Locating Bug IDs and Development Logs in Open Source Software Projects (OSS): An Experience Report

Bilyaminu Auwal Romo
Digital Technology
Coventry University, Coventry, UK
Email: bilyaminu.auwalromo@coventry.ac.uk

Andrea Capiluppi
Department of Computer Science
Brunel University London, UK
Email: andrea.capiluppi@brunel.ac.uk

Ajaz Ali
Digital Technology
Coventry University, Coventry, UK
Email: ajaz.ali@coventry.ac.uk

***Abstract*—The development logs of software projects, contained in Version Control (VC) systems can be severely incomplete when tracking bugs, especially in open source projects, resulting in a reduced traceability of defects. Other times, such logs can contain bug information that is not available in bug tracking system (BT system) repositories, and vice-versa: if development logs and BT system data were used 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.**

Considering a sample of 10 OSS projects and their development logs and BT systems data, the two objectives of this paper are (i) to determine which of the keywords 'Fix', 'Bug' or the '#' identifier provide the better precision; and (ii) to analyse their respective precision and recall at locating the larger amount possible of bug IDs manually.

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 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 software project.

***Index Terms*—Bug traceability, Bug-fixing commits, SZZ algorithm, Development logs, Bug data.**

I. 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 development logs, and by the steps made by the developers in its solving. Finally, its status should change into '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 and BT systems. However, this does not seem to be the case and this forms the **motivation** for this study.

Traceability links are needed to perform various software evolutionary activities, for instance to design and build defect prediction models [21]. However, the available tools to document development logs lack integration [7] with the bug tracking systems: thus, two independent sets of bug-related data are produced, filling different databases [17]. It has been suggested that using the bug IDs from development logs could help to identify and recover missing traceability links [10] [13] [2]. These logs need to be manually or semi-automatically analysed and compared, to determine if development logs and IDs from BT systems are referring to the same set of bug IDs, or if they refer to disjoint sets.

So far, development logs have been mostly searched for bugs in basically two ways: (i) by using keywords such as 'Fixed' or 'Bug' [14]; and (ii) by searching for references to bug reports, for instance the use of the "#" sign and various numeric values (e.g., #1234), which are linked to the ID of a bug [5] [7] [12]. The SZZ algorithm is an example of an approach that combines keywords and proxies to detect bug-fixing commits [19].

In this research, 10 Open Source Software (OSS) projects from GitHub were analysed to pilot and demonstrate an approach to recover the union of sets of bug IDs, from their development logs and BT systems. Therefore, the main **objective** of this paper is, practically, to locate as many bug IDs as possible, using both development logs and BT 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 BT 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 individual keywords or components in locating bug IDs in OSS development logs and evaluate the most precise keywords or components. The study of Casalnuovo et al. [4] demonstrates that researchers in software engineering have difficulty in deciding the right keywords for locating bugs in development log. Thus, the researchers have applied nine (9) keywords in retrieving project evolution history related to development logs: 'error', 'bug', 'fix', 'issue', 'mistake', 'incorrect', 'fault', 'defect' and 'flaw'. Previous studies have used mainly two keywords that is 'bug' and 'fix' to identify bugs in development logs. Although, one can argue that the convention used by software developers in detailing their

© 20XX IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

actions in the source code may differ from one project to another especially in OSS projects. Thus to foster and re-enforce the number of keywords or components researchers might deliberate in code manipulation and analysis, it is vital to measure each component or keywords precision in locating bug IDs in OSS projects. In this way, the **rationale** of this work is to determine which components or keywords of the SZZ algorithm provide the better precision in identifying bugs from development logs; and to determine if, for instance, the keyword 'fix' is more often found in the proximity of a bug ID than the 'bug' keyword or '# + digit'. Our basic **hypothesis** is that, in 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.

The rest of the paper is organised as follows: Section II illustrates the related work and the novelty of our work. Section III 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 IV presents a worked example, where the approach is illustrated step-by-step. We replicate the steps and evaluate the proposed approach in all the 10 OSS projects and highlighted the results in section V. In addition, we discuss the threats to validity in this research in Section VI. Finally, Section VII discusses the findings and present the conclusion of the work.

II. Related Work

In this section, we report the related work that was carried out for developement 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 issue trackers.

A comparison of instantiation of the full SZZ algorithm in tracings development logs and bugs of issues data has been implemented. These researchers [5][8][12][19], attempted to integrate and identify missing links between development logs and bugs related data into Issues Tracking Systems (i.e., BT system) and Version Control System.

In addition, the studies of Kim et al. and Sliwerski et al. [12] [19] have demonstrated and validated manually that the development logs and Bug related data are actually referring and pointing to actual fixes using this algorithm (SZZ) as well as automatically and accurately identify bug-introducing changes.

Similarly, Alencar da Costa et al. [6] evaluates five SZZ implementations using 10 OSS projects and provides a systematic mean for evaluating the data that is generated by a given SZZ implementation.

The novelty of this paper lies with an attempt to synchronise either the missing development logs or bug issues data of software projects retrieved by these tools (CVSAnalY[1] and Bicho[2]) and stored into their respective databases for posterior analysis.

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

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

CVSAnalY retrieves information from Control Version system, Subversion or Git repository logs and transforms it in a SQL database format. It also retrieves meta-data from concurrent versioning systems (CVS) including commmitter's name, lines added, lines removed, commited messages, etc. [15]. Bicho stores information from a given bug tracking system (BTS) to a database format. So far, it only works with the SourceForge.nets BTS [11], both tools require large amount of interaction to locate bug IDs but they recover missing logs and bugs accurately.

However, Kim et al. and Sliwerski et al. [12] [19] have demonstrated and validated manually that the development logs and Bug related data are actually referring and pointing to actual fixes using this algorithm (SZZ) as well as automatically and accurately identifying bug-introducing changes.

III. Methodology

The analysis performed in this section is an attempt to evaluate the precision and recall of the various components of the SZZ algorithm when detecting bug-fixing commits. In particular, the implementation of the SZZ algorithm uses (i) the 'Fixed' term, (ii) the 'Bug' term, and (iii) the '#' identifier (with digits, say #12345). Each of these components need to be checked in terms of their precision and recall when isolating the bug IDs in the development logs.

In this paper, we partially implement the SZZ algorithm [19] to trace bugs and logs within the OSS projects sample we obtained from GitHub. In our formulation, we only look for bugs described by the '#' sign and numeric values (e.g., #1234), that are linked to the ID of a bug. In its original formulation, the SZZ algorithm also searches for keywords like 'Bug', 'Fixed' and others. A tool was developed to search for these IDs within the two databases, and to combine the results into intersection and union of sets.

We illustrate the definitions in sub-section (III-A) and the criteria in sub-section (III-B). The inclusion of the sampling is reported in sub-section (III-C) as well as the steps from raw data to sets of bug IDs in sub-section (III-D).

A. Definitions

In the context of this study, and using 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 as follows (and relatively to the # identifier):

- $TP_{\#,p}$ = 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 development logs (in project p)
- $TN_{\#,p}$ = number of 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 III-A) we partitioned the SZZ algorithm in three components, based on the keywords

used. Therefore, given each keyword or identifier, its relative precision is defined as

$$Precision_{\#,p} = \frac{TP_{\#,p}}{TP_{\#,p} + FP_{\#,p}} \quad (1)$$

$$Precision_{bug,p} = \frac{TP_{bug,p}}{TP_{bug,p} + FP_{bug,p}} \quad (2)$$

$$Precision_{fix,p} = \frac{TP_{fix,p}}{TP_{fix,p} + FP_{fix,p}} \quad (3)$$

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

$$Recall_{\#,p} = \frac{TP_{\#,p}}{TP_{\#,p} + FN_{\#,p}} \quad (4)$$

$$Recall_{bug,p} = \frac{TP_{bug,p}}{TP_{bug,p} + FN_{bug,p}} \quad (5)$$

$$Recall_{fix,p} = \frac{TP_{fix,p}}{TP_{fix,p} + FN_{fix,p}} \quad (6)$$

When considering the 'Fix' and 'Bug' keywords, similar definitions to the ones above (i.e.,in section (III-A) apply. All the development logs and BT system data were manually checked for the projects composing the sample, and the precision and recall of each project are summarised in Table VI (in Section V-A).

B. 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 research:

- The OSS project must be maintained and remain under active development. This ensures the analysed development logs and BT data will not be obsolete or irrelevant to our approach. Following this methodology, only repositories that can be processed by CVSAnalY and Bicho have been considered.
- The OSS project must have at least two accessible repositories: (i) a code repository and (ii) a bug repository. This is to facilitate a joint and automatic synchronisation of missing data. Data from these repositories will be extracted by the tool chain developed for this research using 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. In particular:
  1) The development logs must be based on Subversion, VC system or Git, therefore any VC system supported by CVSAnalY.
  2) The OSS project BT system repository must be either Bugzilla (> 4), Sourceforge.net (abandoned), Jira (unstable), Launchpad, Allura (unstable) and Github (unstable), therefore any tracker supported by the Bicho tool.

C. Project Sampling

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

$$Sample_size = \frac{Z^2 * (p)^*(1-p)}{c^2} \quad (7)$$

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' projects of 344 in which we analysed their development logs and bug data automatically using our approach and the proposed tool-chain 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 10 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 I summarises the metrics and key values of the 10 OSS projects we sample for the study in this paper.

D. 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 database. 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 first step is to store the development logs via the CVSAnalY tool set. Among the tables generated by CVSAnalY, we specifically queried the **SCMlog** table, which holds the number and unique IDs of changes in the development log or version control system. The identity of developers who perform these changes and the comment message

[3]As found in http://flossdata.syr.edu/data/gh/2013/
[4]https://figshare.com/s/be471b90e70865db6a30

TABLE I
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

describing the changes applied to the code. The right-hand side of table 1 depicts the composition of the CVS-AnalY table that was used for retrieval of the information referring to bugs. In order to identify or locate the bugs in development logs, we used the **SCMlog** table, which mentions the number and unique IDs of changes 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 bug IDs that are being mentioned by developers: bug IDs do not necessarily need to be "fixed" or "resolved".

- **Identifying bugs in BT System data**: to locate or identify bugs in BT system, we used the Bicho tool to obtain and store all the information contained in the bug tracking 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 bug tracking 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 numbers and IDs of bugs reported and confirmed by developers.
  Note: it is worth mentioning that both CVSAnalY and Bicho encountered downloading issues during the mining the development logs and bug data (i.e., BT system data). The tool chain developed for this research imposed a delay of 15 seconds before sending each request from the server when mining development logs and bug data, which made the process of obtaining all the raw data slightly longer.
- **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 BT system 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 development logs and BT system data of all the 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 IV, we implement the steps for one of the projects, as a worked example.

IV. WORKED EXAMPLE: BRACKET PROJECT

In this section, we analyse the steps that were performed 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. In this project, there are over 180 contributors to the code. The overall number of commits exceeds 10,000, and 88 releases have been published.

A. Identifying bugs in Development logs and BT data

This step involved the cleaning of raw data and the storage of bug IDs for both CVSAnalY and Bicho 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. Therefore we manually checked 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 information in

[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 [17]

SCMlog referring to the actual #3507 bug 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 as well as the extended study [17],[16].

In the case of the Brackets project, over 2,000 messages refer to the pattern searched for using the '#' sign, but they are all 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 log messages and another set of 349 bug IDs that were mentioned in the issue tracker (i.e., BT system) in Bicho.

In addition, the traditional heuristic *developers leave hints or links about bug fixes in change logs* was used to produce a link between bugs/issues and 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 logs in Bicho and CVSAnalY that are not linked to bug fixes. Figure 1 shows how the two databases are linked: bug IDs were searched and compared in the summary field of the **Issues** table of Bicho, and in the message field of the **SCMlog** table in CVSAnalY.

Finally, we manually analysed each of the remaining bugs in both databases, to make sure that each of the remaining IDs pointed to real bugs.

B. Obtaining the complete set of bug IDs

At first, 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 a query to mimic the SZZ algorithm in order to retrieve all the logs containing either the '#' symbol or the 'Fix' or 'Bug' keywords from the development logs. In this regards, only 3,117 logs were obtained when queried the development logs, and 1,865 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, the results of basic operations on S1 and S2 are provided when considering the "#" identifier:

- $S1$ = 4,634
- $S2$ = 3,117
- $S1 \cap S2$ (Common bugs) = 267
- S1 - $S2$ (only in the BT system) = 4,367
- $S2$ - S1 (only in the Dev.logs) = 1,865

From the list above, we observed that the bug-tracking system of Brackets contains 4,634 bug IDs, but this is not the overall set. Using the '# with digits' proxy, 267 more bug IDs 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 reported in the commits. The set of common bugs i.e., those appearing in both the BT system and the development logs is 267. Using the 'Bug' and 'Fixed' keywords also produces further results, as summarised in Table II below.

By combining all the lists of bug IDs found with the various proxies (i.e., 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 from either source, so it is fundamental to analyse each for completeness.

The second study that needs to be performed is 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. This analysis is performed in the next subsection of this paper.

C. Evaluating the precision of each SZZ component

In this part of our evaluation, we performed a manual analysis of a random sample of 100 development logs to determine the precision and recall of each of the SZZ components. Since the logs are unstructured, we need to analyse each one manually to determine whether 'Fix' or 'Bug' or the '#'

TABLE II
BUG IDS AND SOURCES OF INFORMATION

	BT system	Dev logs			
SZZ part	S1	S2	$S1 \cap S2$	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

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 (i.e., the 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 logs that mention either 'Fix' or 'Bug', but dont have a unique ID attached (i.e., the false negatives). From Table III, IV-C and V 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 III-A we evaluated the precision of 'using the # symbol as a predictor of the presence of the bug ID as equal to 0.983. The recall of such an approach reached 0.576.

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 as detailed below:

TABLE III
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 IV
NUMBER OF LOGS THAT WERE REFERRING TO TP, FP, FN AND TN FOR FIXED

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \quad (8)$$

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 V
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

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 [17], [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 to 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.

V. 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.

A. Replicability and scability of the approach

After illustrating the approach used in the worked example above, we replicated the study with a further set of nine OSS projects, extracted from the same repository (GitHub). This was done for two basic reasons: to replicate the manual approach on a subset of the 344 OSS projects sampled; and to report on the scalability of the approach, in order to give an indication of the effort needed to replicate the experiment. A brief analysis of the internal attributes of the projects was conducted, which is summarised in Table I. The section below presents the precision and recall results when using the individual components of the SZZ algorithm.

The results of the replication of the worked example on nine further software projects are shown in Table VI below. As also performed in the worked example above, each component of the SZZ algorithm ('#' identifier, 'Fix' and 'Bug') has its own subsets of results for precision, recall and F-measure. For longer sets of development logs, we randomly selected a subset of 100 log entries per project depending on the total number of logs in the project. For the projects that had fewer than 100 logs, all the logs were selected, while for the projects that had 100 logs and above we only took the top subset of 100 logs randomly and analysed them manually, to detect the presence of bug IDs.

Similarly to the Brackets project above, and for every analysed project, 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. It is an important finding: development logs are clearly lagging behind in terms of completeness and traceability, as compared to the BT data.

Thus, the scalability of the approach has to be considered under two aspects: (i) size of the projects development logs; and (ii) the time it took to analyse and detect the presence of bug IDs of all the projects we sampled in this research.

In terms of the size of the projects development logs, for the 10 OSS projects in the worked example it took a significant amount of effort and time to manually evaluate the precision of each SZZ component. For instance, for the Brackets project we took 100 development logs. To manually analyse each log three times (i.e., to determine if 'Fix' or 'Bug' and the '#' identifier are referring to a bug) would require a significant amount of effort and time considering the size of the development logs for every project in the 10 OSS projects for this research. As a result, the replication of large OSS projects was extended semi-automatically. This will be detailed in the next section of this paper.

The process followed to extract the precision and recall data was similar to the pilot study: the development logs of the projects were analysed manually and a decision taken as to whether the log was actually related to a bug description or not. The process was repeated for 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 achieve any result, which is particularly worrying for the purpose of bug traceability.

B. Trade-off between recall and precision

The trade-off 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 VI) because none of the logs retrieved in that project referred to the TP and FP as defined in Section III-A 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 2, 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 3 summarises the precision and recall of each component of the SZZ algorithm: these results indicate that, for most of the OSS projects, 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 VI, this is because none of the bug IDs were detected in the version control logs 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 in version control logs.

TABLE VI
MANUAL EVALUATION OF 10 OSS PROJECTS PRECISION, RECALL AND F-MEASURE OF THE THREE MAIN COMPONENTS OF THE SZZ ALGORITHM

Manually analysed		# symbol			Fix			Bug		
S/N	No. Logs	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

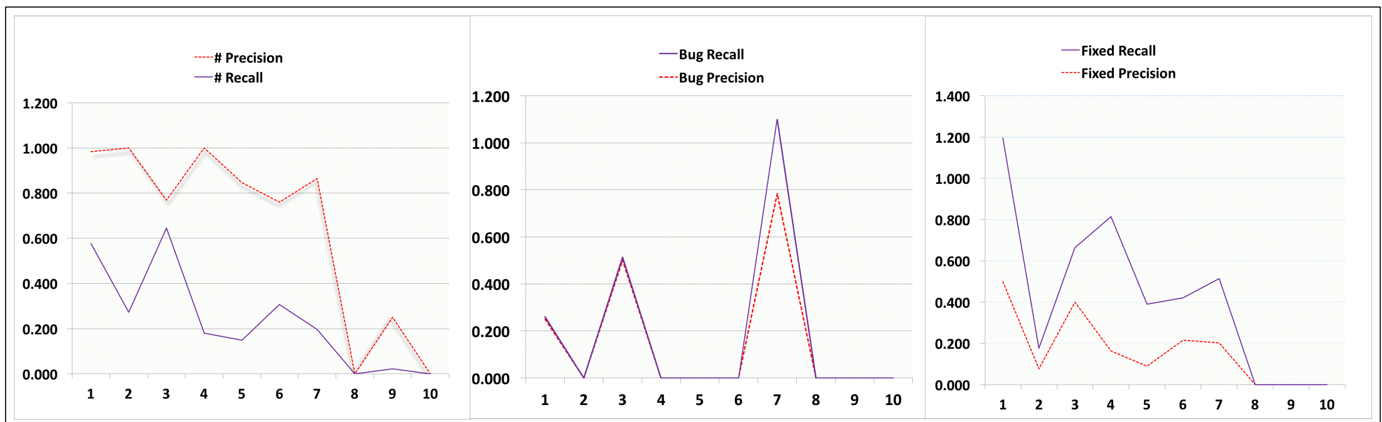

Figure 2. Precision and Recall curve of three main component of the SZZ algorithm

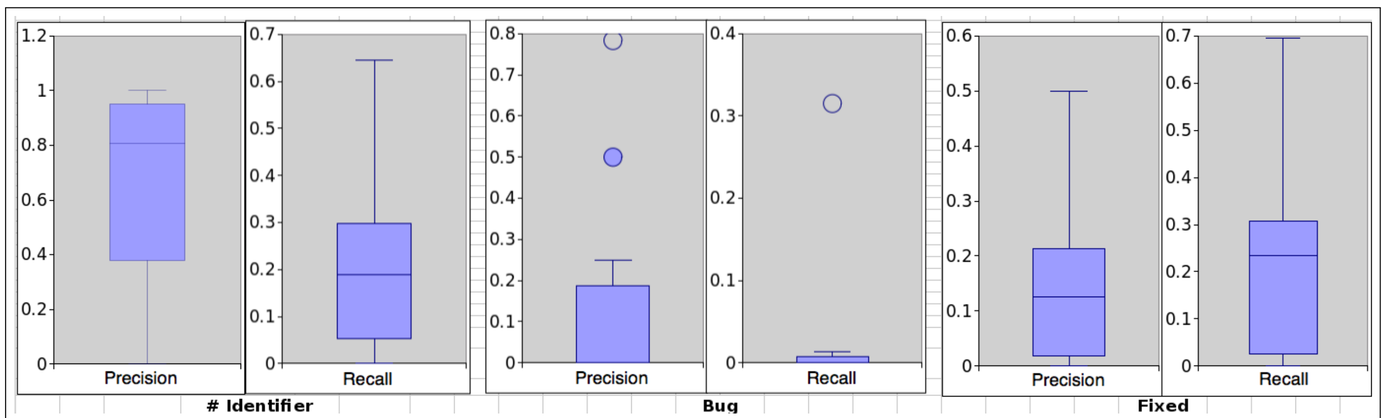

Figure 3. Box-plot: Precision and Recall of the three main component of the SZZ algorithm

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 BT data.

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 VC log

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

VI. Threats to validity

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

1) Internal validity: With respect to internal validity, the evaluation was between development logs retrieved using CVSAnalY, and BT data retrieved using Bicho. Both tools are executed independently and produce independent results. Similarly, the development logs and BT data are stored in different localised databases created by both tools automatically. The extraction process that is to say, mining development logs and BT data of each projects data set was carried out simultaneously, to avoid any discrepancies or over-lagging using the tool chain developed for this research. This allowed us to evaluate and dissect each SZZ component in this study to the best of our knowledge, and thus to minimise any other external factors that might have had an effect on the results in our empirical study.

2) Construct validity: With respect to construct validity, which deals with the relationship between theory and observations, we sampled 10 OSS systems from GitHub in order to pilot the dissection of the SZZ algorithm in its basic components, or proxies, in terms of their precision at pointing to bug IDs.

Also, we have evaluated the precision and recall of each SZZ components at identifying or locating bug IDs. In order to avoid errors or mistakes during our evaluation, we automated the process using the tool chain developed for this research. Moreover, we used the widely adopted metric F-measure to assess the SZZ technique as well as its improvement. We measured the performance of the existing techniques that is to say, the SZZ algorithm on each basic component (i.e., the use of '# 123', 'Fixed' and 'Bug' via Precision-Recall and F-Measure To mitigate such a 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.

3) External validity: The results from this study are only generalisable to Bicho and CVSAnalY tool sets and the Brackets projects we sampled from GitHub via FlossMole. In addition, we do not claim that these 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 future work too.

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

4) Conclusion validity: With respect to conclusion validity, due to the number of OSS projects we sampled in this study, as well as the non-normality of development logs and BT data sets, we evaluate the performance of each SZZ algorithm component using the measures of precision and recall [18] [1] in detecting and locating bugs in version control logs of the OSS projects.

VII. Discussion and Conclusion

This paper demonstrates that the process of locating bug data in development logs, when using Open Source project, is far from established or reputable. Developers 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 Tracking System. Often BT data, that should be considered as the baseline for all the bugs in a project, is found to be incomplete, and further 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 tracking systems: this is required because we found that commonly OSS projects hold different sets of bug IDs when interrogating their own BT system and the VC system.

The second contribution is an in-depth analysis of the SZZ algorithm, that has been used extensively by researchers 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 detecting 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-fixing commits.

Furthermore, we demonstrated that the process of collecting data related to bugs, when using open-source 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 paper 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 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 BT data by using the standard '#' notation for locating development logs and bug IDs in OSS projects.

VIII. AKNOWLEDGMENTS

We would like to thank Dr Boyce Sigweni and Dr Nemitari Ajienka for their constructive feedback on an earlier version of this paper

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[2] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta. 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, pages 215–228, Riverton, NJ, USA, 2007. IBM Corp.

[3] M. Buckland and F. Gey. The relationship between recall and precision. *J. Am. Soc. Inf. Sci.*, 45(1):12–19, Jan. 1994.

[4] C. Casalnuovo, P. Devanbu, A. Oliveira, V. Filkov, and B. Ray. Assert use in github projects. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 755–766, Piscataway, NJ, USA, 2015. IEEE Press.

[5] D. Cubranic and G. Murphy. Hipikat: recommending pertinent software development artifacts. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 408–418, May 2003.

[6] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. Hassan. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, 2016.

[7] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, pages 23–32, 2003.

[8] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 23–32, Sept 2003.

[9] M. Gordon and M. Kochen. Recall-precision trade-off: A derivation. *Journal of the American Society for Information Science*, 40(3):145–151, 1989.

[10] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101, Apr 1994.

[11] I. Herraiz, D. Izquierdo-Cortazar, F. Rivas-Hernández, J. Gonzalez-Barahona, G. Robles, S. Duenas-Dominguez, C. Garcia-Campos, J. F. Gato, and L. Tovar. Flossmetrics: Free/libre/open source software metrics. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pages 281–284. IEEE, 2009.

[12] S. Kim, T. Zimmermann, K. Pan, and E. J. Whitehead Jr. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, pages 81–90. IEEE, 2006.

[13] M. Lormans and A. van Deursen. 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*, pages 10 pp.–56, March 2006.

[14] A. Mockus and L. Votta. Identifying reasons for software changes using historic databases. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 120–130, 2000.

[15] G. Robles, S. Koch, J. M. Gonzlez-Barahona, and J. Carlos. 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)*, pages 51–55, 2004.

[16] B. A. Romo and A. Capiluppi. 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, pages 33:1–33:6, New York, NY, USA, 2015. ACM.

[17] B. A. Romo, A. Capiluppi, and T. Hall. Filling the gaps of development logs and bug issue data. In *Proceedings of The International Symposium on Open Collaboration*, page 8. ACM, 2014.

[18] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[19] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[20] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung. 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, pages 15–25, New York, NY, USA, 2011. ACM.

[21] H. Yang, C. Wang, Q. Shi, Y. Feng, and Z. Chen. Bug inducing analysis to prevent fault prone bug fixes. 2014. Retrieved February 15, 2015 from http://software.nju.edu.cn/zychen/paper/2014SEKE1.pdf.