Filling the Gaps of Development Logs and Bug Issue Data

Bilyaminu Auwal Romo   Andrea Capiluppi   Tracy Hall
Department of Information Systems and Computing
Brunel University
London, United Kingdom
{bilyaminu.auwal,andrea.capiluppi,tracy.hall}@brunel.ac.uk

ABSTRACT

It has been suggested that the data from bug repositories is not always in sync or complete compared to the logs detailing the actions of developers on source code.

In this paper, we trace two sources of information relative to software bugs: the change logs of the actions of developers and the issues reported as bugs. The aim is to identify and quantify the discrepancies between the two sources in recording and storing the developer logs relative to bugs.

Focussing on the databases produced by two mining software repository tools, CVSAnalY and Bicho, we use part of the SZZ algorithm to identify bugs and to compare how the "defects-fixing changes" are recorded in the two databases. We use a working example to show how to do so.

The results indicate that there is a significant amount of information, not in sync when tracing bugs in the two databases. We, therefore, propose an automatic approach to re-align the two databases, so that the collected information is mirrored and in sync.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

Keywords

Bug traceability, bug-fixing commits

1. INTRODUCTION

The integration of different tools for software development is problematic when the tools, that should track complementary artefacts, loose consistency in the recording of the events. One such example is the traceability of bugs in the development logs: when bugs are discovered, developers should mention their existence in the development logs. Likewise, they should open the appropriate procedure in the issue tracker, marking it as "open"; similarly, when a bug has been fixed, the developers should mention its "fixed" status in the development logs and mark it as "closed" in the issue tracker.

Past research has established that there is inconsistency in how bugs are reported, when analysing the issue tracker and the development logs of a software project [1]. It has serious consequences: by considering only certain bug-fixing commits for automated (e.g., prediction) algorithms, researchers might obtain a skewed set of data, and produce a model that might be severely biased by how developers recorded their actions.

Integrating and syncing issue trackers and development logs becomes of paramount importance, especially in open source software projects. Furthemore, information is limited in distributed development since shared meetings to track the issues of the project are not general. In the context of open source projects, developers and testers consider such tools as a medium of communication and collaboration in the project community [3].

In this paper, two different tools for mining software repositories were considered: the issue-tracker parser Bicho and the development log parser CVSAnalY, to extract and store the data relative to the issues recorded for the project, and the development logs of a project, respectively [4, 6]. Our current data-set is a sample of 664 projects extracted from the GitHub development platform: for each project, we extracted and stored the development logs and the issues (i.e., bugs) recorded by the developers throughout the evolution of the source code.

The Bicho and CVSAnalY tools do not interact with each other, but run independently, produce independent reports, and fill in different databases [5]. Such reports are widely used in empirical software engineering. Therefore, automatization and completion of missing bug data become of critical importance [2]. Specifically in the bug-tracking case, developers are expected to record how, when, where and by whom bugs are fixed. Thus, the first objective of this study is to assess how the two generated databases differ when considering the bugs that affect the system under study; the second is to evaluate if we can fill the localised, missing data in either database in an automated way. As a result, the evaluation is based on the entities that exist in either database, and that could be used to re-align the entities mentioned in the databases created by Bicho and CVSAnalY.

Given the objectives above, this paper proposes an automated way to fill the gaps that are observed in databases that record bugs of a software system. We detail the steps performed to extract bug data using one of the projects in our collected sample to illustrate this process (Section 2).

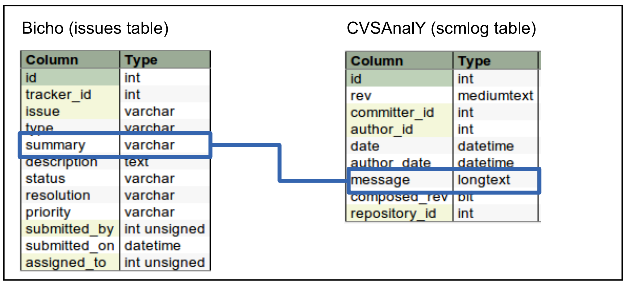
Then, we provide the results of integrating two sources of data (Bicho and CVSAnalY) in order to create a complete picture of the number of bugs reported in an example open source project (Section 3). After this, we introduce an automated approach to complete any gaps identified on either database, by using the correspondence of data in different tables (Section 4). Finally, we highlight the related work, and how our approach advances the state of the art (Section 5), and we propose the future steps of this research alongside the conclusion to this work (Section 6).

2. WORKED EXAMPLE

In this section, we use an example project to illustrate the problem that we observed when analysing most of the projects sampled from GitHub. This section presents the data that can be obtained when analysing the *brackets* project, a "code editor for the web".[1] *Brackets* 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 several releases have been published. Obtaining and cleaning the data used in this study was performed through sequential steps, as detailed below.

2.1 Development Logs – CVSAnalY

The first step was to store the development logs via the CVSAnalY toolset [6]. Among the tables generated by the tool, we specifically queried *scmlog*, which holds the number and unique IDs of changes in the version control system, the identity of developers who perform these changes and the comment message describing the changes applied to the code. The right-hand side of Figure 1 shows the composition of the CVSAnalY table that was used for the extraction of the information referring to bugs.



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

2.2 Issue Tracker Data – Bicho

The second in our data preparation process was to obtain, and store in a separate database all bugs reported for the same system, using the Bicho toolset [4].[2] Bicho retrieves all data regarding issues reported by users of a project, and confirmed as such by developers. One of the tables created is the *issues* table, where the status ("open", "closed".), or the message accompanying the entry is stored and imported for publication by the relative GitHub tracker.

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. Discrepancies or commonalities were flagged and summarised in a Venn diagram.

2.3 Locating Bug Data – SZZ algorithm

The third step involves the logic of how to retrieve bugs: we plan to implement the full SZZ algorithm [8], but in this example we perform it only partially. In its formulation, the algorithm should look for the terms "Bugs," or "Fixed" (case-insensitive) in message logs, along with the '#' sign, that shows the ID of a bug. In the example below, we show how to find and clean the bugs that have been addressed both in CVSAnalY and Bicho, by issuing a number preceded by a '#' sign.

- For the bugs in the Bicho dataset, we queried the *issues* table and extracted all the "summary" fields if they contained a # sign followed by a number: a typical result in this case would be `[PM] Fix #3057: Toggle Block Comment doesn't work if the open/close delimiters are the same`. This message of course signals that the bug with ID #3507 was fixed, with additional information on what was done.

- In order to track messages recorded as dealing with bugs in the version control system, we queried the *scmlog* table, and specifically the "message" field. We check whether this field contains a reference with a # sign. Looking for the #3057 bug, the only information found in the *scmlog* table reads as `Merge pull request #3507 from adobe/jasonsanjose/getting-started-fr`. The ID of this bug should return development information in *scmlog* referring to the actual bug in the issue tracking system. Instead, the information refers to a request to merge some changes in the distributed version control system. We marked these occurrences as "false positives," and excluded them from the study.

2.4 Data Cleaning: False Positives and True Positives

The fourth step was the cleaning and storage of bug IDs for both CVSAnalY and Bicho. The query for the '#' sign followed by numeric values in development log imported with CVSAnalY produces a large number of false positives. In the case of the *brackets* project, over 2,000 messages refer to the pattern searched through 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 are mentioned in the CVSAnalY messages, and another set of 349 bug IDs that are mentioned in the issue tracker by 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 [10]. In our case, we specif-

[1]`https://github.com/adobe/brackets`
[2]The tracker for the issues of the *brackets* project can be found at the URL `https://api.github.com/repos/adobe/brackets/issues`.

ically focused on quantifying the bugs/issues, and the logs in Bicho and CVSAnalY that are not linked to bug fixes.

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.

3. RESULTS

This section details the results of our approach on brackets project in our sample (subsection 3.1).

3.1 Results — brackets Project

The results of the analysis of bugs in the *brackets* project, when identified with the # sign, are visualised in the Venn diagram of Figure 2.

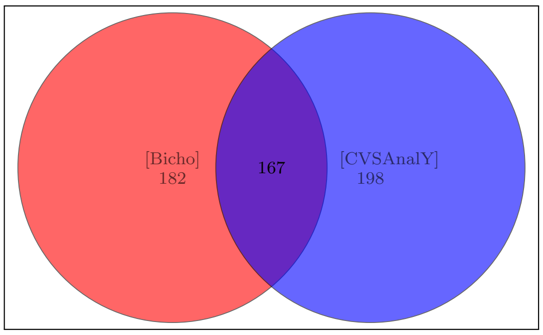


Figure 2: Intersection of the sets of IDs

As visible, the number of bug IDs that were found in **both** CVSAnalY and Bicho is around 1/3 (i.e., 167 bugs IDs, that represents the intersection of the sets in the Venn diagram) of the total number (i.e., 547, the union of all the sets in the diagram). Another 1/3 of the bug IDs are only found in Bicho, while the rest of bug IDs are reported and found in CVSAnalY, but never summarised into issues retrieved by Bicho.

This result varies across projects, depending on the style of how the information on issues is handled by developers, and it only refers to how developers refer to bug IDs. We did not infer any information on whether the bug was fixed, or opened: we just investigated the presence of the bug IDs in the two databases. This is because our aim was to identify and quantify discrepancies between the two sources in recording and to developer information relative to bugs. In the next section we explore the possibility of filling the missing data from one database to the other, by means of the entries of either database.

4. AUTOMATING THE INTEGRATION OF MISSING DATA

Observing the tables of Bicho and CVSAnalY (displayed in Figure 1) and their attributes, we propose to use bug-related data in either database to fill the missing data as detected in the other database. For instance, the 198 bug IDs and attributes stored by CVSAnalY (but not found by Bicho) could be used to fill the *summary* and other attributes in the Bicho database. In consequence, automating the integration of Bicho and CVSAnalY will involve a series of steps such as Pre-processing, Analysing and Post-processing [4]. These steps can be further subdivided as outlined below:

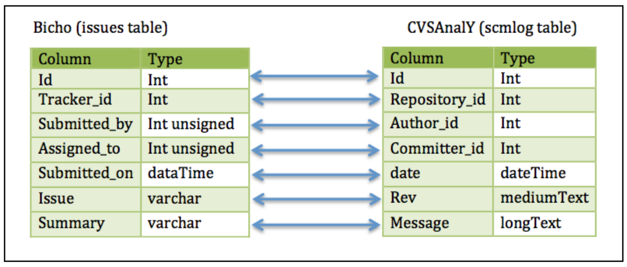
1. For every project, retrieve the bug data on development logs and tracker issues from the *scmlog* and *issues* tables, as linked aboved in Figure 1. In the case of the GitHub repository, this process is automatable by replacing with the name of the project in the `https://api.github.com/repos/adobe//issues` URL to retrieve the issue tracker data; and replacing with the name of the project in the `https://github.com/adobe/` URL to retrieve the development logs.

2. Once obtained and stored the data in the two databases, apply the SZZ algorithm to identify the missing bug data in the *scmlog* table of CVSAnalY and *issues* table of Bicho viceversa. In our case, we found 198 missing bug IDs in the issue archives, but present in the CVSAnalY database; we also found 182 bug IDs in the Bicho database, but not present in the CVSAnalY database.

3. Using a matching algorithm, produce a joint list of bug IDs, and classify them in "missing from the Bicho database", "missing from the CVSAnalY database", or "present in both".

4. In the cases where one bug ID is missing from either database, we propose to use the data found in the other database, to fill in the missing data of that ID automatically. For instance, let's assume that ID #45 was found only in the CVSAnalY database, but not in the Bicho database. The "message" field in the CVSAnalY database could be used to automatically fill the "summary" field of the Bicho database. Similarly, the "Id" of the CVSAnalY database could be used as the "Id" of the Bicho database. "Committer_id" from the CVSAnalY database could be used to fill in the "Assigned_to" attribute in Bicho, and so on.

5. The item that must be carefully linked between the two databases is the project ID: since the two databases are distinct, it is likely that the Repository_id sequentially stored by CVSAnalY will be different from to the Tracker_id (also sequentially) stored by Bicho. In the *brackets* case, CVSAnalY stored the log data in our database with a Repository_id=346, while Bicho stored the issues for the same project with a Tracker_id = 60. An extra table has to be created to automatically link the two IDs in the databases.



Bicho (issues table)

Column	Type
Id	Int
Tracker_id	Int
Submitted_by	Int unsigned
Assigned_to	Int unsigned
Submitted_on	dataTime
Issue	varchar
Summary	varchar

CVSAnalY (scmlog table)

Column	Type
Id	Int
Repository_id	Int
Author_id	Int
Committer_id	Int
date	dateTime
Rev	mediumText
Message	longText

Figure 3: Corresponding fields linked in Bicho and CVSAnalY

All the fields of CVSAnalY or Bicho from the above Figure 1 have been mapped to link the corresponding attributes in both tools. Figure 3 above shows the corresponding fields that have been linked to fill up the missing bug data in either database.

5. RELATED WORK

In this section, we report the related work regarding the tools that trace the bug-fixing commits to the bug traces in the issue trackers. We also report the related work that developed methods to retrieve bug-related data.

The Linkster tool involves a series of steps to retrieve, parse as well as convert and link the data sources [1]. As a result, it requires significant manual effort to analyse recovered links which might be much more accurate. On the other hand, RELINK [10] collects information automatically from the source code repository and bug tracking system, builds the resulting information linked between bugs/issues or logs and output the identified links. In general, both these tools require a large amount of interaction but they recover missing logs and bugs/issues accurately. Our approach completes these tools by filling the missing data in either database in an automatic way.

Regarding the approaches for mining bug-related data, the SZZ is one of the most used algorithms to look for bug-fixing commits, with a set of simple rules [8]. A recent study by Shepperd et al. [7] contributes a significant and more appropriate way to clean bug-related data sets for empirical research applied to the NASA case study.

The study in [9] applies a formal mathematical model to automate the process of identifying missing links between bug-fixing commits in development logs and their associated bug reports. The model is effective in recovering such missing links: what we propose here is to use the identified discrepancies to fill up the development logs or tracker issues when missing links have not been synced.

6. CONCLUSION AND FUTURE WORK

This short paper presented a procedure to automatically fill the gaps discovered in either the development logs or the issue trackers of software projects. We showed that such an approach was completely automated, when partially implementing a well-known algorithm to isolate the bug-fixing commits (i.e., the SZZ algorithm [8]. As a future work, we plan to expand this study significantly: we have extracted development logs and lists of issues from over 600 projects randomly chosen in the GitHub repository, and we plan to integrate the same approach in that sample to test the scalability of our approach. One aspect that we also plan to further investigate is related to the patterns and sequences of events. Some developers might be more reluctant to fill the information on either database, whereas other developers might be posting on the change log first, and then fill in the issue tracker later on.

7. AKNOWLEDGMENTS

We would like to thank Dr. Felipe Ortega for his feedback on an earlier version of the paper and for the constructive comments.

8. REFERENCES

[1] C. Bird, A. Bachmann, F. Rahman, and A. Bernstein. Linkster: enabling efficient manual inspection and annotation of mined data. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 369–370. ACM, 2010.

[2] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, 2012.

[3] H. Hayashi, A. Ihara, A. Monden, and K.-i. Matsumoto. Why is collaboration needed in oss projects? a case study of eclipse project. In *Proceedings of the 2013 International Workshop on Social Software Engineering*, pages 17–20. ACM, 2013.

[4] 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.

[5] M. Legenhausen, S. Pielicke, J. Ruhmkorf, H. Wendel, and A. Schreiber. Repoguard: a framework for integration of development tools with source code repositories. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 328–331. IEEE, 2009.

[6] G. Robles, S. Koch, and J. M. González-Barahona. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. 2004.

[7] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect data sets. 2013.

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

[9] A. Sureka, S. Lal, and L. Agarwal. Applying fellegi-sunter (fs) model for traceability link recovery between bug databases and version archives. In *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*, pages 146–153. IEEE, 2011.

[10] 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*, pages 15–25. ACM, 2011.