Article

Improving the Quality of Bug Data in Open Source Software Repositories

Bilyaminu Romo Auwal * and Andrea Capiluppi

Posted Date: 23 July 2025

doi: 10.20944/preprints202507.1904.v1

Keywords: Bug traceability; Bug-fixing commits; SZZ algorithm; Development logs; Bug data

Article

Improving the Quality of Bug Data in Open Source Software Repositories

Bilyaminu Auwal Romo [1] **and Andrea Capilupp** [2,*]

[1] Department of Computing and Engineering, University of East London, UK
[2] Faculty of Science and Engineering, Software Engineering, Universitair hoofddocent
* Correspondence: a.capiluppi@rug.nl

Abstract

Context: Researchers have increasingly recognised the benefit of mining software repositories to extract information. Although, the development logs of software projects, contained in Version Control (VC) systems can be severely incomplete when tracking bugs, especially in open-source software projects, resulting in reduced traceability of defects. Other times, such logs can contain bug information that is not available in the bug tracking system (BT system) repositories, and vice-versa. If the development logs and bug related-data (i.e., BT system data) were applied together, researchers and practitioners often would have a larger set of bug IDs for a software project, and a better picture of a bug life cycle, its evolution and maintenance. **Method:** The research aims to design and implement a toolchain to support the integration of a VC and BT toolset as well to synchronise the missing development logs and Bug data of open-source software projects automatically. Considering a random sample of 344 Open Source Software (OSS) projects development logs (55,689) and Bugs data (167,800), the two objectives of this paper are (i) to determine which of the keywords 'Fix', 'Bug' or the '#' identifier provide better precision; and (ii) to analyse their respective precision and recall at locating significant amount possible of bug IDs semi-automatically. In its formulation, the SZZ algorithm looks for the terms "Bugs" or "Fixed" (case-insensitive) along with the '#' sign, that shows the ID of a bug in the development logs and Bug data respectively. **Results:** Overall, our results suggest that the use of the '#' identifier in conjunction with the bug ID digits (e.g., #1234) is more precise for locating bugs in the development logs than the use of the 'Bug' and 'Fix' keywords. Such keywords are indeed present in the development logs, but they are less useful when trying to connect the development actions with the bug traces in open source software project. The results indicate that the development log and Bug related data can be track and recovered with better accuracy using only a part of the SZZ algorithm. Thus 80-95% of all The missing bug data and development logs of 344 OSS projects have been synchronised using the proposed tool-chain into Bicho and CVSAnalY database respectively. **Conclusion:** The presented toolchain eliminate and avoid repetitive activities in traceability tasks, software maintenance and evolution. The fact that in the past researchers have proven linking and synchronising development logs and bug data is complicated. Thus Bicho and CVSAnalY tools were developed to mine and store development logs and bug data independently. This research provides a solution towards the automation and traceability of bug data of software projects (in particular, OSS projects) using development logs to complement and track the missing bug data. The Synchronisation involves completing the missing bug data in software repositories with the development logs which details the actions of developers.

Keywords: bug traceability; bug-fixing commits; SZZ algorithm; development logs; bug data

1. Introduction

This work is focused on the relevance of development logs as referrals of bug information. In an ideal world, a bug being 'opened' in a BT system should log its status onto the respective development log too. The eventual fix should be detailed in the VC system, and by the steps made by the developers in its solving. Finally, its status should be changed to ’Fixed’ on the BT system, once the development logs confirmed the correct working of the snippet of code that caused the issue in the first place. The traceability of bugs would require to collect the same number of bug IDs from both development logs in the VC system and BT systems. However, this does not seem to be the case and this forms the motivation for this study.

Traceability links are required to carry out various software evolutionary activities such as designing and building defect prediction models [21]. Conversely, the tools that are required to retrieve development logs lacks integration [7] with the bug tracking systems: as a result, two different independent sets of bug-related data are produced, and filling different databases [15]. Researchers suggested that using bug IDs in the development logs could help to identify and recover missing traceability links [2,10,12]. Inevitably these logs might be manually or semi-automatically analyse to determine if the development logs from the VC system and bug IDs from the BT system are referring to disjoint sets (i.e., the same set of bug IDs in VC system and BT systems).

So far, the locating development logs and bug related data in open source software community have been carried-out mostly in two ways: (i) by using keywords such as ’Fixed’ or ’Bug’ [13]; and (ii) by searching for references to bug reports using the ’#’ sign and numeric values (e.g., #43210), which are linked to the ID of a bug report in the BT system and VC system respectively [5,7,18]. The SZZ algorithm is an exemplary technique that combines such keywords and proxies to locate Bug IDs and Development logs in Open Source Software (OSS) projects [18].

In this research, 344 OSS projects from GitHub were analysed to pilot and demonstrate an approach to recover the union of sets of bug IDs, from their VC system and BT system. Therefore, the main objective of this paper is, practically, to locate as many bug IDs as possible, using both the development logs and bug related-data. The approach is based on two basic steps, for each analysed project: first, to implement the SZZ algorithm, and use its basic components, to analyse their respective precision and recall in isolating bug IDs from the development logs. Second, to use the bug related-data as a baseline, to detect how many of its bug IDs are effectively found in the development logs, if any.

It is imperative to analyse each components in locating bug IDs in OSS development logs and evaluate the most precise components. The main challenge faced by researchers in locating the development logs and bug related-data in OSS project is deciding the right keywords. A typical example is the study of Casalnuovo et al. [4] where the researchers have applied nine (9) keywords to locate development logs and bug related-data such as: ’bug’, ’issue’, ’fix’, ’error’, ’defect’, ’flow’, ’fault’, ’mistake’ and ’incorrect’. Previous studies have used mainly two keywords that is ’bug’ and ’fix’ to locate bugs in the development logs. Although, one can argue that the convention used by software developers in OSS community in detailing their actions in the source code or VC system may differ from one project to another especially in OSS projects. Thus to foster and re-enforce the number of components researchers might deliberate in code manipulation and analysis, it is vital to measure each component or keywords precision and recall in locating bug IDs of bug related-data and the development logs in OSS projects. In this way, the rationale of this study is to determine which components or keywords of the SZZ algorithm provide a better precision in locating bugs in the development logs; and to determine if, for instance, the keyword ’bug’ is more often found in the proximity of a bug ID than the ’fix’ keyword or ’# and numeric values’. Our hypothesis is that, in the development logs, none of the keywords (’fix’ or ’bug’) are to be preferred to the ’# + digit’ identifier approach when locating bug IDs, and they all equally contribute in finding the set of bug IDs.

In order to obtain the complete list of IDs that should be considered, the SZZ algorithm is ’dissected in its basic components, or proxies, in terms of their precision at pointing to bug IDs. In this paper, we will first create the full set of bug IDs from the two sources of information (i.e., VC system and BT system), and second evaluate the precision and recall of the individual SZZ components in identifying or locating bug IDs.

The rest of the paper is organised as follows: Section 2 illustrates the related work and the novelty of our work. Section 3 discusses the methodology to retrieve the bug IDs of OSS projects sampled in this paper. We also illustrate the definitions of precision and recall, and how they evaluate in the context of the set of bug IDs within the development logs. Section 4 presents a worked example, where the approach is illustrated step-by-step. We replicate the steps and evaluate the proposed approach in all the 344 OSS projects and highlighted the results in Section 5. In addition, we discuss the threats to validity in this research in Section 6. Finally, Section 7 discusses the findings and present the conclusion of the work.

2. Related Work

In this section, we report the related work that was carried out for development of methods to retrieve bug-related data. We also report the tools that were used to trace the bug-fixing commits to the bug traces in the BT systems. (3 sentence to suggest the way forward or a road map)

A number of studies have postulated a convergence between development logs and bug-related data. For instance, these researchers [5,8,11,18] instantiated the full SZZ algorithm to identify missing links between development logs and bugs related data in the BT system and VC System. (+ additional recent work been done.. 3 sentence limitation and future work to show identify the gap)

In addition, the studies of Kim et al. and Sliwerski et al. [11,18] also asserted that the SZZ algorithm is locating the development logs and bug related-data. In this pioneering study by Kim et al. and Sliwerski et al. applied the components of the algorithm to automatically identify bug-introducing changes in the source code. (+ additional recent work been done.. 3 sentence limitation and future work to show identify the gap)

Similarly, Alencar da Costa et al. [6] evaluates five SZZ implementations using 10 OSS projects and provides a systematic mean for evaluating the development logs and bug related data retrieved by a given SZZ implementation. (+ additional recent work been done.. 3 sentence limitation and future work to show identify the gap)

3 paragraphs to show and justify the gap and demonstrate the significance of this work.

The aforementioned researchers tended to focus on identify bug-introducing changes in OSS projects rather than dissecting each components of the SZZ algorithm. In addition, the novelty of this paper lies with an attempt to synchronise either the missing development logs or bug related data of software projects using CVSAnalY[1] and Bicho[2] tool sets for posterior analysis.

CVSAnalY retrieves development logs from VC systems such as Subversion and Git, repository logs and transforms the logs in SQL database format. It also retrieves meta-data such as committer's name, lines added, lines removed, commited messages, etc. [14].

On the other hand, Bicho retrieves and stores bug related data from a given bug BT system and also transforms the bug related data in SQL database format. Currently, it supports Bugzilla (>4), Sourceforge.net (abandoned), Jira (unstable), Launchpad, Allura (unstable) and Github (unstable). In the same way, both tools (i.e., Bicho and CVSAnalY) requires large amount of interaction to locate development logs and bug IDs although the tool sets recovers the missing development logs and bug-related data accurately. (+ 2 sentence current state of the art ov CVS and Bicho)

3. Methodology

During the process of locating the development logs and bug related-data of OSS projects, the analysis conducted in this paper will evaluate the precision and recall of each component of SZZ algorithm. In particular, the instantiation and implementation of the SZZ algorithm uses (i) the Fixed term, (ii) the Bug term, and (iii) the '#' identifier (with numeric values 54321). Every component needs to be checked for their precision and recall when isolating the bug IDs in the development logs.

[1] https://github.com/MetricsGrimoire/CVSAnalY
[2] https://github.com/MetricsGrimoire/Bicho

However, in this paper, we partially implement and instantiate the SZZ algorithm [18] to locate the development logs and bugs IDs within the OSS projects we sampled and obtained from GitHub for this study. In our formulation, we search for bugs described by the '#' sign and numeric values (e.g., #1234), that are linked to the ID of a bug report in the BT system and the development logs in the VC system. Likewise, we search for the bugs described by the 'Fixed' and 'Bug' keywords that are linked to the ID of a bug report in the BT system and the development logs in the VC system using Bicho and CVSAnalY tool sets. In its original formulation, the SZZ algorithm also searches for keywords like 'Bug', 'Fix' and others. For the purpose of this study, we developed a tool-chain to search for bug IDs in the BT system and the development logs in the VC system within the two databases of Bicho and CVSAnalY respectively. The tool-chain will ultimately enable researchers and practitioners to circumvent the missing bug related-data and the development logs into Bicho and CVSAnalY respective databases for posterior analysis. (one sentence to show the approach was revisited to justify the dissection)

We illustrate the definitions in sub-section (3.1) and the criteria in sub-section (3.2). The inclusion of the sampling is reported in sub-section (3.3) as well as the steps from raw data to sets of bug IDs in sub-section (3.4).

3.1. Definitions

In this paper, we applied the standard terms used in the information and retrieval terminology. The terms true positive (TP), true negative (TN), false positive (FP) and false negative (FN) are defined relatively to the '#' identifier as follows:

- $TP_{\#,p}$ = number of '#' identifiers that refer to a bug ID (in project p);
- $FP_{\#,p}$ = number of '#' identifiers that do not refer to a bug ID (in project p);
- $FN_{\#,p}$ = number of bug IDs that are not identified by a '#' sign in the development logs (in project p);
- $TN_{\#,p}$ = number of the development logs that do not refer to bug IDs and not considered as referring to bug IDs (in project p);

As illustrated above (i.e. in Section 3.1) we partitioned the three components of the SZZ algorithm, based on the keywords used. In this way, given each keyword or identifier, its relative precision is defined as

$$Precision_{\#,p} = \frac{TP_{\#,p}}{TP_{\#,p} + FP_{\#,p}} \tag{1}$$

$$Precision_{bug,p} = \frac{TP_{bug,p}}{TP_{bug,p} + FP_{bug,p}} \tag{2}$$

$$Precision_{fix,p} = \frac{TP_{fix,p}}{TP_{fix,p} + FP_{fix,p}} \tag{3}$$

Similarly, the recall (or true positive rate) of using one or the other component of the SZZ algorithm is defined as follows:

$$Recall_{\#,p} = \frac{TP_{\#,p}}{TP_{\#,p} + FN_{\#,p}} \tag{4}$$

$$Recall_{bug,p} = \frac{TP_{bug,p}}{TP_{bug,p} + FN_{bug,p}} \tag{5}$$

$$Recall_{fix,p} = \frac{TP_{fix,p}}{TP_{fix,p} + FN_{fix,p}} \tag{6}$$

When considering the 'Fix' and 'Bug' keywords, similar definitions to the ones above (i.e.,in section (3.1) were apply. All the development logs and bug related-data were manually analysed for the projects composing the sample, and the precision and recall of each project we sampled in this paper are summarised in Table 6 (in Section 5.1).

3.2. Criteria for Selection

We impose certain requirements and criteria in sampling the OSS forge. The requirements and criteria itemised below are fundamental in sampling the required number of OSS projects needed for this study:

- The OSS project must be maintained and remain under active development at the time of the study. This is to avoid the development logs and bug related-data for not being obsolete or retrieving in-complete data sets. Thus only a project repository that can be processed by CVSAnalY and Bicho tool sets have been considered.
- The OSS project must have at least two accessible repositories: (i) a code repository in the VC system and (ii) a bug repository in the BT system hosted via GitHub. This is to facilitate a joint and automatic identification of IDs of bug related-data and the development logs. The data sets from these repositories will be extracted using the tool-chain developed for this study by integrating CVSAnalY and Bicho. This criteria has an impact on the OSS projects selected in this research, because the repositories should have a format that can be processed by Bicho and CVSAnalY tool sets. In particular: (provide a github link to the tool-chain)
    1. The development logs must be based on Git or Subversion, therefore any VC system that is supported by CVSAnalY. (currently it support ...)
    2. The OSS project BT system repository must be based on Bugzilla (>4), Sourceforge.net (abandoned), Jira (unstable), Launchpad, Allura (unstable) and Github (unstable). therefore any tracker supported by the Bicho tool. (currently it support ...)

3.3. Project Sampling

The FLOSSmole project contains the population of GitHub OSS projects as of February 2013[3]. The size of the OSS projects data dump is 3,640,870. Formula 7 below was applied in sampling the required number of OSS projects and deliberated for inclusion in the study:

$$\text{Sample size} = \frac{Z^2 \times (p) \times (1-p)}{c^2} \tag{7}$$

Where:

- Z = confidence level
- p = percentage picking a choice
- c = confidence interval (or merging error)

The result of the sizing is 384 projects: a randomiser retrieved 384 random numbers between 1 and 3,640,870, corresponding to the project IDs to be considered for inclusion in the study. After manual inspection, we found that 40 of the sampled OSS projects were empty. Hence giving an overall number of 'alive' OSS projects. (i.e., 344 OSS projects met the criteria we imposed in sub-Section 3.2). We then analysed their development logs and bug-related data automatically using our approach and the tool-chain we developed for this research [16]. The data sets (i.e., 344 OSS projects repositories) can be found on Figshare[4].

However, the essence of the study is to determine which keywords such as 'Fix', 'Bug' or whether the '#' identifier provide a better precision; as well as analysing their respective precision and recall in locating bug IDs in the development logs or version control system manually. In this way, we identified only a subset of 20 OSS projects randomly extracted from this larger data sets (i.e., the 344 OSS project that we sampled in order to improve the bug data in their respective repositories). Table 1 summarises the metrics and key values of the 10 OSS projects we sample for the study in this paper.

[3] As found in http://flossdata.syr.edu/data/gh/2013/

[4] https://figshare.com/s/be471b90e70865db6a30

Table 1. Attributes of the projects selected.

S/N	Project Name	URL	Dev. logs	kLOC	No. Devs
1	Brackets	github.com/adobe/brackets	16,665	300k	285
2	Leaflet	github.com/Leaflet	3,677	6.89	194
3	Reddit	github.com/reddit	6,000	200	140
4	CocoaPods	github.com/CocoaPods	4,800	22.2	160
5	Puma	github.com/puma	1000	8.39	30
6	AutoMapper	github.com/AutoMapper	700	2.78	50
7	MonoDevelop	github.com/mono/monodevelop	30,000	900	170
8	CodeHub	github.com/thedillonb/CodeHub	305	12	2
9	Manos	github.com/jacksonh/manos	1,113	66.4K	27
10	puppet	github.com/puppetlabs/puppet	20,256	379	337

3.4. Bug ID Data Extraction

The extraction of bug IDs from raw data is comprised of two parts: one is the storage of each project's metadata in the CVSAnalY and Bicho databases; the second is the extraction of the bug-related IDs that appear in either databases. After the raw data extraction, it is necessary to determine how the two sources were aligned, in terms of contained bug IDs; also, each of the SZZ component needs to be evaluated against the data obtained in the development logs.

Incisively, the steps are detailed as follows:

- **Identifying bugs in development logs:** The extraction process is to store the development logs using CVSAnalY. Amongst the tables generated by CVSAnalY, we specifically queried the SCMlog table, which holds the number and unique IDs of the development logs in the VC system. The right-hand side of Table 1 depicts the composition of the CVSAnalY table that was used for retrieval of the information referring to bugs. In order to locate the bugs in the development logs, we used the SCMlog table, which mentions the number and unique IDs of the development logs in the VC system. In the presence of a bug ID, the development logs also mentioned the bug ID with the '#1234' format. For the purpose of this research, we are only interested in the bug IDs that are being mentioned by developers: bug IDs do not necessarily need to be "fixed" or "resolved".
- **Locating bugs related-data in the BT system:** Locating bugs related-data in the BT system, we used Bicho to obtain and store all the information contained in the BT System of the OSS projects we sampled; as well as all the issues or bugs reported by the users of the software and confirmed as such by developers on BT system. Two of the tables created by Bicho are the issues and issues_ext_bugzilla table, where the status (open or closed) or the message accompanying the entry is stored and imported for publication by the relative GitHub tracker. In this way, we queried specifically the issues_ext_bugzilla table to obtain the set of unique IDs of bugs report and confirmed by developers.
  Note: it is worth mentioning that both CVSAnalY and Bicho encountered issues in retrieving the development logs and bug related-data. We imposed a 15 seconds delay in the tool-chain before sending request to the server each time when mining the development logs and bug related-data.
- **Obtaining the complete set of bug IDs:** in this step, it is necessary to combine the two sets of bug IDs from the development logs and the bug related-data, in order to determine the intersection and the union of the two sets. The intersection contains the common subset of bug IDs, as found in both the development logs and bug related-data of all the OSS projects. On the other hand, the union of the sets is useful to obtain the overall number of bug IDs found in the two databases, and to identify how many bug IDs are actually missing from each database.
- **Evaluating the precisions of each SZZ component:** the final step of the data extraction is to evaluate the formulas [(1) - (6)] as defined above. Since the uncertainty of the SZZ algorithm is based on the free text of the development logs, we evaluated the TP, TN, FP and FN terms only considering the entries of the development logs, for each of the analysed software project.

In the next Section 4, we implement the steps for one of the projects, as a worked example.

4. Worked Example: Bracket Project

In this section, we practically conduct an experiment and implement the steps to produce the TP, TN, FP and FN terms from an exemplar case study. The precision and recall are also evaluated to exemplify the approach. The project that we use for such exemplification is the Brackets[5] project. Brackets is described as a "code editor for the web: it is a large JavaScript project, with around 300kLOC of source code in the main development trunk". There are over 350 contributors in this OSS project. The overall number of development logs (i.e., commits) exceeds 17,000, and 110 releases have been published.

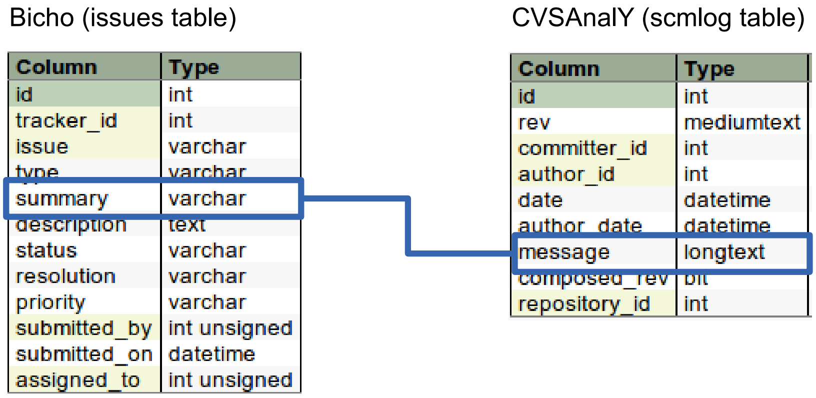
4.1. Identifying Bugs in Development Logs and Bug Related-Data

This step involved cleansing the raw data and the storage of development logs and bug related-data including their IDs in both CVSAnalY and Bicho tool sets for the Brackets project. The query for the '#' sign, followed by numeric values in the development log imported with CVSAnalY, produces a large number of false positives (FT). Therefore, we manually checked to confirm whether the message field in the SCMlog table of CVSAnalY contains a reference with a '#' sign. As a way of an example, a false positive is found with the #3057 bug ID (as found in the Bug Tracker) of the Brackets project. The information relative to the #3057 ID, as found in the SCMlog table, reads as `Merge pull request #3507 from adobe/jasonsanjose/getting-started-fr`. The ID of this bug should return the development logs in the SCMlog table of CVSAnalY referring to the actual #3507 bug related-data in the BT system. Instead, the information refers to a request to merge some changes in the distributed VC system. We marked these occurrences as 'false positives' and excluded them from the pilot study in order to maintain and adhere to the standard and criteria we set in this study. [15,16].

Similarly, using the '#' proxy over 2,000 development logs was linked to a request of pulling a merge from another distributed repository into the original one under GitHub. These were filtered out automatically, after discarding these false positives, we obtained a set of 366 bug IDs that were mentioned in the CVSAnalY Database as the development logs in the SCMlog table from the VC system. On the other hand, another set of 349 bug IDs that were mentioned in the issue tracker table in Bicho database from the BT system.

In addition, the traditional heuristic developers leave hints or clue about the bug related-data in the development logs which established a link between bugs related-data and the development logs in both tools, as this is widely used to mark bug fixes [20]. In this paper, we specifically focused on quantifying the bugs, and the the development logs in Bicho and CVSAnalY databases respectively that are not linked to bug fixes. Figure 1 depict how we mapped and linked the two databases: bug IDs were searched and compared in the summary field of the Issues table of Bicho database, and in the message field of the SCMlog table of CVSAnalY database respectively.

[5] https://github.com/adobe/brackets



Bicho (issues table)

Column	Type
id	int
tracker_id	int
issue	varchar
type	varchar
summary	varchar
description	text
status	varchar
resolution	varchar
priority	varchar
submitted_by	int unsigned
submitted_on	datetime
assigned_to	int unsigned

CVSAnalY (scmlog table)

Column	Type
id	int
rev	mediumtext
committer_id	int
author_id	int
date	datetime
author_date	datetime
message	longtext
composed_rev	bit
repository_id	int

Figure 1. Corresponding fields linked in Bicho and CVSAnalY adapted from [15].

Finally, we then manually analysed to check each of the remaining bugs in both databases, in order to ensure each of the remaining IDs are pointed to real bug related-data with the corresponding development logs.

4.2. Obtaining the Complete Set of Bug IDs

In this regard, we retrieved all the bug IDs contained in the BT system of Bracket project, and then created the first set (S1), containing over 4,000 bug IDs; afterwards, we produced an SQL query to mimic the SZZ algorithm in order to retrieve all the development logs containing either the 'Fix' or 'Bug' keywords or the '#' symbol from the development logs. We obtained only 3,117 development logs when queried the **SCMlog** table in CVSAnalY database from the VC system. In the same vein, 1,865 development logs contained unique bug identifiers: this list of bug IDs created the second set of bug IDs ($S2$), as found in the VC system.

Below are the results of basic operations on S1 and S2 when considering the '#' identifier:

- $S1 = 4{,}634$
- $S2 = 3{,}117$
- $S1 \cap S2$ (Common bugs & logs) = 267
- S1 - $S2$ (only in Bicho - bug related-data) = 4,367
- $S2$ - S1 (only in CVSAnalY - development logs) = 1,865

Observing the results from list above, the bug related-data in Bicho from the BT system of Bracket project contains 4,634 bug IDs, although that is not the overall set. Using the '# with numeric values' proxy, there are 1,865 more bug IDs which were found in the development logs that were not reported in the BT system. On the other hand, the development logs are much more incomplete, since only 3,117 bugs are mentioned in the development logs (i.e., commits). The set of common bugs i.e., those appearing in both the BT system (i.e., Bicho) and the VC system (i.e., CVSAnalY) is 267. Similarly, using the 'Bug' and 'Fixed' keywords also produces further results, as summarised in Table 2

Table 2. Bug IDs and sources of information.

	BT system	Dev logs			
SZZ part	S1	S2	S1 ∩ S2	S1 - S2	S2 - S1
#	4,634	3,117	267	4,367	1,865
Fix	4,634	63	31	4,603	32
Bug	4,634	154	79	4,555	75

The results reported in table 2 appear to support the assumption that by combining all the lists of bug IDs found with the various proxies of the SZZ components, it is possible to obtain a complete set of bug IDs contained in the two information sources, i.e. the BT system and the VC system. More importantly, it is evident that bug IDs are missing in both tools (i.e., Bicho and CVSAnalY). Thus it is fundamental to analyse each of the SZZ algorithm components to reinforce it completeness when obtaining the complete sets of bugs IDs. Moreover, it is crucial to carry-out an analysis of what is found in the unstructured development logs, to make sure that what is retrieved is a bug ID and not a false positive. The analysis is detailed in the next subsection of this paper.

4.3. *Evaluating the Precision of Each SZZ Component*

We performed a manual analysis of a random sample of 100 development logs to determine the precision and recall of each of the SZZ algorithm components. Since the development logs are unstructured, we then need to analyse each development logs manually to determine whether 'Fix' or 'Bug' or the '#' identifier are referring to a bug. Regarding the # symbol, we found 58 development logs (out of 100) that mentioned #: after a close inspection, we realised that 57 of these development logs were actually referring to a bug ID. In this case, we classified them as true positive, (TP), while only one of those logs did not refer to a bug ID (i.e., the false positives, FP). Furthermore, there are 42 development logs that mention either 'Fix' or 'Bug', but do not have a unique ID attached (i.e., the false negatives). From Tables 3–5 we present the number of development logs that were referring to TP, FP, FN and TN for the Brackets project and the remaining 9 OSS projects. Given the formulas above in Section 3.1 we evaluated the precision of 'using the # symbol as a predictor of the presence of the bug ID as equal to 0.983 while the recall is 0.576.

Table 3. Number of development logs that were referring to TP, FP, FN and TN for # symbol.

#number (e.g., #123)						
S/N	**Project Name**	**No. logs**	**TP**	**FP**	**FN**	**TN**
1	Brackets	100	57	1	42	0
2	Leaflet	22	6	0	16	0
3	Reddit	74	40	12	22	0
4	CocoaPods	100	18	0	82	0
5	Puma	81	11	2	63	0
6	AutoMapper	68	19	6	43	0
7	MonoDevelop	100	19	3	78	0
8	CodeHub	42	0	0	42	0
9	Manos	100	1	3	46	0
10	puppet	100	0	22	92	0

Table 4. Number of logs that were referring to TP, FP, FN and TN for Fixed.

Fixed						
S/N	**Project Name**	**No. logs**	**TP**	**FP**	**FN**	**TN**
1	Brackets	100	41	41	18	0
2	Leaflet	22	1	12	9	0
3	Reddit	74	14	21	39	0
4	CocoaPods	100	15	77	8	0
5	Puma	81	6	61	14	0
6	AutoMapper	68	8	29	31	0
7	MonoDevelop	100	14	55	31	0
8	CodeHub	42	0	35	7	0
9	Manos	100	0	63	37	0
10	puppet	100	0	58	17	0

Table 5. Number of logs that were referring to TP, FP, FN and TN for Bug.

Bug						
S/N	**Project Name**	**No. logs**	**TP**	**FP**	**FN**	**TN**
1	Brackets	100	1	3	96	0
2	Leaflet	22	0	0	22	0
3	Reddit	74	1	1	72	0
4	CocoaPods	100	0	3	97	0
5	Puma	81	0	6	75	0
6	AutoMapper	68	0	1	67	0
7	MonoDevelop	100	29	8	63	0
8	CodeHub	42	0	8	34	0
9	Manos	100	0	2	98	0
10	puppet	100	0	18	82	0

Similarly, regarding the 'Bug' keyword, we found that only one development log mentioning 'Bug' also referred to a bug ID (i.e., TP), while three development logs mentioning 'Bug' were not related to any bug ID (i.e., FP); the remainders of the logs created the FN element. Using the 'Bug' keyword as a predictor of a bug ID had a precision of 0.25 and a recall of 0.01. Finally, for the 'Fix' keyword, we evaluated a precision of 0.500 and a recall of 0.695. Based on the precision and recall, we then computed the F-measure using the formula as detailed below:

$$F - measure = 2 \times \frac{precision \times recall}{precision + recall} \tag{8}$$

The '#' symbol gained an F-measure of 0.726, the 'Fix' keyword 0.582 and the 'Bug' keyword only 0.019. Since the F-measure is often used, in the context of information retrieval, to assess the performance of searches, this further test confirms the earlier findings reported by [15,16]. Analysing the unstructured data of the development logs of the Brackets project as a pilot study, we conclude that the most precise proxy of bug IDs is the '#' identifier, when considering the free-text descriptions of changes written by developers as an addendum to their commits (development logs) in the VC systems. Comparatively, the 'Bug' keyword performs very poorly: very often developers cite the keyword without attaching the correct bug ID for future traceability.

These findings, if confirmed, will re-enforce that the traceability of bug IDs from BT systems into VC systems and vice versa can represent a real issue, at least for OSS projects. In the next section, we repeat the analysis for nine further projects, to check whether the results are confirmed in general.

5. Replication and Results

In this section we report the analysis of the overall sample projects from GitHub. In particular, we report on how many bug IDs are mentioned in the two databases, per project.

5.1. Replicability and Scalability of the Approach

After illustrating the approach applied in the previous worked example, the study was conducted again with the remaining nine additional OSS projects taken from the same repository (i.e., from GitHub). This was carried-out on a subset of 344 OSS project samples to replicate the results of manual approach and to identify the level of effort needed to replicate the examples.

Table 1 contains the summary of the analysis on internal attributes of the project. This section details the precision and recall results of each components of the SZZ algorithm.

The results of the replication of the worked example on nine further OSS projects are shown in Table 6 below. As also performed in the worked example above in Section 4, each component of the SZZ algorithm such as '#' identifier, 'Fix' and 'Bug' has its own subsets of results for precision, recall and F-measure. In order to detect the presence of bug IDs, a selection of a subset of 100 log entries per project was made for longer sets of development logs. All the logs were selected for projects with fewer than 100 logs.

Table 6. Manual evaluation of 10 OSS projects Precision, Recall and F-measure of the three main components of the SZZ algorithm.

S/N	No. Logs	# symbol			Fix			Bug		
		P	R	F	P	R	F	P	R	F
1	100	0.983	0.576	0.726	0.500	0.695	0.582	0.250	0.010	0.020
2	22	1.000	0.273	0.429	0.077	0.100	0.087	0.000	0.000	0.000
3	74	0.769	0.645	0.702	0.400	0.264	0.318	0.500	0.014	0.027
4	100	1.000	0.180	0.305	0.163	0.652	0.261	0.000	0.000	0.000
5	81	0.846	0.149	0.253	0.090	0.300	0.138	0.000	0.000	0.000
6	68	0.760	0.306	0.437	0.216	0.205	0.211	0.000	0.000	0.000
7	100	0.864	0.196	0.319	0.203	0.311	0.246	0.784	0.315	0.450
8	42	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	100	0.250	0.021	0.039	0.000	0.000	0.000	0.000	0.000	0.000
10	100	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Based on our experiment, it is evident that the use of the ′#′ identifier outperformed both the ′Fix′ and the ′Bug′ keywords in locating bug IDs and the development logs of the Bracket project. This indicates that the development logs are clearly lagging behind in terms of completeness and traceability as compared to bug related-data.

The size of the projects development logs and the time it may take to analyse and detect the presence of bug IDs should also be considered for the scalability of this approach. In terms of the sizing of the OSS projects that were sampled, it took significant amount of time and effort to manually evaluate the precision of each worked example (i.e., to determine if ′Fix′ or ′Bug′ and the ′#′ identifier are referring to a bug). As a result, the replication of large OSS projects was extended semi-automatically.

The development logs of the projects were manually analysed to evaluate if they were related to bug description or not. This technique was similar to the earlier pilot study. The same process was applied to the ′#′ identifier and the ′Bug′ and ′Fix′ keywords (as well as their derivatives, like ′Fixed′ or ′Fixing′).

For all the analysed projects, the F-measure obtained when using the ′#′ identifier is always higher than for any of the other proxies (′Bug′ or ′Fix′). In specific cases, the precision of the ′#′ identifier reaches maximum values (in Projects 1, 2 and 4); in other cases, such as Project 8, none of the SZZ components achieved any result, which is particularly concerning for bug traceability.

5.2. Trade-off Between Precision and Recall

The quid pro quo between precision and recall in the context of this paper occurs with an increased proportion of ′#′ symbol precision leading to decreased proportion of ′Fixed′ and ′Bug′ precision. In addition, the Recall proportion of Fixed and Bug component of the SZZ algorithm was high at the expense of low proportion in the Recall of ′#′ symbol. However, manually evaluating the precision and recall of puppet and CodeHub projects (i.e., project 8 and 10 as visible from the Precision and Recall curve in Figure 2 below), the proportion of the three main component of the SZZ algorithm (i.e., ′#′ symbol, fixed and bug) were zero (also visible in Table 6) because none of the development logs retrieved in that project referred to the TP and FP as defined in Section 3.1 of this paper. Similarly, same applies to the rest of the 10 OSS projects evaluated where the proportion of both zero recall and zero precision were obtained for the three main components of the SZZ algorithm.

Previous studies by Buckland et al [3] and Gordan et al [9], regarding the origins of the recall and precision trade-off assume knowledge of the size of the set of retrieved logs as a fraction of the total number of logs in the database.

In addition, the trade-off between precision and recall can be observed using the precision-recall curve in Figure 3 , and an appropriate balance between the precision and recall of the three main components of the SZZ algorithm evaluated using 10 OSS projects.

Moreover, the box plot presented in Figure 4 summarises the precision and recall of each component of the SZZ algorithm: these results indicate that, for most of the OSS projects we sampled, using the '#' identifier their precision is higher than the recall. Using the fixed keyword the majority of the OSS projects recall was higher than the precision while the rest obtained zero precision and recall as visible in Table 6, this is because none of the bug IDs were detected in the VC system in most of the OSS projects using the fixed keywords. However, for the Bug keyword as visible in the box plot in around 95% of the OSS projects obtained zero precision and recall resulting to many outliers in the box-plot. This means overall the '#' identifier is more precise in terms of detecting and locating bugs IDs and the development logs in OSS projects.

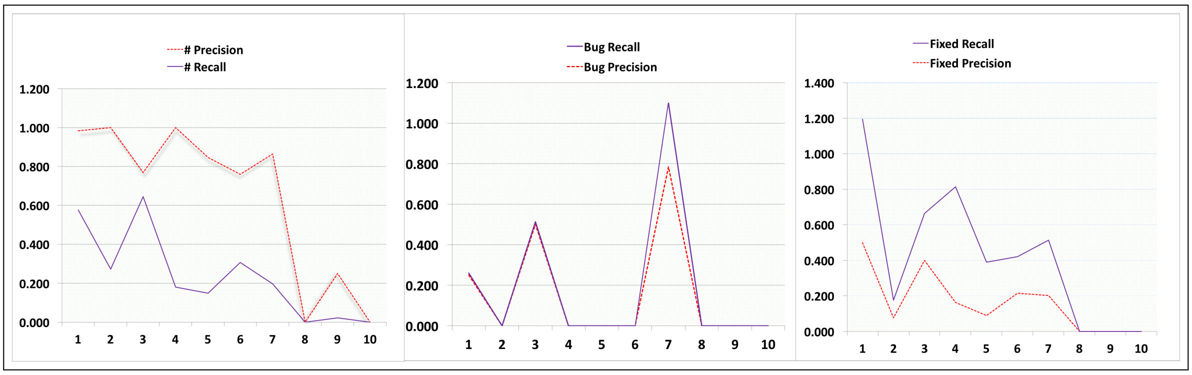


Figure 2. Precision and Recall curve of the three main components of the SZZ algorithm.

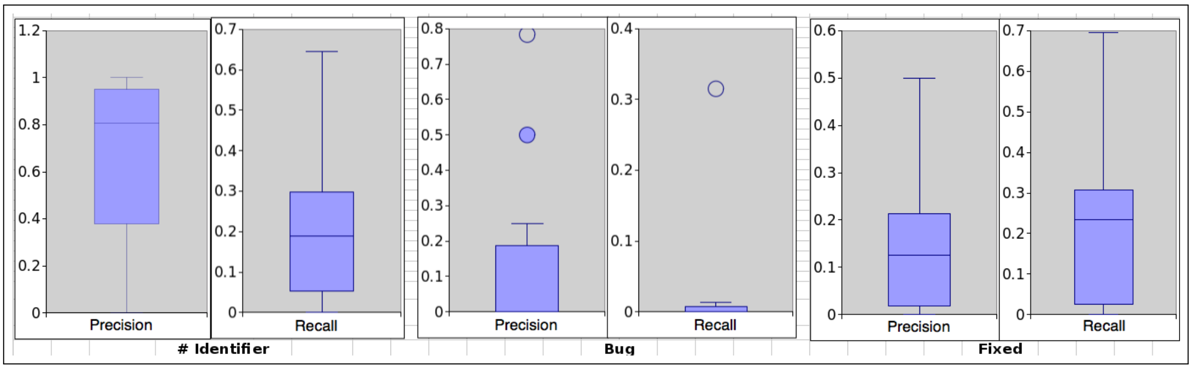


Figure 3. Box-plot: Precision and Recall of the three main components of the SZZ algorithm.

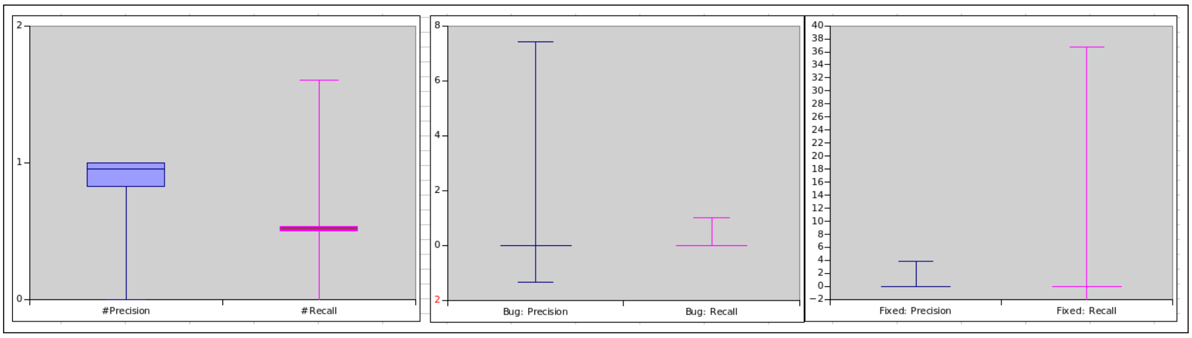


Figure 4. 344 OSS Projects: Precision and Recall of three main component of the SZZ algorithm.

5.3. Replication: With a Large Sample of OSS Projects

After evaluating the approach (i.e., SZZ components) used in the worked example above, we replicated the study with the rest of the OSS projects sampled for this research. The 344 OSS projects were extracted from the same repository (GitHub).

This section presents the precision and recall results for each project when using the each components of the SZZ algorithm of all 344 OSS projects.

In addition, the result of each component was further examined by applying a MannWhitney test to further unveil the significance of using each component and prove the scalability of the approach [19]. Similarly, the results of the replication of the 344 OSS projects can be found on Figshare[6]. Also, each component of the SZZ algorithm (# identifier, “Fix” and “Bug”) has its own subsets of results for precision, recall and F-measure for each project. We compute the precision, recall and F-measure as detailed in Section 3.1 of this chapter.

Moreover, the results of the replication of the 344 OSS projects on Figshare[7] was used to compute the statistical significance of using each component of the SZZ algorithm (i.e., # identifier, Fix and Bug) presented in the matrix Table 7 below using WESSA[8]

The analysis of the 10 OSS projects was carried out manually. Also, in some OSS projects only the top 100 subsets of development logs were considered when evaluating each component, while the rest of the 10 OSS projects had fewer than 100 development logs. However, where the proportion of the three main components of the SZZ algorithm (i.e., # symbol, fixed and bug) were zeros. Thus, none of the logs retrieved in that project referred to the TP and FP as mentioned in the previous Section 5.1 and defined in Section 3.1 of this paper. Similarly, in most of the projects, we observed that the use of the ’#’ identifier outperformed both the ’Fix’ and the ’Bug’ keywords in the identification of the bug IDs from the development logs. Iteratively, this is an important finding: development logs are clearly lagging behind in terms of completeness and traceability, as compared to the bug related-data.

Table 7. Statistical significance of the SZZ algorithm (p-value).

P#	**Pfixed**	**Pbug**	**R#**	**Rfixed**	**Rbug**	**F#**	**Ffixed**	**Fbug**	
P#		2.9193E-96	4.9707E-104	1.8594E-66	3.815E-94	1.0329E-98	6.1107E-64	1.0139E-92	1.0139E-92
Pfixed			3.8349E-20	1.8921E-85	0.79254	0.58966	2.6336E-91	0.1038	0.1038
Pbug				9.1993E-95	9.7608E-19	4.7604E-19	1.194E-102	5.6958E-11	5.6958E-11
R#					1.3882E-87	1.3882E-87	6.1107E-64	1.6694E-86	7.4359E-104
Rfixed						0.7962	1.2705E-93	0.16261	1.4199E-13
Rbug							1.294E-96	0.24014	7.07E-14
F#								3.2641E-92	7.9778E-111
Ffixed									4.9455E-06
Fbug									

Legend:		
PBug = Precision bug	P# = Precision #	Pfixed = Precision fixed
RBug = Recall bug	R# = Recall #	Rfixed = Recall fixed
Fbug = F-measure bug	F# = F-measure #	Ffixed = F-measure fixed

The matrix table in Table 7 shows the significance of the three main components of SZZ algorithm. This was evaluated using the precision, recall and F-measure against each component of the SZZ algorithm that is, the # identifier and “Bug” and “Fix” keywords.

The process followed to extract the precision and recall data was similar to the pilot study: the development logs of the projects were extracted semi-automatically using the tool chain by issuing the following SQL query.

Code 1: SQL Query to retrieve the development logs

```
select message from scmlog where repository_id= ? and
```

[6] https://figshare.com/s/be471b90e70865db6a30

[7] https://figshare.com/s/be471b90e70865db6a30

[8] Web-enabled scientific services and applications (WESSA) is a free Statistics Software calculation: http://www.wessa.net/

```
message NOT like '%Merge pull request%' and
message like '%#%'
```

The same syntax for the SQL query was repeated for the '#' identifier and the 'Bug' and 'Fix' keywords. When comparing all the SZZ components, the '#' identifier is always more significant than any of the other proxies ('Bug' or 'Fix'). In some of the projects, the precision of the '#' identifier reaches maximum values as well. In other projects, none of the SZZ components achieve a result, which is particularly concerning for the purpose of bug traceability.

6. Threats to Validity

In this section, we will discuss the threats to validity that are specific to our findings. The four main threats to validity in this paper are internal, external, construct and conclusion validity.

1) Internal validity: In this regards, the study was carried-out to dissect the development logs retrieved using CVSAnalY, and bug related-data retrieved using Bicho. Ideally, the tools were typically run independently thus the development logs and bug related-data are stored in their respective databases created by both tools automatically. The extraction of development logs and bug related-data of each project's data set was carried out simultaneously, in order to avoid any discrepancies or over-lagging using the tool-chain developed for this study. This enabled us to evaluate and dissect each SZZ component to minimise any other external factors that might have had an effect on our empirical findings in this study.

2) Construct validity: With regards to the construct validity, which deals with the relationship between theory and observations, we sampled 10 OSS projects from GitHub in order to pilot the dissection of the SZZ algorithm in its basic components, or proxies, in terms of their precision and recall in locating the development logs and bug IDs in VC system and BT system.

Also, we evaluate the precision and recall of each SZZ components, in order to avoid errors. During the evaluation process, we automated the process using the tool-chain we developed for this study. Moreover, we used the widely adopted metric F-measure to assess the SZZ technique. We also measured the performance of the existing techniques that is to say, the SZZ algorithm on each basic component (i.e., the use of '# with numeric values', 'Fixed' and 'Bug' via Precision-Recall and F-Measure. In mitigating such threat, we began with a pilot study, in which we studied 1 OSS projects and manually analysed each development log to determine if 'Fix' or 'Bug' or the '#' identifier were referring to a bug in the BT system and VC system respectively.

3) External validity: The results from this study are only generalisable to Bicho and CVSAnalY tool sets and the 10 OSS projects we sampled from GitHub via FlossMole. In addition, we do not claim that the results would apply to all MSR tools we mentioned in this study. Further empirical studies are needed to validate this generalisation. We leave this as a future work.

We welcome researchers in software engineering community to build on the results in this study and replicate our study with different and large data sets (i.e., OSS projects), and dissect the SZZ algorithm to advance the body of knowledge. Replicating this study with different and large data sets from different repositories could help reduce this threat. We leave this as a future work.

4) Conclusion validity: Due to the number of OSS projects we sampled in this study, as well as the non-normality of development logs and bug related-data, we evaluate the performance of each SZZ algorithm component using the measures of precision and recall [1,17] in locating bugs in the development logs and bug related-data of the OSS projects.

7. Discussion and Conclusion

This paper demonstrates that the process of locating bug related-data in the development logs in OSS projects, is far from established or reputable. Developers or contributors in OSS project tend to record their actions in different ways, and very often the bug-fixing commits are not reflected onto, and from, the corresponding bug related-data in the BT system. Often bug relate-data, that should be considered as the baseline for all the bugs in a project, is found to be incomplete, and further bug IDs are found when harvesting the development logs.

This work has two main contributions: the first is to show an approach to build a (more) complete set of bug IDs that were documented in the evolution of a software system. This comprises the analysis and parsing of both the development logs and the bug related-data: this is required because we found that commonly OSS projects hold different sets of bug IDs when interrogating their own BT system and VC system. The second contribution is an in-depth analysis of the SZZ algorithm, that has been used extensively by researchers in software maintance and evolution to track the bug fixing commits of software systems. We partitioned the algorithm in its three basic components, and with a manual check-up, we showed the precision and recall of each component in locating bug identifiers in the development logs. We found that the guideline of using the '#' symbol and the bug ID largely outperforms the other proxies to detect bug related-data and the development logs.

Furthermore, we demonstrated that the process of collecting data related to bugs and the development logs using OSS projects, is far from established or standardised. Developers tend to record their actions in different ways, and very often the bug-fixing commits are not reflected onto and from the corresponding BT system.

Manually inserting the references to bug IDs is clearly not achieving the required traceability, and a better (automated) approach should be designed to have the two sources of data aligned and in sync. The possible way to do so would be to generate an automatic commit into the development logs that details the bug-fixing activity, as obtained by the BT system. Likewise, when the BT system is not aligned to the VC system, an entry could be automatically generated to insert the bug development activity, as detailed in the development logs, into the BT system.

The results in this study are relevant to the research community: models, techniques and empirical approaches that use defect data, would produce seemingly different (or complementary) results, when the complete set of bug related-data was to be extracted and considered for study. Replication studies could be performed to assess whether the results as proposed in past papers could be complemented with further evidence of bug-fixing activity. On the other hand, the use of the SZZ algorithm shows that some keywords ('Fix' and 'Bug') are linked to less precision and higher recall. This result reinforces the message that practitioners should synchronise the development logs with the bug related-data by using the standard '#' notation for locating development logs and bug IDs in OSS projects.

References

1. Anvik, J.; Hiew, L.; Murphy, G.C. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.
2. Ayari, K.; Meshkinfam, P.; Antoniol, G.; Di Penta, M. Threats on building models from cvs and bugzilla repositories: The mozilla case study. In *Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '07*, Riverton, NJ, USA, 2007; IBM Corp., pp. 215–228.
3. Buckland, M.; Gey, F. The relationship between recall and precision. *J. Am. Soc. Inf. Sci.* **1994**, *45*(1), 12–19.
4. Casalnuovo, C.; Devanbu, P.; Oliveira, A.; Filkov, V.; Ray, B. Assert use in github projects. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15*, Piscataway, NJ, USA, 2015; IEEE Press, pp. 755–766.
5. Cubranic, D.; Murphy, G. Hipikat: recommending pertinent software development artifacts. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, 2003, pp. 408–418.
6. da Costa, D.A.; McIntosh, S.; Shang, W.; Kulesza, U.; Coelho, R.; Hassan, A. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering* **2016**.
7. Fischer, M.; Pinzger, M.; Gall, H. Populating a release history database from version control and bug tracking systems. *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, 2003, pp. 23–32.
8. Fischer, M.; Pinzger, M.; Gall, H. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, 2003, pp. 23–32.
9. Gordon, M.; Kochen, M. Recall-precision trade-off: A derivation. *Journal of the American Society for Information Science* **1989**, *40*(3), 145–151.

10. Gotel, O.; Finkelstein, A. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, 1994, pp. 94–101.
11. Kim, S.; Zimmermann, T.; Pan, K.; Whitehead Jr., E.J. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, 2006, pp. 81–90.
12. Lormans, M.; van Deursen, A. Can lsi help reconstructing requirements traceability in design and test? In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, 2006, pp. 10 pp.–56.
13. Mockus, A.; Votta, L. Identifying reasons for software changes using historic databases. In *Software Maintenance, 2000. Proceedings. International Conference on*, 2000, pp. 120–130.
14. Robles, G.; Koch, S.; González-Barahona, J.M.; Carlos, J. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *In Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, 2004, pp. 51–55.
15. Romo, B.A.; Capiluppi, A.; Hall, T. Filling the gaps of development logs and bug issue data. In *Proceedings of The International Symposium on Open Collaboration*, 2014; ACM, p. 8.
16. Romo, B.A.; Capiluppi, A. Towards an automation of the traceability of bugs from development logs: A study based on open source software. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, EASE '15*, New York, NY, USA, 2015; ACM, pp. 33:1–33:6.
17. Sebastiani, F. Machine learning in automated text categorization. *ACM computing surveys (CSUR)* **2002**, *34*(1), 1–47.
18. Śliwerski, J.; Zimmermann, T.; Zeller, A. When do changes induce fixes? *ACM SIGSOFT Software Engineering Notes* **2005**, *30*(4), 1–5.
19. Wessa, P. Free statistics software, office for research development and education. 2016.
20. Wu, R.; Zhang, H.; Kim, S.; Cheung, S.C. Relink: Recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11*, New York, NY, USA, 2011; ACM, pp. 15–25.
21. Yang, H.; Wang, C.; Shi, Q.; Feng, Y.; Chen, Z. Bug inducing analysis to prevent fault prone bug fixes. 2014. Retrieved February 15, 2015 from http://software.nju.edu.cn/zychen/paper/2014SEKE1.pdf.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.