Computer Science and Information Systems 13(1):173–197 DOI: 10.2298/CSIS141228061M

A Systematic Data Collection Procedure for Software Defect Prediction

Goran Mauša[1], Tihana Galinac Grbac[1], and Bojana Dalbelo Bašić[2]

[1] Faculty of Engineering, Vukovarska 58
51 000 Rijeka, Croatia
gmausa@riteh.hr, tgalinac@riteh.hr
[2] Faculty of Electrical Engineering and Computing, Unska 3
10 000 Zagreb, Croatia
bojana.dalbelo@fer.hr

Abstract. Software defect prediction research relies on data that must be collected from otherwise separate repositories. To achieve greater generalization of the results, standardized protocols for data collection and validation are necessary. This paper presents an exhaustive survey of techniques and approaches used in the data collection process. It identifies some of the issues that must be addressed to minimize dataset bias and also provides a number of measures that can help researchers to compare their data collection approaches and evaluate their data quality. Moreover, we present a data collection procedure that uses a bug-code linking technique based on regular expression. The detailed comparison and root cause analysis of inconsistencies with a number of popular data collection approaches and their publicly available datasets, reveals that our procedure achieves the most favorable results. Finally, we implement our data collection procedure in a data collection tool we name the Bug-Code (BuCo) Analyzer.

Keywords: software defect prediction, data collection issues, dataset bias, bug-code linking, open-source projects

1. Introduction

The first step of any empirical research is data collection. Empirical research is highly dependent on the quality of the data used. Low-quality data are not a good source for the construction of new theories. The threats to the generality of the conclusions drawn from such data are predominantly posed by bias. That is why there are guidelines for data collection in various research areas. The area of software defect prediction (SDP) lacks such a set of guidelines for collection and validation of data. Instead, there are some often used practices that were never thoroughly compared against a set of rules nor against each other. The goal of our research is a systematically defined data collection procedure for SDP studies. So far, we have addressed and evaluated the issues that may become a significant source of bias if left open to interpretation [26] and we have developed a tool for automated data collection [25]. In this paper, we aim to make the data collection guidelines complete by proposing how to perform a systematic comparison of data collection procedures. Then we use it for an empirical study to compare our own data collection procedure with existing practices and to study the effectiveness of linking techniques in different contexts.

SDP deals with finding a predictive model that would identify areas of source code that are most prone to defects. Such a recommendation system require a data collection mechanism and a recommendation engine to collect development data, to analyze the data model and to generate recommendations [32]. The majority of studies suffers from the generalization of results and unknown sources of bias that are not properly addressed [17]. SDP research relies strongly on data from several software development repositories. Linking the data across disparate sources from the software life cycle and matching all records related to the same software entity offers a powerful resource for the study of software behavior. Data linking has also been recognized as a powerful tool for gaining new knowledge and developing new theories in many other disciplines, such as health [21], national statistics [15], business, bioinformatics, and climate change. However, the quality of the collected data is influenced by the quality of the reporting system. If the reporting has no standardized methodologies, the quality of collected data will be deteriorated. The data collection procedures should be verifiable, repeatable and accurate [4], [17]. Numerous efforts have been devoted to identifying sound techniques, procedures and useful tools that could be used for such purposes. Some examples include the efforts put toward constructing traceability requirements [1], bug-code linking [38], [30], and bug localization [23], [40]. However, none of the existing approaches seams to have achieved satisfactory accuracy and bias seams to be the dominant problem [42]. Furthermore, many repositories suffer from data quality issues, such as outliers, missing values, redundant or irrelevant attributes and instances, overlapping classes, contradictory samples, data shifting, imbalance of classes or unreliable replicability [37], [33]. Hence, the SDP community must develop a systematic approach to assess the reproducibility of empirical studies based on the datasets retrieved from development repositories [16].

The quality of the linked data has been addressed in other research fields as is reviewed in [9] and it depends on data linking approach. If there is no standard for data linking in SDP studies, datasets may suffer from bias and the generalisation of results cannot be achieved. Thus, we examine the data collection procedures used in the field of SDP research, investigate their weaknesses, and reveal hidden issues. Using supporting tools and datasets, we quantitatively evaluate the effectiveness of already available techniques and develop our own. Our contribution to the research community will be to identify concrete issues that must be discussed when applying a particular linking technique, to address threats to its validity and the generalization of its results, and a set of guidelines how to systematically evaluate its performance in comparison to already existing ones. In this paper we distinguish two terms in data collection: **linking technique** that is a matching technique to connect the pieces of information from separate repositories, and **data collection procedure** that includes a particular linking technique and a set of decisions that define which data are going to be collected. The output of a linking technique are bug file pairs and the output of a SDP data collection procedure are SDP datasets that consist of files described with software metrics and number of associated bugs in a given release of a software product.

Our research questions are as follows:

RQ1 How strongly is linking effectiveness influenced by development context and why?
RQ2 Which linking technique identifies the highest number of correct links?

We use the term *development context* as a general description of specificities behind a development process. It comprises not only the structure of the repository, but also the specifics of the development process, the habits of the developers and the procedures and rules they follow. The habits of developers within a community are typically more similar than the habits of developers across multiple communities because they are governed by strict naming conventions, rules and procedures that are not part of general development practice. For example, the Eclipse community has its own naming conventions and informal rules that may have significant influence on the data quality it produces and the linking techniques that can be applied. How to establish context is a difficult task and a general challenge [32]. In our analysis, each dataset is regarded as a separate development context.

The contributions of this paper are the presentation of an exploratory study with the goal of identifying all issues present in data collection, the performance of a comparison experiment among bug-code linking techniques, the performance of a comparison experiment between the proposed data collection approach and other popular datasets and the development of a tool for the automatic replication of the entire data collection process for a wider range of projects. The remainder of this paper is organized as follows: Section 2 presents the related work in the field of data collection for the SDP research, focusing on various linking techniques, approaches and reporting quality; Section 3 describes the details of our empirical study; Section 4 presents the results of the quantitative assessments and root cause analysis of our experiment, answers the research questions and discusses the threats to validity; and Section 5 summarizes the paper with a discussion and conclusion.

2. Background

2.1. Linking techniques

In related work a various approaches to bug-code linking are explored that we mention below. The simplest one, the deterministic approach, is about joining data from diverse repositories based on a unique entity identifier that is available in all repositories [5]. For example, the *Bug ID* is typically used as a unique identifier to connect the bugs from a bug repository with the related source code changes in a source code management repository. These source code changes, i.e., commits, are briefly explained in commit messages, which is where the Bug ID is sought. We use this technique in our experiment and refer to it as the **Simple Search** approach. However, such a unique entity identifier is not always present. In the example of bug-fix connections, for most source code repositories, it is not mandatory to state the Bug ID in the commit message. Thus, we are unable to discover all bug-fix links using the Simple Search technique. The explicit inclusion of the Bug ID in the commit message is highly dependent on the culture and habits of the open-source development community and the maturity of the organization. This is a crucial issue that can easily lead to biased data, and it is the focus of this paper. The positive correlation between project members' experience (which may also be related to the maturity of the organization) and the presence of linkage information was confirmed in [6]. Thus, the deterministic approach can provide good linking results only if the entity identifiers or linkage keys are of high quality, meaning that they are precise, stable over time, available and robust with regard to errors. Moreover, the usage of different linkage keys, as presented in [5] for the linking of data in the health domain, may result in differing data quality. In this paper, one of the analyses investigates the effect of using different linkage keys in the same repository. Another analysis investigates the effect of using the most commonly used linkage key across the most often used repositories in SDP research.

The probabilistic linking approach is based on the existing common attributes in the repositories. For example, links are confirmed only if their common attributes predominantly agree, and they are not confirmed if these attributes predominantly disagree [13]. This approach has been applied to create SDP datasets in several studies. An algorithm based on regular expressions that matches the file names present in both the bug tracking and source code management repositories is proposed in [14].

The **SZZ approach**, proposed by Śliwerski, Zimmermann and Zeller in [38], is another example of a probabilistic linking approach used in the SDP research community. It uses multiple keywords together with the Bug ID and assigns a syntactic weight to all commits and a semantic confidence weight to all links. The syntactic weight is incremented for commit messages that contain Bug ID, and any of the following keywords: ”fix”, ”bug”, ”defects” and ”patch” OR no other words aside from numbers. The semantic weight is incremented for all previously found bug-commit links that satisfy any of the following conditions: The bug is resolved as fixed, a short description of the bug is contained in the commit message, the author of the commit message was assigned to the bug (we use this technique in our experiments and call it **Authorship Correspondence**), and at least one file changed in the commit is associated with the bug. The threshold for valid links was determined based on a manual inspection. Links that satisfy more than 1 semantic criterion OR that satisfy 1 semantic AND at least 1 syntactic criterion are regarded as valid. This approach is the best thus far described in the SDP research literature and is broadly used in many other studies (e.g., [19], [18], [20], [7], [28], [20].

The use of the time attribute in matching algorithm was proposed in [2]. This approach combines the use of regular expressions and the search for certain keywords with a check of the time difference between bug reporting and fix committing activities. Better performance than that of the SZZ approach was reported for the application of this method to the same Eclipse dataset, where a linking rate of 43.7% was achieved instead of 24.3%. D'Ambros et al. [10], [11] used the same pattern-matching approach as in the SZZ approach and introduced a linking verification criterion that excludes any link with a bug report date that is after the commit date. We use this technique in our experiments and call it **Time Correlation**.

A complex linking approach that combines the techniques of (1) Simple Search, (2) Authorship Correspondence, (3) Time Correlation and (4) text similarity between bug reports and change logs was implemented in the **ReLink** tool [41]. We also use this same tool in our experiments. Finally, the Fellegi-Sunter approach [39], performs matching based on nine attributes: last comment author, last comment time-stamp, 2^{nd}-to-last comment author, 2^{nd}-to-last comment time-stamp, fixed author, fixed time-stamp, product, component, and title. The performance of this algorithm and its ability to successfully recover links were demonstrated on the Apache and WikiMedia repositories. In [29] is presented a six-layer link detection approach based on textual similarity, which they called the multi-layered approach (MLink). The six layers can be described as follows: (1) a pattern-based detector similar to that of the SZZ approach, (2) a filtering layer that removes any link with committing time that do not lie between the bug opening and bug closing dates, (3) a patch-based detector that isolates code fragments within the text of a bug report and searches within the commit log, (4) a name-based detector that searches for names of entities and system components that are common to the bug report and the commit log, (5) a text-based detector similar to the ReLink tool and (6) an association-based detector that attempts to link bugs and commits that do not share any text similarity.

Our tool for automated data collection [25] offers all the aforementioned linking techniques. Furthermore, we developed another technique that is based on regular expressions so we call it the **BuCo Regex** technique. It is an improved Simple search technique which predefines the surrounding characters of a Bug ID in a commit message and it achieved very favorable results in our first analyses [27].

2.2. Datasets

Benchmarking datasets are needed to give researchers a basis for comparison. Several attempts have been made in that direction. Bachmann et al. [3] explored the missing links between bug tracking and source code management repositories. They manually collected a dataset from the Apache project, with the assistance of an expert developer, and termed it "**the ground truth**" (we also refer to this dataset in the same manner for the remainder of the paper). Unfortunately, that dataset was not made public. The same benchmarking dataset was adopted, further enlarged and used in [41] and [29]. A different approach to the creation of a benchmark dataset was taken in [8]. These authors criticized the approach of the a posteriori manual establishment of links by a person who was not the actual developer of all of the commits. Their approach instead consists of using a JIRA bug tracking system that provides the developers with an add-on to connect the bug with the commit at the time the changes to the source code are made. Using that valuable information, they constructed datasets for 10 Apache projects and made them publicly available. We used one such dataset in our comparison of the ReLink tool with more simple techniques in [27]. We also use that dataset in our present experiments and refer to it as the **Benchmark** throughout the remainder of this paper.

In related studies, efforts are rarely made to identify and properly address the bias in the existing SDP research. The studies that have addressed the existence of bias have considered its impact only through the linking rate or only on a single dataset. The community still lacks a universal approach to address this issue. One interesting approach is proposed in [24]. This approach is based on the artificial insertion of false instances into the training sample to evaluate the resistance of a particular technique to bias. Alternatively, according to [31], the bias could be effectively minimized by simply gathering larger samples. However, it may be difficult to determine and achieve the sample size needed to overcome the bias effect. In the linking research community, it is well known that the quality of a technique varies depending on the nature of the datasets and the techniques that are applied to these datasets [34]. Therefore, if we wish to produce comparable results and evaluate the performances of various linking techniques, we must perform our experiments using the same datasets. Moreover, these datasets should be freely available to other researchers and practitioners in the field. Finally, the techniques used should be repeatable for ease of comparison. For all these reasons, the objectives of this study are to investigate the repeatability of SDP linking techniques, assess their performances on common datasets and start to build a comparison network for bias evaluation.

2.3. SDP Data Collection Procedure

Besides the linking technique, there are several issues that need to be addressed in the SDP data collection procedure. Our previous research identified several issues that are inherent to the nature of many open-source repositories and that could present a significant source of bias for bug-code linking if left open to interpretation [26]:

- Bug repositories contain various **bug statuses** that are indicated as closed (resolved, verified) or merely opened (unconfirmed, confirmed, in_progress) and closed ones are relevant to SDP
- Bug repositories contain various **resolution levels** for bugs that are fixed and relevant to SDP and for bugs that could not have been fixed (invalid, wontfix, duplicate, works-for-me).
- Bug repositories usually contain various **severity levels** for bugs that cause loss of functionality (minor, normal, major, critical, blocker) and are relevant to SDP, bugs that do not cause loss of functionality (trivial) and even enhancement requests.
- There is no strict naming convention for release designations that allows for the straightforward identification of connections between bug tracking and source code management repositories.
- Large projects may be divided into several partitions within the top-level source code management repository.
- The relationship between bugs and files may be many-to-many rather than one-to-one for the following reasons:
  - A single Bug ID may be present in more than one commit
  - A single commit may contain changes to more than one file
  - A single file may be changed in commits linked to more than one bug
- Duplicated file-bug links may occur for files changed in multiple commits that are linked to one bug and they need to be counted distinctly.
- Bug IDs with fewer digits may be incorrectly identified in Bug IDs with a larger number of digits and the linking technique should comprise this fact.

There are also several issues that need to be defined to avoid bias in the SDP data collection procedure:

- The **granularity level** must be determined prior to beginning data collection, because it affects the spectrum of possible software metrics we are able to collect. The file level is the dominant choice and the file-based nature of most source code management repositories is the usual motivation [12].
- Besides the number of bugs, the files in the SDP dataset are described through a number of software metrics. There are 3 major categories of **software metrics** used in SDP research. The product metrics are the dominant choice [20], [22], [36] and the development metrics are sometimes added to them [7], [20], [22]. The deployment and usage metrics are the least used ones [22], [30].
- There are two **repository search order** approaches to the data collection. The typical order is to examine the source code repository first [43], [20], [10]. The popular **SZZ** approach also uses this order [38]. The other course of action is to first collect all bugs from bug tracking repository and then search for them in the commit messages. Schröter et al. [35] are the only authors to have compared these two approaches. They

achieved a bug identification rate of approximately 50% using the first approach and found an additional 20% using the second. Although obviously important, this seems to be an underestimated and often unaddressed issue.

3. Empirical Study

In this section, we define our empirical study strategy following the steps for constructing a verifiable, repeatable and accurate data collection plan given by Basili and Weiss [4].

3.1. Variables and Measurement Instruments

The typical evaluation measures used in linking comparisons are the linking rate and various metrics based on the confusion matrix [9], [8], [41]. We also define the Linking Rate (LR) as the ratio of linked to fixed bugs. The confusion matrix is typically used in the classification domain, and it represents all possible outcomes of correctly and incorrectly assigning instances into two classes. In our case, the instances are bug-code links, and the two classes are the presence (1) or absence (0) of links. True Positive (TP) links are those that we correctly establish. False Positive (FP) links are incorrectly established links. The links we miss are False Negatives (FNs). Missing links are a known issue in open-source community repositories [41], [8], and they are reflected in the LR. The linking techniques we use should maximize TPs, avoid FPs and minimize FNs. The category of True Negative (TN) links is more difficult to comprehend because these links represent all links that do not exist and are also not established. None of the linking techniques we have encountered make any attempt to address TN links. With the exception of the Benchmark datasets, the actual links are unknown a priori and require extensive manual investigation to discover them. For this reason, we calculate these metrics and the ones that are derived from them (precision, recall and F-measure) only for Benchmark dataset and for the first release of JDT project. Accuracy, the most standard measure, requres TN to be calculated, so it is not calculated because of afore mentioned reasons. On the other hand, we merely comment the influence of various linking techniques on their values for the remaining datasets, after we perform manual inspections. To assess the experiments and evaluate their results for the remaining datasets, instead of confusion-matrix-based measures, we define the metrics presented in Table 1. The second column of Table 1 provides brief descriptions, and the third column summarizes the purpose of each metric. Further details regarding metrics usage are provided in this section.

Several researchers have performed data collection procedures and made their data available. There are two types of datasets: those that record only the total number of bugs per file and those that reveal all underlying links. Our empirical study will use both types of datasets to evaluate our data collection procedure. The SZZ datasets (Eclipse 2.0, 2.1, 3.0) and the D'Ambros dataset (Eclipse JDT Core 3.4) give only the total number of bugs per file. Therefore, we collect the data for the same releases of the Eclipse projects to compare with the popular SZZ procedure. The Nf_i and Nb_i metrics serve to ensure that comparisons are being performed on the same input data, if possible. The comparison is then performed in terms of Nf_o and Sb_o. However, these metrics do not reveal whether the two procedures found the same bug-code links and that is why we perform additional analysis. We calculate the number of files with equal number of associated links, i.e.

Table 1. Evaluation metrics.

Metric	Description	Purpose
(Nf_i)	Number of files in the input of analysis	Comparison of input data
(Nb_i)	Number of bugs in the input of the analysis	
(Nl_o)	Number of identified links	Comparison of linking output
(Nc_o)	Number of linked commits	
(Nb_o)	Number of linked bugs	
$(LR = Nb_o/Nb_i)$	Linking rate	
(Nf_o)	Number of linked files	Comparison of final datasets
(Sb_o)	Sum of all linked bugs in all the files	
For comparing the final dataset obtained by two different linking techniques		
(Equal Nb_o)	Number of files with equal number of bugs	Finer comparison of final datasets
(Greater Nb_o)	Number of files with greater number of bugs	
(Lower Nb_o)	Number of files with lower number of bugs	
For manually comparing the output links obtained by two different linking techniques		
(Equal Nl_o)	Number of equally established links	Finer comparison of linking output
(Incorrect Nl_o)	Number of established incorrect links	
(Correct Nl_o)	Number of established correct links	
(Questionable Nl_o)	Number of established links for which the correctness is unsure	
(Precision)	Correctly established links / All established links	Finer comparison using standardized evaluation metrics
(Recall)	Correctly established links / All actual links	
(F-measure)	Harmonic mean between Precision and Recall	

number of bugs (Equal Nb_o), and for files with unequal number of associated links, we calculate the number of files for which our technique links more bugs (Greater Nb_o) and the number of files for which our technique links fewer bugs (Lower Nb_o). We distinguish these two categories because we wish to reveal which technique yields the better LR. After having found the files with equal and unequal number of bugs, we plan to manually select a number of randomly selected files with (Equal Nb_o), (Greater Nb_o) and (Lower Nb_o), analyze all the commits that are linked to these files and try to find regularities which reveal the reasons for unequal number of linked bugs.

To be able to fairly judge the appropriateness of a particular linking technique, one should obtain a benchmark dataset with all links already identified. ”The ground truth” dataset for the Apache HTTPD project was developed in [3], but unfortunately, it was not made public. However, several studies have performed comparisons against this dataset. The ReLink tool was applied to the same input by Wu et al. [41] and these authors made their Apache HTTPD[3] dataset available. Using the same input, we also indirectly compare our results with ”the ground truth.” Another approach to constructing ground truth datasets was taken by Bissyande et al. [8]. They constructed their Benchmark dataset for the Apache OpenNLP[4] project by collecting data from already-linked repositories. We use their dataset as well. Because the ReLink and Benchmark datasets provide all bug-code links, we are able to perform comparisons at a finer level of detail. The LR serves as a basis for comparing the effectiveness of various linking techniques in different contexts.

[3] https://code.google.com/p/bugcenter/wiki/ReLink
[4] http://momentum.labri.fr/bugLinking/

Table 2. Experiment objects.

Dataset		
Name	Project	Contains
SZZ	Eclipse 2.0, 2.1, 3.0	pre-release and post-release bugs per file
D'Ambros	Eclipse JDT Core 3.4	total number of bugs per class
ReLink	Apache HTTPD	list of bug-commit links established by the tool
Benchmark	Apache OpenNLP	list of bug-commit links extracted from JIRA

Tools		
Name	Input	Linking Technique
ReLink	Bugs, SVN	Simple search and NLP based prediction
BuCo Analyzer	Bugzilla GIT	Simple Bug ID Search Time Correlation Authorship Correspondence Regular Expression Bug ID Search

The Nl_o and Nc_o metrics are used to achieve more detailed comparisons of the linking techniques we examine. To reveal the intersection between two linking techniques, we calculate the number of equally established links (Equal Nl_o). For the links that the two techniques do not identify in common, we manually assess their correctness and calculate (Incorrect Nl_o) and (Correct Nl_o). For links that are not identified in common for which we cannot say for certain whether the link actually exists, we calculate (Questionable Nl_o). Since Benchmark datasets reveal which links should have been made, we also calculate the precision, recall and F-measure for ReLink and for BuCo Analyzer.

We perform a similar comparison between ReLink and BuCo Analyzer on JDT 2.0 dataset. We manually compare all the links they yielded and calculate the number of (Equal Nl_o), (Incorrect Nl_o), (Correct Nl_o), and (Questionable Nl_o). Making the assumption that the union of all links discovered by these two techniques is equal to total number of TP, we calculate their precision, recall and F-measure.

3.2. Objects

Our analyses are applied to several projects from two open-source communities: the Eclipse and Apache communities. We use these projects because considerable effort has already been directed toward empirical research of their repositories and because we need a common basis for the comparisons of the linking techniques and the data collection approaches. The datasets, the tools and the linking techniques we used are presented in Table 2.

Many studies that rely on Eclipse datasets actually reuse the datasets provided by SZZ[5] or D'Ambros[6]. The datasets provided by SZZ are the first three releases of the Eclipse project, whereas D'Ambros provided a dataset for the Eclipse JDT Core 3.4 version. This is an intermediate version, and our technique should not be evaluated solely based on its effectiveness on that version. Eclipse has had 13 releases, and it may be interesting to study the consistency of a technique's effectiveness across all of them. The

[5] https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/
[6] http://bug.inf.usi.ch

Table 3. Raw data statistics and context.

	JDT	**PDE**	**BIRT**	**HTTPD**	**OpenNLP**
Releases	13	13	9	1	1
Files	18,752	6,829	8,104	3,744	1,784
Bugs	198,206	42,582	65,173	673	100
Domain	Development Tools	Development Environment	Business Intelligence	Web Server	NLP - Language Processing

Table 4. The description of anaylsis methods used in the experiment

Experiments and Quantitative Assessment	
Goal:	assess the linking bias produced by linking techqniques and datasets
Approach:	apply the linking techniques and datasets presented in table 2
Input:	the datasets presented in table 2
Output:	the bias of each approach with respect LR and Nl_o, Nc_o, Nf_o, Nb_o
Detailed Quantitative Root Cause Analysis	
Goal:	manually identify the causes of inconsistencies between the linking results
Final Dataset Comparison	
Approach:	comparison of the Nb_o obtained by BuCo and by SZZ and D'Ambros
Input:	randomly selected 100 files and all commits linked with these files
Output:	the number of files with equal and unequal numbers of bugs
Linking Comparison	
Approach:	compare the Nc_o obtained by BuCo and by ReLink and Benchmark
Input:	all the links made by each technique
Output:	the Nl_o that are equally and unequally established links

dataset statistics and project domains are summarized in Table 3. For the Eclipse projects, this table presents the number of major releases we analyze, the total number of .java files, the total number of bugs in the input of the analysis (reported and fixed) and the total number of files in the input of analysis for all analyzed projects. Later, we remove all files that are not related to the actual project functionality. Every file that contains ".test" or ".example"[7] in its file path is omitted from the final datasets. SZZ and D'Ambros used the same approach. For the Apache projects, the table presents the total number of .c and .h files and bugs given by [41] and [8]. Files with "test" or "example" in the name and a certain number of other files for which we did not find any distinguishing features are removed from the sample.

For the purposes of systematic data collection from open-source projects for SDP, we developed the Bug-Code (BuCo) Analyzer tool that is described in details in [25] and we used it for the purpose of this study. The tool offers the possibility of selecting among the above mentioned linking techniques, including an interface for the ReLink tool.

In our study, we employ several analysis methods presented in Table 4, with the objective of developing accurate algorithms for linking techniques and reducing empirical study bias.

[7] $http://wiki.eclipse.org/Naming\_Conventions$

4. Results

In this section, we present our results and discuss the answers to the research questions stated in Section 1.

4.1. Experiments and Quantitative Assessment

The experiment and quantitative assessment are divided into three studies and are explained below.

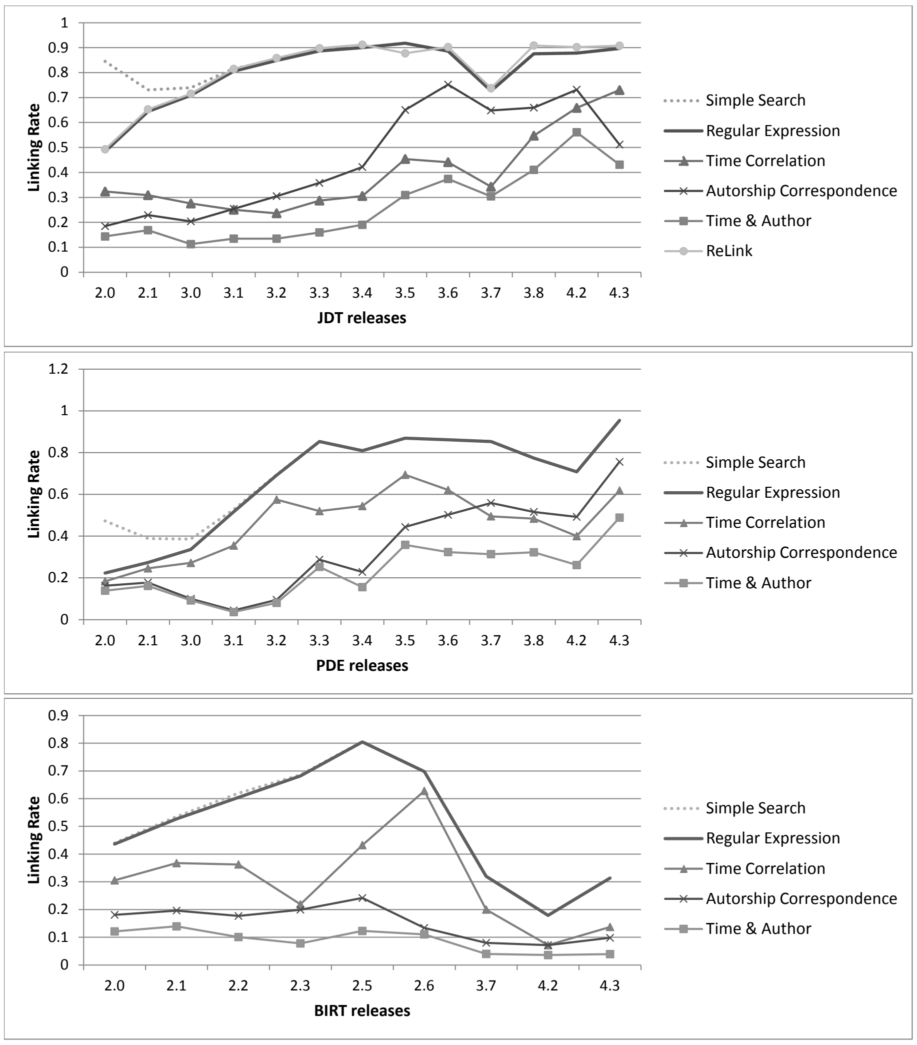

Fig. 1. Linking rate for six linking techniques applied to JDT, PDE and BIRT respectively.

Table 5. Comparison of linking techniques The first column indicates release in the analysis and the number of bugs collected from the Bugzilla repository (Nb_i). The following columns represent the linking techniques (Time stands for Time Correlation, Author stands for Authorship Correspondence, T & A stands for combination of both Time Correlation and Authorship Correspondence), the number of linked bugs (Nb_o) and the LR (Nb_o/Nb_i) each technique yields. The mean value (μ) and the standard deviation (σ) of LR for the whole project are presented for each technique.

JDT		Simple	Enhancement			BuCo	ReLink
Rls.	Nb_i	Search	Time	Autor	T & A	Regex	Tool
1	4276	84.5%	32.3%	18.5%	14.3%	48.4%	49.3%
2	1875	73.1%	30.9%	22.9%	16.9%	64.4%	65.3%
3	3385	73.9%	27.5%	20.4%	11.3%	70.9%	71.5%
4	2653	81.6%	25.0%	25.4%	13.5%	80.6%	81.5%
5	1879	85.0%	23.6%	30.5%	13.5%	84.9%	85.8%
6	1341	88.7%	28.7%	35.8%	16.0%	88.7%	89.8%
7	989	90.0%	30.5%	42.2%	19.0%	90.0%	91.5%
8	595	91.8%	45.4%	65.0%	30.9%	91.8%	92.8%
9	492	88.6%	44.1%	75.2%	37.4%	88.6%	90.2%
10	549	72.7%	34.2%	64.8%	30.4%	72.7%	73.8%
11	329	87.5%	54.7%	66.0%	41.0%	87.5%	90.9%
12	41	87.8%	65.9%	73.2%	56.1%	87.8%	90.2%
13	348	89.7%	73.0%	51.1%	43.1%	89.7%	90.8%
μ		84.2%	39.7%	45.5%	26.4%	80.5%	81.4%
σ		6.8%	16.0%	21.3%	14.4%	12.9%	12.9%

PDE		Simple	Enhancement			BuCo	**BIRT**	Simple	Enhancement			BuCo
Rls.	Nb_i	Search	Time	Autor	T & A	Regex	Nb_i	Search	Time	Autor	T & A	Regex
1	561	47.2%	18.4%	16.2%	13.9%	22.3%	2034	44.0%	30.5%	18.1%	12.1%	43.6%
2	427	38.9%	24.6%	17.8%	16.2%	27.4%	596	53.5%	36.7%	19.6%	13.9%	52.7%
3	1041	38.6%	27.2%	10.0%	9.2%	33.6%	2630	61.9%	36.3%	17.7%	10.1%	60.5%
4	769	52.8%	35.5%	4.4%	3.6%	51.6%	1761	68.5%	21.9%	19.9%	7.8%	68.2%
5	546	69.4%	57.5%	9.5%	8.1%	69.2%	807	80.5%	43.2%	24.2%	12.3%	80.4%
6	727	85.3%	52.0%	28.7%	25.3%	85.3%	172	69.8%	62.8%	13.4%	11.0%	69.8%
7	963	80.9%	54.4%	22.8%	15.6%	80.9%	25	32.0%	20.0%	8.0%	4.0%	32.0%
8	879	86.9%	69.4%	44.4%	35.8%	86.9%	28	17.9%	7.1%	7.1%	3.6%	17.9%
9	454	86.1%	62.1%	50.2%	32.4%	86.1%	51	31.4%	13.7%	9.8%	3.9%	31.4%
10	204	85.3%	49.5%	55.9%	31.4%	85.3%						
11	62	77.4%	48.4%	51.6%	32.3%	77.4%						
12	65	70.8%	40.0%	49.2%	26.2%	70.8%						
13	131	95.4%	61.8%	75.6%	48.9%	95.4%						
μ		70.4%	46.2%	33.6%	23.0%	67.1%	μ	51.1%	30.3%	15.3%	8.7%	50.7%
σ		19.6%	15.9%	22.2%	13.1%	25.0%	σ	21.1%	16.9%	6.0%	4.0%	20.9%

Comparison of Linking Techniques An examination of the results presented in Table 5 and graphically analyzed in Figure 1 yields the following observations:

OBS1: The ReLink tool links a similar but consistently higher number of bugs as BuCo Regex. The difference is low, approximately 1%, but present in all 13 examined releases of JDT. We believe that the advanced techniques implemented for the recovery of missing links are responsible for the difference. This assumption is further analyzed in the detailed root cause analysis.

OBS2: The Time Correlation and Authorship Correspondence techniques are stricter in their linking behavior (establish fewer links than other techniques). The Time Correlation and Authorship Correspondence techniques rely upon linking pieces of information from bugs and commits that are neither formally defined nor strictly guided by the community development process. They are merely recognized as potential links by researchers, and they may carry significant bias. Compared with BuCo Regex, the Time Correlation linking rate is lower by factors of 1.2-3.6 for JDT, 1.1-1.7 for PDE and 1.1-3.2 for BIRT; the Authorship Correspondence linking rate is lower by factors 1.1-3.4 for JDT, 1.2-11.6 for PDE and 2.4-5.2 for BIRT; and the combined usage of Time Correlation and Authorship Correspondence yields linking rates that are lower by factors of 1.5-6.3 for JDT, 1.6-14.1 for PDE and 3.6-8.7 for BIRT. Thus, these two techniques yield many FN links and therefore should not be used to establish links; instead, they should only be used to check questionable links.

OBS3: The BuCo Regex and Simple Search techniques link equal numbers of bugs after a few releases. Regardless of the very different numbers of bugs in certain releases, after the 5th release in the case of the PDE and BIRT projects and after the 6th release in the case of the JDT project, these two techniques link equal numbers of bugs. Unlike Simple Search, BuCo Regex defines the surrounding characters of a Bug ID and thus achieves more accurate bug linking for Bug ID numbers with fewer digits that may be incorrectly identified as a part of a Bug ID with more digits or any other number present in a commit message. When the Bug ID number reaches a sufficiently high number of digits (in the later releases of a project), these two techniques perform equally well.

OBS4: The linking rate rises for later releases. We believe that developers require a learning period to become accustomed to the community development process and that they evolve together with the project. Over time, they become more consistent in reporting bug fixing activities in their commit messages. For BuCo Regex and ReLink, the techniques that outperform the other techniques in terms of FP links, the LRs increase from below approximately 50% to above 90% for JDT, from 22% to above 80% for PDE and from 43% to 80% for BIRT. This observation leads us to conclude that later releases are less prone to developer bias, causing the lack of data quality to be less severe.

OBS5: The linking rate exhibits similar behavior for projects with similar development context. The linking rates exhibit similar behavior for the JDT and PDE projects but less similar behavior for the BIRT project. The linking rates for the later releases of the JDT and PDE projects are above 80% and reach as high as 95%. By contrast, the linking rate for the BIRT project rises above 80% for only one release and even drops below 33% in later releases. The separate development process used for this project, which is evident in the number and designations of their releases, may be the cause of this difference. This observation proves that different contexts may be present even within the same community and may influence the linking bias.

Comparison of Data Collection Contexts Table 6 presents the comparison results. The Nf_i, Nf_o and Sb_o metrics are calculated for source code files only (.c and .h for C++ and .java for the Java programming language). We ensured the comparison of the same files, as is evident from the equal values of Nf_i. The only exception is the Apache HTTPD project, for which ReLink used the SVN repository and we use the GIT repository, which lacks a certain period of source code history records. Because the SZZ dataset provides only the

Table 6. Comparison of data collection contexts: The columns represent the files in the input of analysis (Nf_i), the bugs in the input of the analysis (Nb_i), the identified links (Nl_o), the linked files (Nf_o), the linked bugs (Nb_o) and the sum of all linked bugs in all the files (Sb_o).

Input			Collection	Output			
Source	Nf_i	Nb_i	Approach	Nl_o	Nc_o	Nf_o	Sb_o
JDT 2.0	2406	unknown	SZZ	unknown	unknown	1188	3860
	2406	4276	BuCo Regex	1809	1733	1111	3794
	2406	4276	ReLink	1838	1739	1122	3924
JDT 2.1	2748	unknown	SZZ	unknown	unknown	1008	2506
	2748	1875	BuCo Regex	1085	1066	875	2142
	2748	1875	ReLink	1095	1068	817	2134
JDT 3.0	3292	unknown	SZZ	unknown	unknown	1271	3498
	3292	3385	BuCo Regex	2212	2116	1289	4423
	3292	3385	ReLink	2231	2114	1300	4465
PDE 2.0	567	unknown	SZZ	unknown	unknown	25	34
	567	561	BuCo Regex	119	112	113	246
PDE 2.1	765	unknown	SZZ	unknown	unknown	22	42
	765	427	BuCo Regex	123	122	119	229
PDE 3.0	1041	unknown	SZZ	unknown	unknown	42	77
	1041	1041	BuCo Regex	330	329	282	584
JDT Core 3.4	997	unknown	D'Ambros	unknown	unknown	206	374
	997	989	BuCo Regex	855	839	316	719
	997	989	ReLink	883	861	317	820
OpenNLP	1784	100	Benchmark	127	125	100	131
	1784	100	BuCo Regex	128	126	100	131
	1784	100	ReLink	115	113	80	102
HTTPD	2243	673	BuCo Regex	703	664	138	624
	3744	673	ReLink	1014	957	280	857

number of bugs per file, we do not know exactly which bugs are linked. Because of the different repository search order, beginning with the source code management repository, we are unable to reconstruct their input list of bugs. For this reason, we marked their values of Nb_i, Nl_o and Nc_o as unknown. The output number of files Nf_o is the number of files that contain at least one bug, and the output sum of bugs Sb_o is the total number of all bugs in all files. That number must not be misunderstood as the linking rate because one bug may be linked to more than one file.

An examination of the results presented in Table 6 yields the following observation:

OBS6: The performance of a linking technique depends on the context. The BuCo Regex technique links a far greater number of files and bugs than does the SZZ approach for the PDE project. The difference is between 77% and 85% for the PDE project. However, it produces fewer links for the first 2 JDT releases and more links for the third JDT release. These differences are not severe, ranging from 1% to 6.5% (SZZ) or 33% (D'Ambros). The ReLink tool links more files and bugs than BuCo Regex for nearly every analyzed release. The exception is the Apache OpenNLP project, where it yields approximately 20% lower values of Nf_o and Sb_o. The influence of context on the effectiveness of

Table 7. Final dataset comparison: The comparison is done between SZZ datasets and BuCo Regex for JDT and PDE, and between D'Ambros dataset and BuCo Regex for JDT Core.

Project	Total Files	**Equal** Nb_o	**Greater** Nb_o	**Lower** Nb_o
JDT 2.0	2406	1740 (72.3%)	341 (14.2%)	325 (13.5%)
JDT 2.1	2748	1383 (50.3%)	605 (22.0%)	760 (27.7%)
JDT 3.0	3292	1442 (43.8%)	970 (29.5%)	880 (26.7%)
PDE 2.0	575	482 (83.3%)	90 (15.7%)	3 (0.5%)
PDE 2.1	765	666 (87.1%)	97 (12.7%)	1 (0.1%)
PDE 3.0	874	621 (71.1%)	253 (28.9%)	0 (0.0%)
JDT Core 3.4	997	621 (62.3%)	253 (25.3%)	123 (12.3%)

a linking technique is visible in three layers: the open-source community (evident in the links missed by ReLink only for Apache HTTPD), the project (evident from the opposite results achieved by SZZ and BuCo Regex for JDT and PDE) and the release (evident from the opposite results achieved by SZZ and BuCo Regex within the JDT project). Therefore, we perform further analyses to elucidate the issues that may give rise to these differences.

4.2. Detailed Qualitative Root Cause Analysis

The detailed qualitative root cause analysis begins with the search for intersections among different linking techniques. When comparing with the SZZ approach, we do not have access to their intermediate linking results; we know only the final number of links per file. This comparison is referred to as a *Final dataset comparison*, and it is based on Nf_o, Sb_o, Equal Nb_o, Greater Nb_o and Lower Nb_o, as defined in previous section. When comparing with ReLink and Benchmark, we have access to all linking results; this comparison is termed a *Linking comparison*, and it is based on Nf_o, Sb_o, and Nl_o. However, the sole comparison of these numbers cannot reveal the causes of any inconsistencies. Therefore, a manual investigation of a subset of consistent and inconsistent results is performed by the main author of this paper. To reveal the cause of inconsistencies and check the correctness of the intersecting results, all links are manually investigated. For *Final dataset comparison*, we search for root causes of inconsistencies and calculate the number of files affected by each of them. For *Linking comparison*, we search for correctness of established links and calculate **Equal** Nl_o, **Correct** Nl_o, **Incorrect** Nl_o and **Questionable** Nl_o, as defined in previous section.

Final Dataset Comparison The first analysis consists of identifying the files with equal and unequal numbers of bugs between the BuCo and SZZ approaches and between the BuCo and D'Ambros approaches. The results are presented in Table 7. The rows contain the following pieces of information: the release of the analyzed projects, the total number of files in the output (Nf_o), the number of files with equal numbers of bugs (**Equal** Nb_o), the number of files for which BuCo linked more bugs (**Greater** Nb_o) and the number of files for which BuCo linked fewer bugs (**Lower** Nb_o). For the files with equal and unequal numbers of bugs, we also provide the percentage as a ratio with respect to the total number of files.

The root cause analysis of the inconsistencies presented in Table 7 continues with a manual investigation. We randomly select a subset of 100 files from the datasets reported by SZZ and D'Ambros, search the source code history for all changes made to these files and compare the commits with those linked by BuCo. This subset of files consists of an equal 50:50 ratio of files with equal numbers of bugs and unequal numbers bugs and a 60:40 ratio of files from the JDT and PDE projects. We find evidence of 3 discrepancies in the data collection techniques that caused different Nb_o values per file to be obtained, which are presented in Table 8. After the project name in the first column, the following columns present the number of files analyzed (Files Analyzed), the number of files with no difference in the numbers of bugs linked and for which we discovered no inconsistencies (Equal Nb_o) and the number of files for which different numbers of bugs were linked because of 3 types of differences: **DIFF1** ($\Delta t > 6$ months), **DIFF2** (Key word missing) and **DIFF3** (Other project). Detailed descriptions of these differences are provided below.

DIFF1: ($\Delta t > 6$ months) The SZZ data collection approach includes only bugs that are reported more than 6 months before or after the release date of the release under examination. This difference manifests in several ways. For example, the Bug ID 6839 was reported in December 2001, more than 6 months away from the release date of JDT 2.0. Our approach linked this bug to a commit with the message "Fix for 6839" from January 2002, two days after the bug had been closed. The SZZ approach decided not to establish that link and therefore found one fewer bug in one file. A more severe difference is observed for the Bug IDs 28622, 28559 and 28682. These bugs were all reported in December 2002, less than 6 months away from the release date of JDT 2.0. However, these bugs were reported for the JDT 2.1 release instead. The commits with messages "Fixes for 28559, 28622 and 28682" and "Fix for 28682" are linked to these bugs and give rise to a twofold difference: for release 2.0, 3 more files are linked and the sum of bugs is higher by 7 bugs, whereas for release 2.1, 3 fewer files are linked and the sum of bugs is lower by 7 bugs.

DIFF2: (Key word missing) The SZZ data collection approach includes links only to commits that contain keywords, such as "bug" or "fix", or a character "#" in front of a Bug ID within the commit message [43]. The ClassFileStruct.java file[8] provides an example for which this difference in approach is relevant; for this file, the SZZ method found 0 bugs, and our method identified 2 pre-release bugs. Both of these bugs were found in the commit message "Update for 10979 and 10697." There are two pieces of evidence in favor of linking this commit: the time elapsed between the closing of the bug and the committing of the change is less than 1 day, and the bug assignee for Bug ID 10679, Olivier Thomannis, is the author of this commit. Because of the lack of keywords, such as "bug," or "fix", or the character "#," the SZZ technique also decided not to link the Bug ID 10697 with the commit message "Update for 10697." The fact that these two commits are ignored means that the number of files linked to bugs by the SZZ approach is lower by 7 files, and the sum of bugs is lower by 13 bugs.

DIFF3: (Other project) The SZZ data collection approach occasionally includes bugs that are reported for other Eclipse projects within the Bugzilla repository that are different from the project under examination. The ASTNode.java file[9]provides an example of this difference. For this file, the SZZ approach found 9 pre-release and 1 post-release

[8] org/eclipse/jdt/internal/compiler/classfmt/ClassFileStruct.java
[9] org/eclipse/jdt/core/dom/ASTNode.java

Table 8. Root cause analysis of final dataset comparison: The first column presents the source of the subset, the second column gives the number of files in the subset (50% of equal number of bugs and 50% of unequal number of bugs) and the remaining columns give the amount of files affected by each difference.

Project	Files	**Δ t $>$ 6 months**	**Key word missing**	**Other project**
JDT	60	16 (26.6%)	9 (15.0%)	3 (5.0%)
PDE	40	13 (32.5%)	8 (20.0%)	1 (2.5%)
Total	100	29 (29.0%)	17 (17.0%)	4 (4.0%)

bugs, whereas our approach identified 8 pre-release and 1 post-release bugs. Analyzing the entire GIT repository for changes made to that file reveals only two additional potential links. These are commits with the following messages: ”Fix for 10495” and ”Add useful toString methods for ASTNodes (bug 11076).” Because the Bug ID 10495 is of trivial severity and is thus not included in the data collection process by either technique, the inclusion of the Bug ID 11076 makes the difference; however, this is a bug reported for the Platform project, not the JDT project. Such differences result in an increase in the sum of bugs and the number of files linked to bugs identified by the SZZ approach. The commit presented here causes 6 files to be linked to bugs.

An examination of the results presented in Tables 7 and 8 yields the following observations:

OBS7: Linking effectiveness may be influenced by overly strictly defined linking procedure. One of recommendations made by Basilli and Weis is to make the data collection forms should provide certain degree of flexibility. The strict limiting factors of the 6-month time frame and the insistence on the presence of keywords are the predominant reasons for the different linking results. The 6-month period limitation imposed by SZZ and D'Ambros is the major cause of bias between BuCo and their data collection approach. This limitation impacts 29 of the analyzed files and is therefore responsible for 58% of the files with different numbers of bugs (Nb_o). The lack of certain keywords in the commit message is a secondary cause of bias, impacting 17 of the analyzed files and responsible for 34% of the files with different Nb_o values. The least significant source of bias is the fact that their approach also included bugs associated with other projects. However, this source of bias is not entirely insignificant because it impacts 4 of the analyzed files and is therefore responsible for 8% of the files with different Nb_o values. All of these sources of bias influence the linking rate: in certain cases, they lead to the omission of obvious links, whereas in other cases, they cause the creation of links to bugs from different releases or different projects.

OBS8: The linking techniques do not yield consistently different outputs across different contexts . This observation is to OBS6 but confirmed in a different metric, this one related to the number of bugs per file. The intersection of the results is greater than 70% for all PDE releases and drops to below 45% for the JDT project. The trend of the change in the level of intersection is different for the two projects as well. The level of intersection drops from 72.3% to 43.9% for later JDT releases, whereas for the PDE project, it initially rises from 83.8% to 87.1% and then drops to 71.1%. We also observe various trends in the categories of files with unequal numbers of bugs. The number of files with lower Nb_o rises from 13.5% to 27.7% and then drops slightly to 26.7% over time

Table 9. Linking comparison: The total number of links Nl_o and the quantity of links in each of the manually validated categories is presented for each technique.

Project	JDT 2.0				OpenNLP			
Linking approach	Relink		BuCo Regex		Relink		BuCo Regex	
Nl_o	2979		2830		115		128	
Equal Nl_o	2793	93.8%	2793	98.7%	115	90.6%	127	100.0%
Incorrect Nl_o	61	2.0%	3	0.1%	0	0.0%	1	0.8%
Correct Nl_o	12	0.4%	34	1.2%	0	0.0%	11	8.6%
Questionable Nl_o	114	3.8%	0	0.0%	0	0.0%	0	0.0%
Precision	94.13% - 97.87%		99.89%		100%		99.22%	
Recall	98.80%		99.58%		90.55%		100%	
F-measure	96.44% - 98.34%		99.74%		95.16%		99,61%	

for the JDT releases. By contrast, it remains approximately 0% for all releases of PDE. The number of files with greater Nb_o is constantly rising from 14.2% to 29.5% over time for the JDT project, and for the PDE releases, it initially drops from 15.7% to 12.7% and then rises to 28.9%. It is clearly impossible to observe any trends from the sole release analyzed by D'Ambros. However, for the 3 subsequent JDT releases analyzed by SZZ, both categories of files with unequal numbers of bugs change at similar rates. For the 3 PDE releases, we observe no change in the category of **Lower** Nb_o, but we observe a sudden jump in the category of **Greater** Nb_o after the second release. Therefore, again we observe the influence of context in two layers: the project and different releases of the same project.

Linking Comparison The second analysis consists of determining the numbers of equally and unequally linked commits between BuCo and ReLink for the JDT 2.0 project and among BuCo, ReLink and Benchmark for the OpenNLP project. For unequally linked commits, we distinguish the number of correctly and incorrectly established links. The correctness of the links is evaluated through the following parameters: the presence of the Bug ID in the commit message, the authorship correspondence between the bug assignee and the commit author and the time correlation between bug closing and commit submission. Because the Bug ID is the parameter of dominant relevance, we also label commits without any Bug ID in the commit message as questionable links. Such commits are subsequently evaluated through the remaining two parameters. The results are presented in Table 9. The rows present the following pieces of information: the release of the analyzed projects, the linking approach, the total number of links (Nl_o), the number of commits equally linked by ReLink and BuCo Regex (**Equal** Nl_o), the number of incorrect links (**Incorrect** Nl_o), the number of correct links (**Correct** Nl_o) and the number of questionable links (**Questionable** Nl_o). For each commit category, we also provide the percentage with respect to the total number of commits (Nl_o). We also calculate precision, recall and F-measure as it was explained in Section 3.1 and in Table 1. The precision for ReLink in JDT 2.0 project has a range of values, from the worst case scenario in which all the Questionable Nl_o are considered as FP, to the best case where they all are marked as TP.

An examination of the results presented in Table 7 and a further manual investigation of all links yield the following observation:

OBS9: The use of prediction based linking technique may be outperformed by a simpler technique based on regular expression. Although over 93% of the links are established by both tools, the specific behaviors of these techniques can lead to the establishment of some incorrect links or the omission of some obvious links. It remains unclear why the ReLink tool failed to find 45 (1.6%) of the links correctly identified by the BuCo tool. Among the 12 (0.4%) correct links missed by our method, 8 were duplicated entries that represented changes in the same Java class files, 3 were duplicated entries that were stored in different repositories and 1 was a correctly predicted typographical error. The commit message ”Fixed bug 207141: Inexact Matches dialog check box isn't properly positioned [search]” was linked to the Bug ID 20741 by ReLink. The Bug ID 207141 indicated in the commit message was a bug from the BIRT project reported in 2007, and the commit itself occurred in 2002 on the same day that Bug ID 20741 was closed. However, the ReLink linking technique that allows for Bug IDs with typographical errors in commit messages typically creates obviously incorrect links. There are 61 (2%) such incorrect links for which the difference between the Bug ID and the number in the commit message involves more than 1 digit and the number in the commit message is equal to another, already linked bug. For example, the commit ”bug 9456” was already correctly linked to the Bug ID 9456, but ReLink also linked it to the Bug ID 9591, which was still unlinked. The reason for the incorrect links created by BuCo was always a low Bug ID number. All 3 of the incorrect links created by our method included the Bug ID as part of a Java class name. These bug identifiers had only 4 digits, and our BuCo Regex technique is prone to making incorrect links for Bug IDs that are such low numbers. Such behavior of ReLink and BuCo Regex techniques resulted in better precision, recall and F-measure obtained by BuCo. The only exception is the recall for OpenNLP, for which ReLink made no FP links and BuCo made 1 FP link.

OBS10: The linking rate may lead to misleading conclusions regarding the linking effectiveness. The higher linking rate and the higher Nf_o and Sb_o values achieved by ReLink, which we note in **OBS4**, do not necessarily indicate that the technique is more effective than the BuCo Regex technique. Manual investigation reveals that 99.2% of the links created by BuCo were correct, whereas 94.2% of those created by ReLink were correct. Moreover, BuCo missed only 0.4% of the links that were identified by ReLink, and aside from duplicated entries of various types, there was only 1 such link that BuCo could not have established. By contrast, ReLink missed 1.2% of the correct links. Among the 114 questionable links created by ReLink, 76 were committed within 10 days of the bug closing date, 25 satisfied the authorship correspondence condition and only 18 satisfied both of these conditions. Obviously, any comparison of linking technique performance in which knowledge of the true links is not available (as it is for Benchmark) must be examined more thoroughly than a mere calculation of the linking rate. It may be overly difficult to identify all missed links (FNs in the confusion matrix), but the ratio between the numbers of TP and FP links may provide more conclusive evidence.

4.3. Answers to Research Questions

Our case study resulted in 10 valuable observations that enable us to answer the two RQs stated in Section 1.

RQ1: How strongly is linking effectiveness influenced by development context and why? Our results indicated that context does influence the bug linking effectiveness. We presented several observations that support this claim: **OBS4**, **OBS5**, **OBS6** and **OBS8**. The development context may be observed in terms of linking rate obtained by two linking techniques within the same community and within the same project. The linking rate trend may vary within the same community, as we observed the increasing trend for JDT and PDE projects that never decreases below 70%, while for BIRT it decreases below 20% after having increased up to 80%. The linking rate may significantly vary within the same project, as we observed for JDT (49% - 93%), PDE (22% - 95%) and BIRT (18% - 80%).

RQ2: Which linking technique identifies the highest number of correct links? Our results indicated that the prediction based linking technique and overly strict linking techniques and procedures may influence the bug linking effectiveness negatively. Finally, we came to conclusion that the BuCo Regex linking techniques demonstrated the best performance. We presented several observations that support this claim: **OBS2**, **OBS3**, **OBS7**, **OBS9** and **OBS10**. The Time Correlation and Authorship Correspondence were the overly strict linking techniques that yielded significantly lower LR than BuCo Regex and ReLink. The SZZ approach also used a strict linking technique which made mandatory for a commit to contain a key word and the commit must have been made within the six months period from the date of the release. In comparison with BuCo Regex technique, these facts lead to increasing discrepancy between the Nb_o per file. It ranges from 13% for PDE 2.0 up to 56% for JDT 3.0. The Relink tool was the prediction based linking technique that yielded worse results than the simpler BuCo Regex technique. The F-measure was lower by 3% for JDT 2.0 and by 4.5% for OpneNLP.

4.4. Threats to Validity

The range of open-source communities and the range of projects we analyzed may be regarded as a threat to the validity of our analysis. Nevertheless, we used both small and large projects from the Eclipse and Apache open-source communities, which are commonly addressed in related research. We also obtained conclusive evidence that offered answers to our research questions. The projects of the Eclipse community are the most often investigated in this type of analysis because of their large size, long-lasting evolution period and ready availability of data. Our choice of potential projects was also narrowed by the public availability of data related to SDP research and by our experimental intention to compare previous results with our own.

The problem of untraceable bugs is one of the major threats to the validity of this research. Bugs may be untraceable if they are not mentioned in the commit messages that accompany the bug fixes. The impact of this issue was confirmed in our results through the linking rates presented in Table 5. However, bugs may also be untraceable if they are never noted in the bug tracking repository. The impact of this issue is impossible to assess without access to all activities of the developers and without their personal assistance. Some natural language processing techniques have been used to cope with the issue of untraceable bugs but as we observed, with limited and questionable success that came at the cost of creating incorrect bug-code links.

5. Conclusion

The quality of the data subjected to data collection and analysis is related to the performance of the applied linking technique. The performance of a linking technique depends on the nature of the technique as well as the context to which it is applied. Therefore, if we wish to produce comparable results and evaluate the performances of various linking techniques, we should perform our experiments using the same datasets. If we wish to make progress in evaluating the quality of data available to the SDP research community, then the datasets on which we conduct comparisons should be made freely available to other researchers and practitioners in the field. Thus, we will be able to build a common benchmark base. Because the performance of a technique is highly dependent on the nature of the dataset to which it is applied, it is of vital importance to continue to build a common evaluation network, as proposed in this article, and to begin to identify context information that reflects the data bias. As a result of this exhaustive research, we provided detailed guidelines for researchers for the evaluation of data collection bias in their studies. To make research in the SDP field more generalizable, researchers must follow a standardized procedure when collecting bugs to minimize bias. Our research identified several issues that must be considered and addressed: granularity level, software metrics, bug severity, repository search order, and duplicated bug-file links. Some of these issues have not been previously reported in related research papers. Nevertheless, they are often encountered while performing even exploratory research. It may be dangerous to believe that these issues are irrelevant or harmless, as some of them have proven to significantly affect dataset construction and analysis.

It is important to provide all of the details of data collection procedures because the context in which they are applied may also influence their effectiveness. We proved that data quality may vary between different research communities, within the same research community, among projects that share a common development process and even throughout the evolution of a single project. Should the results of data collection indicate a low level of data quality, the results of any research based on such data may be incorrect or questionable at best. Further research should analyze whether it may be possible to establish a quality threshold to define acceptable data quality.

For the purpose of comparing and validating bug linking techniques and procedures, we showed how important it is to use the Benchmark datasets and to report all the intermediate results that were performed during the data collection procedure. In our case, the most important intermediate result are the bug commit links. Otherwise, the validation has to be done manually, which is time consuming and very difficult to perform on larger projects. However, our empirical study demonstrates which measures and comparison procedures can be used to perform such a manual comparison. The goal of such procedures should not be to quantify the inconsistencies, but to discover the root causes that lead to inconsistencies. We believe this to be the proper way to further develop this research area.

We also developed the BuCo Analyzer tool with the goal of supporting researchers in obtaining equivalent datasets regardless of the open-source project they choose to analyze[10]. We demonstrated its effectiveness in collecting and analyzing data from all projects used in the establishment of the SZZ, D'Ambros, ReLink and Benchmark datasets.

[10] We encourage all researchers interested in performing such an analysis to contact us.

This tool is applicable to any project that contains the specified input data or for which both Bugzilla and GIT repositories are available. It is capable of analyzing both large-scale projects, such as JDT, and smaller ones, such as OpenNLP. It also offers a bug linking technique based on regular expressions, for which we demonstrated performance superior to that of other more complex techniques, such as the approach used by ReLink. Furthermore, this tool incorporates our bug linking approach, which offers several advantages compared with the popular SZZ approach. Our future efforts will include making this tool available to the entire research community.

Acknowledgements. This work has been supported in part by Croatian Science Foundation's funding of the project UIP-2014-09-7945 and by the University of Rijeka research grant Grant 13.09.2.2.-16.

References

1. Ali, N., Guhneuc, Y.G., Antoniol, G.: Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. IEEE Trans. Software Eng. 39(5), 725–741 (2013)
2. Bachmann, A., Bernstein, A.: Data retrieval, processing and linking for software process data analysis. Tech. Rep. IFI-2009.0003b (2009)
3. Bachmann, A., Bird, C., Rahman, F., Devanbu, P., Bernstein, A.: The missing links: Bugs and bug-fix commits. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 97–106. FSE '10, ACM, New York, NY, USA (2010)
4. Basili, V.R., Weiss, D.: A methodology for collecting valid software engineering data. IEEE Computer Society Trans. Software Engineering 10(6), 728–738 (1984)
5. Bass, J., Garfield, C.: Statistical linkage keys: How effective are they? In: Proceedings of the 12th Symposium on Health Data Linkage. pp. 40–45. Public Health Information Development Unit, Syndey, NSW, Australia (2002)
6. Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., Devanbu, P.: Fair and balanced?: Bias in bug-fix datasets. In: Proceedings of ESEC/FSE '09. pp. 121–130. ACM, New York, NY, USA (2009)
7. Bird, C., Nagappan, N., Gall, H., Murphy, B., Devanbu, P.T.: Putting it all together: Using socio-technical networks to predict failures. In: ISSRE. pp. 109–119. IEEE Computer Society (2009)
8. Bissyande, T.F., Thung, F., Wang, S., Lo, D., Jiang, L., Reveillere, L.: Empirical evaluation of bug linking. In: Proceedings of CSMR '13. pp. 89–98. IEEE Computer Society, Washington, DC, USA (2013)
9. Christen, P., Goiser, K.: Quality and complexity measures for data linkage and deduplication. In: Guillet, F., Hamilton, H.J. (eds.) Quality Measures in Data Mining, Studies in Computational Intelligence, vol. 43, pp. 127–151. Springer (2007)
10. D'Ambros, M., Lanza, M., Robbes, R.: On the relationship between change coupling and software defects. In: Antoniol, G., Pinzger, M., Chikofsky, E.J. (eds.) WCRE. pp. 135–144. IEEE Computer Society (2009)
11. D'Ambros, M., Lanza, M., Robbes, R.: An extensive comparison of bug prediction approaches. In: MSR. pp. 31–41 (2010)
12. D'Ambros, M., Lanza, M., Robbes, R.: Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empirical Softw. Engg. 17(4-5), 531–577 (Aug 2012)
13. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of the American Statistical Association 64, 1183–1210 (1969)

14. Fischer, M., Pinzger, M., Gall, H.: Analyzing and relating bug report data for feature tracking. In: Proceedings of the 10th Working Conference on Reverse Engineering. pp. 90–. WCRE '03, IEEE Computer Society, Washington, DC, USA (2003)
15. Gill, L. (ed.): Methods for automatic record matching and linking and their use in national statistics. Oxford University (2001)
16. González-Barahona, J.M., Robles, G.: On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Empirical Softw. Engg. 17(1-2), 75–89 (Feb 2012)
17. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. 38(6), 1276–1304 (Nov 2012)
18. Hu, W., Wong, K.: Using citation influence to predict software defects. In: Proceedings of the 10th Working Conference on Mining Software Repositories. pp. 419–428. MSR '13, IEEE Press, Piscataway, NJ, USA (2013)
19. Jiang, T., Tan, L., Kim, S.: Personalized defect prediction. In: ASE. pp. 279–289. IEEE (2013)
20. Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.i., Adams, B., Hassan, A.E.: Revisiting common bug prediction findings using effort-aware models. In: Proceedings of the 2010 IEEE International Conference on Software Maintenance. pp. 1–10. ICSM '10, IEEE Computer Society, Washington, DC, USA (2010)
21. Kelman, C.W., Bass, J., Holman, D.: Research use of linked health dataa best practice protocol. Australian NZ J. Public Health 26, 251–255 (2002)
22. Khoshgoftaar, T.M., Yuan, X., Allen, E.B., Jones, W.D., Hudepohl, J.P.: Uncertain classification of fault-prone software modules. Empirical Software Engineering 7(4), 295–295 (2002)
23. Kim, D., Tao, Y., Kim, S., Zeller, A.: Where should we fix this bug? a two-phase recommendation model. IEEE Trans. Softw. Eng. 39(11), 1597–1610 (Nov 2013)
24. Kim, S., Zhang, H., Wu, R., Gong, L.: Dealing with noise in defect prediction. In: Proceedings of the 33rd International Conference on Software Engineering. pp. 481–490. ICSE '11, ACM, New York, NY, USA (2011)
25. Mauša, G., Galinac Grbac, T., Dalbelo Bašić, B.: Software defect prediction with bug-code analyzer - a data collection tool demo. In: Proceedings of SoftCOM '14. Split, Croatia (2014)
26. Mauša, G., Galinac Grbac, T., Dalbelo Bašić, B.: Data collection for software defect prediction an exploratory case study of open source software projects. In: Proceedings of MIPRO '14. pp. 513–519. Opatija, Croatia (2015)
27. Mauša, G., Perković, P., Galinac Grbac, T., Štajduhar, I.: Techniques for bug-code linking. In: Proceedings of SQAMIA '14. Lovran, Croatia (2014)
28. Mizuno, O., Ikami, S., Nakaichi, S., Kikuno, T.: Spam filter based approach for finding fault-prone software modules. In: MSR. p. 4. IEEE Computer Society (2007)
29. Nguyen, A.T., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N.: Multi-layered approach for recovering links between bug reports and fixes. In: Proceedings of FSE '12. pp. 63:1–63:11. ACM, New York, NY, USA (2012)
30. Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Where the bugs are. SIGSOFT Softw. Eng. Notes 29(4), 86–96 (Jul 2004)
31. Rahman, F., Posnett, D., Herraiz, I., Devanbu, P.: Sample size vs. bias in defect prediction. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. pp. 147–157. ESEC/FSE 2013, ACM, New York, NY, USA (2013)
32. Robillard, M., Walker, R., Zimmermann, T.: Recommendation systems for software engineering. IEEE Software 27(4), 80–86 (2010)
33. Rodriguez, D., Herraiz, I., Harrison, R.: On software engineering repositories and their open problems. In: Proceedings of RAISE '12. pp. 52–56 (2012)
34. Salzberg, S.L., Agrawal, A.: On comparing classifiers: Pitfalls to avoid and a recommended approach. Data Mining and Knowledge Discovery 3(1), 317–327 (Feb 1997)

35. Schröter, A., Zimmermann, T., Zeller, A.: Predicting component failures at design time. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering. pp. 18–27. ISESE '06, ACM, New York, NY, USA (2006)
36. Shatnawi, R., 0014, W.L.: The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. Journal of Systems and Software 81(11), 1868–1882 (2008)
37. Shepperd, M.J., Song, Q., Sun, Z., Mair, C.: Data quality: Some comments on the nasa software defect datasets. IEEE Trans. Software Eng. 39(9), 1208–1215 (2013)
38. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? SIGSOFT Softw. Eng. Notes 30(4), 1–5 (May 2005), http://doi.acm.org/10.1145/1082983.1083147
39. Sureka, A., Lal, S., Agarwal, L.: Applying fellegi-sunter (fs) model for traceability link recovery between bug databases and version archives. In: Proceedings of APSEC '11. pp. 146–153. IEEE Computer Society, Washington, DC, USA (2011)
40. Thomas, S.W., Nagappan, M., Blostein, D., Hassan, A.E.: The impact of classifier configuration and classifier combination on bug localization. IEEE Trans. Software Eng. 39(10), 1427–1443 (2013)
41. Wu, R., Zhang, H., Kim, S., Cheung, S.C.: Relink: Recovering links between bugs and changes. In: Proceedings of ESEC/FSE '11. pp. 15–25. ACM, New York, NY, USA (2011)
42. ZHANG Jie, WANG XiaoYin, H.D.X.B.Z.L.M.H.: A survey on bug-report analysis. SCIENCE CHINA Information Sciences 58(2), 21101 (2015)
43. Zimmermann, T., Premraj, R., Zeller, A.: Predicting defects for eclipse. In: Proceedings of the Third International Workshop on Predictor Models in Software Engineering. pp. 9–. PROMISE '07, IEEE Computer Society, Washington, DC, USA (2007)

Goran Mauša received his Masters degree in Electrical Engineering at the Faculty of Engineering, University of Rijeka, Croatia in 2010. He is a PhD student in Computing at the Faculty of Engineering and Computing, University of Zagreb, Croatia since 2011 and a junior researcher at the Software Engineering and Information Processing Laboratory at the Faculty of Engineering, Rijeka. His research interests are soft computing and their usage in software defect prediction of complex software systems.

Tihana Galinac Grbac received MS and PhD degree from the University of Zagreb, Croatia in 2005 and 2009, respectively. She is an assistant professor at the Faculty of Engineering, University of Rijeka, Croatia, and the leader and founder of the Software Engineering and Information Processing Laboratory (SEIP Lab) at the same institution. Until 2007 she was with Ericsson Nikola Tesla for eight years. Her research interests are in modeling complex software systems behaviour and their evolution, and autonomous control mechanisms for their reliable and secure operation.

Bojana Dalbelo Bašić is currently full professor at University of Zagreb, Faculty of Electrical Engineering and Computing. She received BS degree in Mathematics at the Faculty of Natural Sciences and Mathematics, University of Zagreb in 1982. She received MS and PhD degree at the Faculty of Electrical Engineering and Computing, University of Zagreb in 1993 and 1997, respectively. She worked as a programmer and project manager in Croatian companies. Since 2000 she works at the Faculty of Electrical Engineering and Computing, University of Zagreb. She has been the leader of several international and national professional and scientific research projects. She is the author of over 90 professional and scientific papers. Her research focuses on using machine learning and statistical methods in knowledge discovery and natural language processing.

Received: December 28, 2014; Accepted: November 10, 2015.