developer has to fix (on average) the less bugs get fixed. Contrariwise, both projects have a high bug duplicate rate of about 33% (see Table III) which may have an effect such as duplicates usually never get fixed rather than merged with other bug reports. OpenOffice also has a weak negative correlation with a similar finding, that this project has a bug duplicate rate above average. Eclipse

TABLE VI
CORRELATIONS BETWEEN "AVERAGE BUG REPORTS PER DEVELOPER" AND "RATE OF FIXED BUG REPORTS" OR "RATE OF LINKED BUG REPORTS"

| | "Rate of fixed bug reports" | | "Rate of linked bug reports" | |
|---|---|---|---|---|
| **Project** | $\tau$ **value** | $p$ **value** | $\tau$ **value** | $p$ **value** |
| Apache | 0.032 | $<5.103{\cdot}10^{-1}$ | 0.021 | $<6.842{\cdot}10^{-1}$ |
| Eclipse | 0.160 | $<1.398{\cdot}10^{-5}$ | −0.407 | $<2.22{\cdot}10^{-16}$ |
| GNOME | −0.449 | $<2.22{\cdot}10^{-16}$ | −0.209 | $<7.215{\cdot}10^{-11}$ |
| NetBeans | −0.080 | $<1.531{\cdot}10^{-2}$ | 0.005 | $<8.751{\cdot}10^{-1}$ |
| OpenOffice | −0.261 | $<1.382{\cdot}10^{-14}$ | 0.132 | $<1.053{\cdot}10^{-4}$ |
| Mozilla | −0.412 | $<2.22{\cdot}10^{-16}$ | −0.550 | $<2.22{\cdot}10^{-16}$ |
| BS #1 | 0.022 | $<6.852{\cdot}10^{-1}$ | 0.109 | $<5.477{\cdot}10^{-2}$ |
| BS #2 | 0.327 | $<1.783{\cdot}10^{-6}$ | 0.333 | $<8.007{\cdot}10^{-7}$ |

and BS #2 have weak positive correlations. We believe this to be related to Hypothesis 2: both projects have positive correlations (Eclipse $\tau = 0.574$, BS #2 $\tau = 0.181$) between "Average commits per bug report" and "Rate of linked bug reports". Hence, the more bugs get reported (and therefore the developer's workload and the proportion of bugs to commits arise), the higher the fixing rate. In contrast, Mozilla and Gnome have also high correlations between "Average commits per bug report" and "Rate of linked bug reports" but have negative correlations between developer workload and "Rate of fixed bug reports". All other projects have no significant correlations in the data (left part of Table VI).
Regarding the correlations between developer workload and "Rate of linked bug reports", Eclipse, Gnome, and Mozilla support the hypothesis, that the higher the developer's workload the lower the "Rate of linked bug reports". In contrast, OpenOffice and interestingly both CSS projects have a weak to moderate positive correlation (right part of Table VI). These three projects have less developers compared to the other projects, which may have an impact on the correlations.

In OSS projects it is a usual practice to release alpha and beta versions of a product and let users perform testing. In the CSS projects investigated, in contrast, a professionalized product testing is performed by a few testers [5]. These testers follow a defined test-procedure composed of test-cases, test-scenarios, and test-data. Whereas in professionalized testing only a few people report bugs, in OSS alpha and beta testing many people use/test a new release and report bugs. Therefore, in many OSS projects, most of users only report a few bugs in their life, potentially leading to a lower quality of the bug reports due to lacking experience. A poor bug report may lower the probability of it being fixed, as a developer may not understand it.

HYPOTHESIS 4. *The more experienced the bug reporters, the higher the chance to get fixed bug reports.*

Mozilla, GNOME, Eclipse, and BS #2 have a moderate to strong positive correlation; Apache has a weak positive correlation. All these projects support our hypothesis with very low $p$-values that the more bugs a reporter reports, the higher the linking rate of its bug reports. Only NetBeans and OpenOffice weakly reject the hypothesis (Table VII). Despite these two counterexamples we believe that the evidence of the other projects suggests that the more bugs somebody reports, the better the quality of these reports, and, therefore, the higher the chance for a developer to understand and fix the problem. This finding justifies methods for enhancement of bug report quality (e.g., Bettenburg *et al.* [15], [16], [17]).

TABLE VII
CORRELATIONS BETWEEN "AVERAGE BUG REPORTS PER BUG REPORTER" AND "RATE OF FIXED BUG REPORTS"

| **Project** | $\tau$ **value** | $p$ **value** |
|---|---|---|
| Apache | 0.123 | $<1.855{\cdot}10^{-2}$ |
| Eclipse | 0.342 | $<2.22{\cdot}10^{-16}$ |
| GNOME | 0.427 | $<2.22{\cdot}10^{-16}$ |
| NetBeans | −0.118 | $<3.611{\cdot}10^{-4}$ |
| OpenOffice | −0.192 | $<1.577{\cdot}10^{-8}$ |
| Mozilla | 0.604 | $<2.22{\cdot}10^{-16}$ |
| BS #1 | 0.054 | $<3.305{\cdot}10^{-1}$ |
| BS #2 | 0.417 | $<6.492{\cdot}10^{-9}$ |

Important bugs usually get high attention by the community and, therefore, many users and developers make use of the discussion function of the bug tracker. We hypothesize that the more attention a bug has (measured by number of comments), the higher the chance a developer mentions the report in the commit message of the bugfix.

HYPOTHESIS 5. *The more people discuss bug reports, the higher the chance for bug reports to get linked.*

TABLE VIII
CORRELATIONS BETWEEN "AVERAGE COMMENTS PER BUG REPORT" AND "RATE OF LINKED BUG REPORTS"

| **Project** | $\tau$ **value** | $p$ **value** |
|---|---|---|
| Apache | 0.255 | $<3.682{\cdot}10^{-7}$ |
| Eclipse | −0.026 | $<4.702{\cdot}10^{-1}$ |
| GNOME | 0.605 | $<2.22{\cdot}10^{-16}$ |
| NetBeans | 0.323 | $<2.22{\cdot}10^{-16}$ |
| OpenOffice | 0.112 | $<1.025{\cdot}10^{-3}$ |
| Mozilla | 0.297 | $<2.22{\cdot}10^{-16}$ |
| BS #1 | 0.227 | $<5.909{\cdot}10^{-5}$ |
| BS #2 | 0.348 | $<4.148{\cdot}10^{-7}$ |

This hypothesis is supported by the strong correlation in Gnome and the weak to moderate correlations in all other projects except of one (Table VIII). Eclipse neither supports nor contradicts this hypothesis with $\tau = -0.026$. We believe, that high attention of the community acts as incentive to developers for a better documentation of their work (which includes the linking of bug fixes to bug reports). On the other hand, without attention from the community, a developer has less benefits from linking bug fixes and, therefore, might act not that strictly.

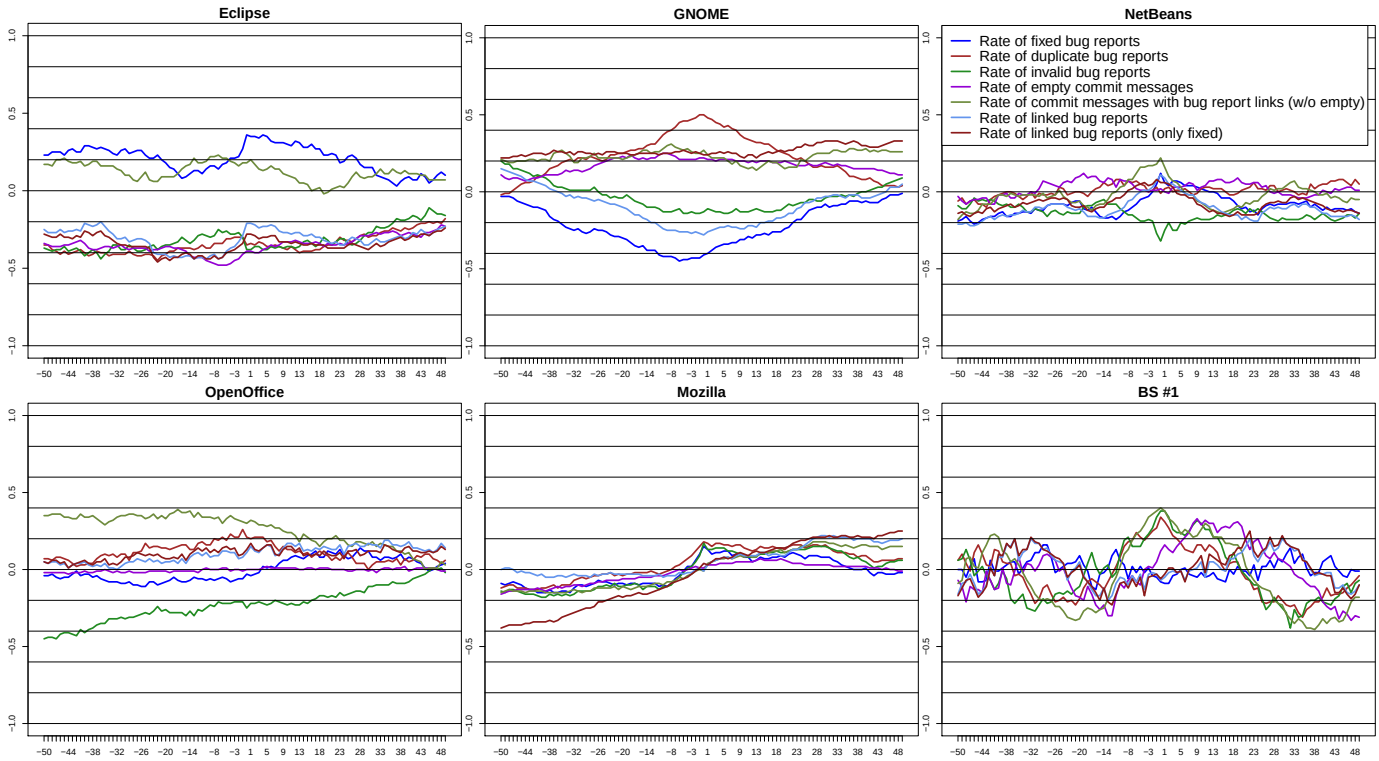


Fig. 2. Time-shifted correlations between process quality measures and number of defects ($t \pm 0..50$ weeks)

### B. *Influence of Process Quality on Product Quality*

With our second research question (RQ4), we try to evaluate the impact of process quality on product quality. Again, we present several hypotheses, provide a contextual discussion and discuss the results based on our project datasets. Unfortunately, we found no significant product quality correlations in Apache and BS #2. Therefore, we omit a discussion of these two projects in this sub-section.

First, we hypothesize that an impact of process quality on product quality may be time-shifted. E.g., when the process quality drops then more bugs get introduced later e.g., due to poor documentation quality. Hence, we test our hypotheses using time-shifting correlations. Figure 2 illustrates the time-shifted $\tau$ correlation values between our process quality measures and the number of bugs. For each project we claculated the correlation between: quality measures at time $t = 0$ and number of bugs at time $t \pm 0..50$ weeks. Interestingly, most correlations have their maximum values at $\Delta t = 0$ (except for OpenOffice). Hence, we do not seem to have any time-shifting effects in these datasets. A finding we find surprising given our previous work [4], [30].

Particularly in OSS projects, all users of a system are allowed to report bugs (see discussion above). Hence, many duplicate and invalid bug reports are stored in the bug tracking database (as shown in [5]) and need attention of developers. If duplicate bug reports are valuable or not, is a controversial discussion in current research. For instance, Bettenburg *et al.* believe, that duplicates provide additional information to a specific problem and are not harmful per se [17]. Nevertheless, duplicate and invalid bug reports may increase the needed effort by developers to fix a specific problem and may raise the risk, that multiple developers work on similar problems. This could give rise to two effects: (1) developers have less time to carefully implement bug fixes and feature requests, and (2) a synchronous work of multiple developers on the same problem may lead to conflicts. Therefore, we believe that duplicate and invalid bug reports may influence the number of (future) bugs.

HYPOTHESIS 6. *The proportion of duplicate and invalid bugs correlate with the number of bugs.*

Analyzing Figure 2, we see that the maximum values lie at $\Delta t = 0$, which we also verified statistically. Therefore, we present only the $\tau$ and $p$ values for $\Delta t = 0$ (Table IX).

TABLE IX
CORRELATIONS BETWEEN "NUMBER OF BUG REPORTS" AND "RATE OF DUPLICATE BUG REPORTS", $\Delta t = 0$

| **Project** | $\tau$ **value** | $p$ **value** |
|---|---|---|
| Eclipse | $-0.304$ | $<2.22 \cdot 10^{-16}$ |
| GNOME | $0.640$ | $<2.22 \cdot 10^{-16}$ |
| NetBeans | $0.086$ | $<8.891 \cdot 10^{-3}$ |
| OpenOffice | $0.310$ | $<2.22 \cdot 10^{-16}$ |
| Mozilla | $0.169$ | $<1.696 \cdot 10^{-9}$ |
| BS #1 | $0.424$ | $<1.147 \cdot 10^{-11}$ |

GNOME, BS #1 and OpenOffice have a moderate to strong and Mozilla a weak positive correlation and, therefore, support Hypothesis 6. Only Eclipse contradicts our hypothesis with a

moderate negative correlation ($\tau = -0.304$). We believe that in the Eclipse project duplicate bug reports are not a substantial problem and provide additional information to the developers as proposed by Bettenburg *et al.*. Otherwise, a negative correlation would not be meaningful. For the correlations with invalid bug reports, we got similar correlations and omit a discussion due to space considerations.

A version control system allows a concurrent development and tracks all changes: all source code changes are associated with a developer and change-time. In addition, developers usually describe in the commit message why a change was done. In the bug fixing process, such commit messages ideally contain links to bug reports they fix. Hence, these changes are justified and other developers can follow the changes' rationale. Explicit rationale is conducive to an efficient maintenance of the product by developers.

Consequently, we hypothesize that the better source code changes (commits) are documented, the less bugs get implemented in future releases. E.g., we can assume, the higher the linking rate of bug reports, the more bug fixing commits are documented/justified and therefore, the less bugs get introduced.

HYPOTHESIS 7. *The proportion of linked bugs correlate with the number of bugs.*

Again, we calculated the $\tau$ correlation values between the number of bugs and "Rate of linked bug reports" for $t \pm 0..50$ weeks. Unfortunately, we did not found any significant correlations for any $\Delta t$ in any project and have to reject Hypothesis 7.

Nonetheless, for similar reasons, we believe that empty commit messages have an impact on the number of bugs. Empty commit messages provide no information about the commit's reason. Again, we hypothesize that a good documentation of source code changes allow a faster and less buggy maintenance and enhancement of a product. We therefore believe, that developers tend to implement more bugs due to the missing information in commits.

HYPOTHESIS 8. *The proportion of empty commit messages correlate with the number of bugs.*

As already mentioned in Hypothesis 1, Eclipse is the only project with a high "Rate of empty commit messages". Therefore, we were only able to calculate the correlation for this project. Interestingly, we found an almost strong negative correlation of $\tau = -0.483$ with a two-sided $p$-value of $< 2.22 \cdot 10^{-16}$ at $t = -6$. The negative correlation indicates that when the "Rate of empty commit messages" sinks, the number of bug reports rises. Or in other words: if we have less empty commit messages, we get more bug reports. The time-shift of $t = -6$ signifies that a high number of bug reports correlates with a high number of empty messages six weeks later. This is exactly the opposite of what we hypothesized. Hence, it could be hypothesized that an increase of bug reports leads to "overworked" developers who then lower the quality of their reports to save time and catch up.

## VI. THREATS TO VALIDITY

Software engineering tools and processes vary in different projects and, therefore, our findings based on the used projects may not generalize. However, we analyzed often-used and well-known OSS projects. Therefore, it is reasonable to conclude that the selected projects are no exception in OSS engineering.
On the other hand, we were only able to analyze two CSS projects, both provided by the Zurich Cantonal Bank. Therefore, we acknowledge threats in generalizing these results to other CSS projects.

Our choice of number of bugs as the only indicator for software product quality gives rise to internal and/or construct validity issues. In particular, we counted bugs when they were reported on the bug tracking database. The reported bugs were introduced a varying time before they were found and reported. Due the variable delay between the bug introducing commit and bug report, we may have time shifting threats in our correlation calculations. We evaluated the number of developers based on the version control log file. In other words, a developer equals to a committer. This may raise to threats, as not all OSS developers are allowed to commit and, therefore, are not counted in this study. Also, some bugs may not have been reported in the bug tracking system or not at all.

## VII. DISCUSSION AND CONCLUSIONS

In prior work, we analyzed the impact of poor data quality in software engineering on empirical software engineering research. We found that considering data quality concerns is crucial for research in empirical software engineering. In this paper, we extend this work and show why practitioners – software engineers and software project managers – should care about the quality of their processes and data.

Specifically, we analyzed two research questions and defined several research hypotheses. To test these hypotheses, we extracted and integrated software engineering process data from six widely-used OSS and two CSS projects as presented in previous work [14]. Using these datasets, we calculated data quality and characteristics measures [5] for each week and project. We then calculated Kendall tau rank correlations based on this measures. Later we extended our calculations with time-shifted correlations to analyze the data for time-shifting effects.

In particular, we showed that data quality and characteristics issues affect each other. For instance, the proportion of empty commit messages in Eclipse correlate with bug report quality. In addition, we showed, that the more active a community in its communication – such as the discussion and commenting bug reports – the better the linking rate provided by the developers. We also showed that product quality – measured by number of bugs reported – is affected by process data quality. Again, the proportion of empty messages leads to

curious findings in Eclipse. These findings have the potential to prompt practitioners to increase the quality of their software process and its associated data quality.

Nonetheless, we were only able to support some of our hypotheses. Therefore, this work can only be a first step in analyzing interrelationship between process (data) quality and software product quality. In future work, we want to extend this study and analyze subsets of bug datasets. For instance, we hypothesize that only bugs with a high severity correlate with process measures, because trivial bugs are not affected by process quality. In addition, we plan to verify process quality correlations for other definitions of product quality and define new quality and characteristics measures for other aspects of software engineering.

Furthermore, we plan to extend our study on other CSS datasets to get a deeper insight into the quality and characteristics of CSS process data and their correlations to product quality.

## ACKNOWLEDGMENT

This work was partially supported by Zurich Cantonal Bank (Bachmann) and Swiss National Science Foundation award number 200021-112330 (Bernstein). The closed source data used in this work was provided by the Zurich Cantonal Bank.

## REFERENCES

[1] M.-T. J. Ostrand, F.-E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, 2005.

[2] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *MSR '05: Proceedings of the 2005 International Workshop on Mining Software Repositories*. New York, NY, USA: ACM, 2005.

[3] H. Joshi, C. Zhang, S. Ramaswamy, and C. Bayrak, "Local and global recency weighting approach to bug prediction," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2007.

[4] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," in *IWPSE '07: Proceedings of the International Workshop on Principles of Software Evolution*. Dubrovnik, Croatia: IEEE Computer Society Press, September 2007, pp. 11–18.

[5] A. Bachmann and A. Bernstein, "Software process data quality and characteristics - a historical view on open and closed source projects," in *IWPSE-Evol'09: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, Amsterdam, The Netherlands, August 2009, pp. 119–128.

[6] E. Aune, A. Bachmann, A. Bernstein, C. Bird, and P. Devanbu, "Looking back on prediction: A retrospective evaluation of bug-prediction techniques," Student Research Forum at SIGSOFT/FSE 16, November 2008.

[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 *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering*, Amsterdam, The Netherlands, August 2009, pp. 121–130.

[8] M. Fischer, M. Pinzger, and H. C. Gall, "Populating a release history database from version control and bug tracking systems," in *ICSM '03: Proceedings of the International Conference on Software Maintenance*. Amsterdam, Netherlands: IEEE Computer Society Press, September 2003, pp. 23–32.

[9] M. Fischer, M. Pinzger, and H. Gall, "Analyzing and relating bug report data for feature tracking," in *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*. Victoria, B.C., Canada: IEEE Computer Society Press, November 2003, pp. 90–99.

[10] D. Čubranić and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," in *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 408–418.

[11] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, "If your bug database could talk..." in *ISESE '06: Proceedings of the 5th International Symposium on Empirical Software Engineering. Volume II: Short Papers and Posters*, Rio de Janeiro, Brazil, September 2006, pp. 18–20.

[12] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007.

[13] T. Zimmermann and P. Weissgerber, "Preprocessing cvs data for fine-grained analysis," in *MSR '04: Proceeding of the 1st International Workshop on Mining Software Repositories (MSR)*, Edinburgh, UK, 2004.

[14] A. Bachmann and A. Bernstein, "Data retrieval, processing and linking for software process data analysis," Dynamic and Distributed Information Systems Group, Department of Informatics, University of Zurich, Technical Report IFI-2009.0003, May 2009, http://www.ifi.uzh.ch/ddis/people/adrian-bachmann/pdq/.

[15] N. Bettenburg, S. Just, A. Schroeter, C. Weiss, R. Premraj, and T. Zimmermann, "Quality of bug reports in eclipse," in *eTX '07: In Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, Montreal, Quebec, Canada, October 2007, pp. 21–25.

[16] N. Bettenburg, S. Just, A. Schroeter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" Saarland University, Saarbrücken, Germany, Tech. Rep., September 2007.

[17] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful... really?" in *ICSM '08: Proceedings of the International Conference on Software Maintenance*, October 2008, pp. 337–345.

[18] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *ASE07: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Atlanta, Georgia, USA, November 2007, pp. 34–43.

[19] A. G. Koru and J. Tian, "Defect handling in medium and large open source projects," *IEEE Software*, vol. 21, no. 4, pp. 54–61, 2004.

[20] K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller, "Open-source change logs," *Empirical Software Engineering*, vol. 9, no. 3, pp. 197–210, 2004.

[21] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *IEEE Transactions on Software Engineering*, vol. 30, no. 4, pp. 246–256, 2004.

[22] L. Yu and K. Chen, "Evaluating the post-delivery fault reporting and correction process in closed-source and open-source software," in *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops, Fifth International Workshop on Software Quality (WoSQ'07)*. Washington, DC, USA: IEEE Computer Society, 2007.

[23] G. A. Liebchen and M. Shepperd, "Data sets and data quality in software engineering," in *PROMISE '08: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2008, pp. 39–44.

[24] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed. Addison Wesley, September 2002.

[25] N. Ashrafi, "The impact of software process improvement on quality: in theory and practice," *Information Management*, vol. 40, no. 7, pp. 677–690, 2003.

[26] D. E. Harter and S. A. Slaughter, "Process maturity and software quality: a field study," in *ICIS '00: Proceedings of the twenty first international conference on Information systems*. Atlanta, GA, USA: Association for Information Systems, 2000, pp. 407–411.

[27] J. P. Cavano and J. A. McCall, "A framework for the measurement of software quality," in *Proceedings of the software quality assurance workshop on Functional and performance issues*, 1978, pp. 133–139.

[28] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, June 1938.

[29] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. New Jersey: Lawrence Erlbaum Associates, Inc., 1988.

[30] J. Ekanayake, J. Tappolet, H. C. Gall, and A. Bernstein, "Tracking concept drift of software projects using defect prediction quality," in *MSR '09: Proceedings of the Sixth IEEE Working Conference on Mining Software Repositories*. IEEE Computer Society, May 2009.