Syncing Development Logs and Bug Tracking Systems

Bilyaminu Auwal Romo
Department of Computer Science
Brunel University London, UK
Email: bilyaminu.auwal@brunel.ac.uk

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

***Abstract*—The development logs of software projects, contained in versioning control systems (VC system) 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 VC system 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.**

The aim of this work is to implement a tool-chain that supports the automatic integration (synchronisation) of various data sources, including the VCS logs and BT system data of open source software projects. The syncing process was achieved using the Bicho[1] and CVSAnalY[2] tools.

Overall, our results indicate that Version control commits system logs (VCC system logs) and Bug tracking system data (BT system data) often contain different subsets of bug-related data. Therefore, the presented syncing process has the potential to produce more complete datasets in both development logs and bug-tracking systems.

I. Approach and Toolset

Combining and cross-analysing the VCC system logs and BT system data shows incompleteness, inconsistency and skewed sets of data. In terms of bug-coverage, the VC system logs and BT system data can have the following four scenarios, depicted graphically in Figure 1.

1) The first scenario is when the set of bug IDs as found in the BTS database has no intersection with the set of bug IDs coming from the VCC logs (first left in Figure 1).
2) The second scenario is when all the bug IDs of either of the sets are contained within the other set: in the theory of sets, the cardinality of the union of the sets is the cardinality of the containing set; while the cardinality of the intersection of the sets is the cardinality of the contained set (middle part of Figure 1).
3) The third scenario is the most common: there is a subset of bug IDs that is common to the two data sources (e.g., the intersection of the sets, right of Figure 1). Apart from the common IDs, there are also (i) one subset of bug IDs that appear only in the VCC system logs, and (ii) another subset of bug IDs that appears only in the BT system.
4) The final scenario is when all the bug IDs are found in both the BT system, and the VC system: in the set theory language, the union of the sets is equal to the intersection of the sets. This is the ideal scenario, because the bugs are being mirrored exactly in the two databases.

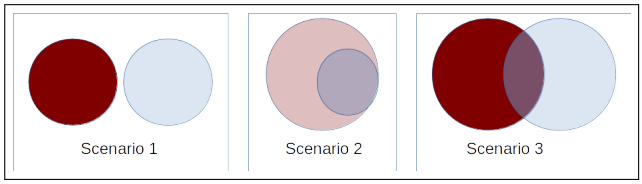

Fig. 1. Scenarios of intersection of BTS and VCC bug-related sets of data

The objectives of this work are to integrate the sets of bug IDs in the first and third scenarios: in both of them, the union of sets is larger than what held in the BT system data or the VCC system logs, therefore forming a larger set of bug IDs. We do so by using the Bicho tool to recover the BT system data, and the CVSAnalY tool to recover the VCC system logs.

After the creation of a larger data set of bug IDs, we propose to integrate the BT system data of the bug-related activities exclusively found in the VC system; and to integrate the VCC system logs of the bug IDs and activities found in the BT system data, but not reported in the VC system.

II. Concept

Observing the tables of Bicho and CVSAnalY and their attributes, we propose to use bug-related data in either database to fill the missing data as detected in the other database. Any 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 VCC system log with BT system data (and vice-versa) will require the use of meta-data contained in the 'scmlog' table (populated by CVSAnalY) to be copied in the 'issues' table (populated by Bicho). Figure 2 shows that attributes could be used from either table to fill the gaps in the other table.

III. The framework

In this section, the structure of the framework is discussed, comprising six modules: The Bug Tracking Issue System, Control Version System, Bicho, CVSAnalY, SCMLog and

[1]http://metricsgrimoire.github.io/Bicho/
[2]http://metricsgrimoire.github.io/CVSAnalY/

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

Fig. 2. Corresponding fields linked in Bicho and CVSAnalY

Issues. Figure 3 below depicts the components in a UML notation that will be instantiated of the final implementation. On the other hand, Figure 4 shows the architectural overview of the framework. The next subsections describe the main components, what has been achieved so far, and what is currently missing.

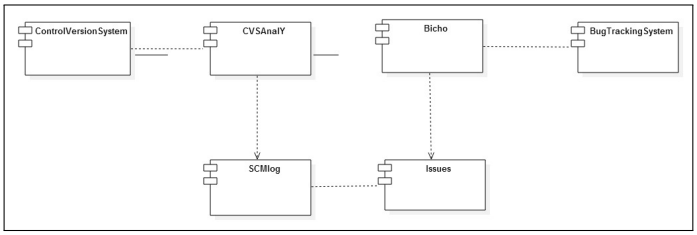

Fig. 3. UML Diagram of Components

A. Issues-Tracker Parser through Bicho

The Issues-Tracker Parser through *Bicho* component in figure 3 provides an interface in which the interaction between Bicho and any Bug Tracking System is defined. The interface that must be implemented by each client when mining data from issue trackers is the Issues interface. Thus, the interface would enable the interaction between *Bicho* and the supported *BT systems*. Some *BT systems* are currently supported by our framework: JIRA, Bugzilla, GitHub, SourceForge, Launchpad and Allura. Among these systems only GitHub requires the user to authenticate their identity using the logging credentials already registered on GitHub, before it allows any interaction or communication.

B. Development Log Parser through CVSAnalY

Th Development Log Parser through *CVSAnalY* component in figure 3 defines the interaction between CVSAnalY and any *VC systems*. In addition, the *SCMLog* component serves as the main entry point where development logs (VCC system log) are stored as extracted by *CVSAnalY*. In this way, the *SCMLog* interface must be implemented, in order to allow communication with any in figure 3. Currently, the framework supports the interaction with other VC system such as Git, CV system and Subversion.

However, one of the main obstacles among the supported VC system is that Git requires authentication by the clients or user before CVSAnalY can point to a repository in Git to extract and stored development logs. User-name and password need to be entered, in order to allow a communication between CVSAnalY and the Git (VC system). As a result, this paper implemented this framework only in its static interaction with Git: users need to first specify their logging credentials for authentication in GitHub in order to extract data by CVSAnalY from the remote VC system and stored development logs into a database generated by CVSAnalY.

IV. Implementation

We implement the majority of the tool chain in Perl programming language, whose strengths are text manipulation, portability, fast development capabilities and rapid development cycle [3]. In addition, Perl has an impressively broad range of standard library. However, Perl DBI package makes the automation and integration of databases easy.

In this research, we partially implement the SZZ algorithm [7] to trace bugs and logs within the OSS sample obtained from GitHub. In our formulation, we only look for bugs described by the '# + digit' regular expression (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.

The components discussed in section III were integrated in the tool-chain developed to search for bug IDs within the two databases, and combine the results into intersection and union of sets. In addition, not even to cross analyse VCC system logs and BT system data but to ultimately synchronise the VCC system logs and BT system data in Bicho and CVSAnaly respective databases.

In this section, we detail the steps and process of the implementation. These include retrieving the IDs from the two databases, combining the results into intersection and union of sets, synchronising the identified missing VCC system logs and BT system data into their respective databases automatically.

A. Retrieving VCC system Logs

Obtaining the VCC system Logs: the tool is capable of interfacing with, and execute *CVSAnalY* and *Bicho* tool set, in order to parse VCC system log and BT system data at once. *CVSAnalY* and *Bicho* automatically create databases and tables with meta-data, storing all the VCC system logs and BT system Data of the sample. Among the tables generated by *CVSAnalY*, we then specifically query the *scmlog* table. In the presence of a bug ID, the VC system logs also mentions 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".

B. Data Cleaning

Data Cleaning: False Positives and True Positives: The fourth step was the cleaning step, before isolating the bug numbers and IDs for both CVSAnalY and Bicho. The query

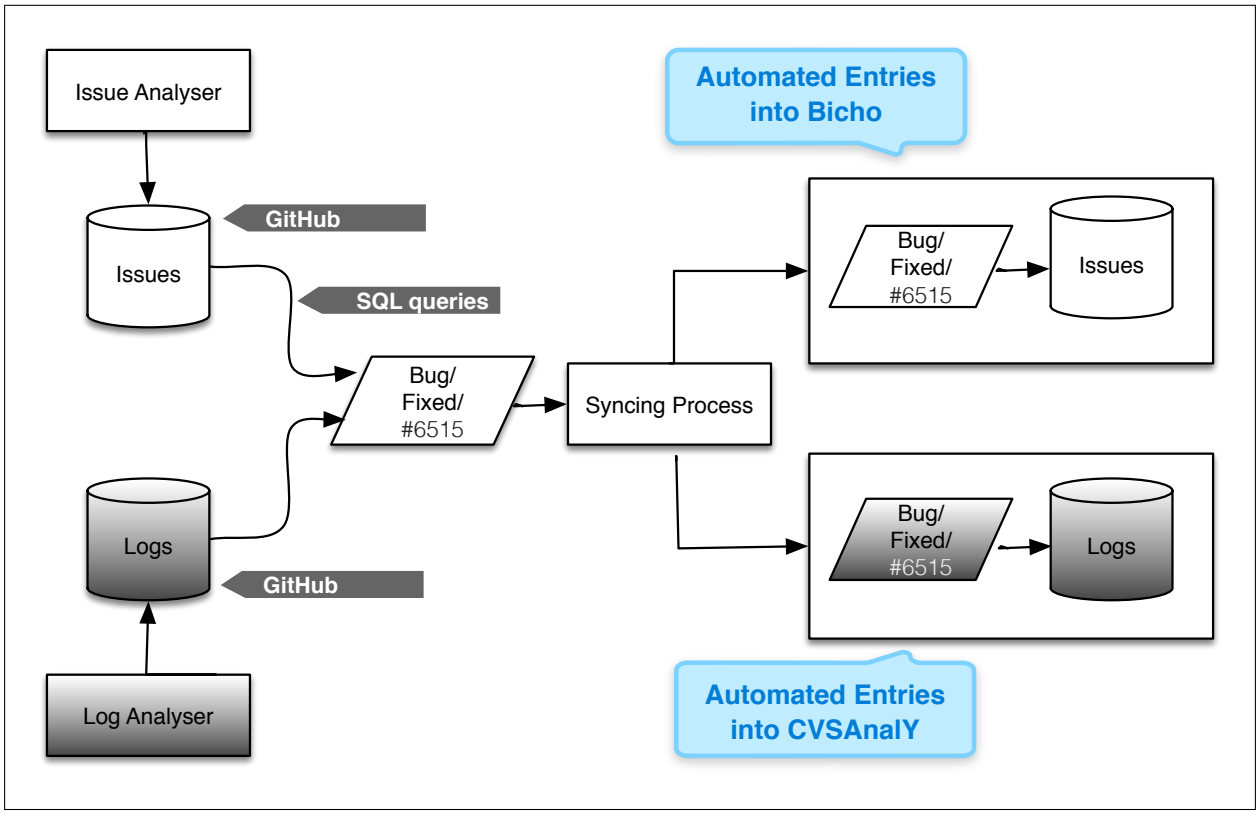

Fig. 4. Architectural Overview of the framework

for the '#' sign followed by numeric values in the version control commit logs imported with CVSAnalY produces a large number of false positives in most of the sample of 344 OSS projects we obtained from GitHub. In this case, the messages refer to the pattern searched for 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 using the SQL-query integrated into the tool that was developed for this research.

C. Isolating The Bug IDs

Isolating the bug numbers and IDs: After cleaning the data and remove all the trailing strings and white spaces, we composed two sets of bug IDs, one from the VCC system logs, and the other from the issue tracking systems. In the VCC system logs, we looked for the bug IDs in the free text descriptions left by developers (and stored in the "scmlog" table). In the bug tracking data, we used the bug IDs as assigned by the developers to the issues reported as bugs. These steps are performed within the developed tool, by querying the appropriate tables and cleansing the results.

D. Evaluation

In addition, we randomly pick few on the sample of 344 OSS projects obtained from GitHub and manually analysed each of the remaining bugs in Bicho and CVSAnalY databases, to make sure that each of the remaining IDs pointed to real bugs before evaluating the union and intersection of the sets. The bug IDs within the data set obtained through Bicho are always related to bug IDs.

Evaluating the union and intersection of the sets: the penultimate step was to evaluate the union and intersection of the sets, per project. Given a set of bug IDs mentioned in the *SCMlog* table, and the list of bug IDs stored from the issue trackers of a project, we evaluated the intersection (i.e., the common bug IDs) of these two sets, as well as the union of such sets (i.e., the overall set of unique bug IDs jointly held in the two databases). We then formulated a metric (named *Shared Bug Coverage*) to describe how many bug IDs are common in the two databases. Also this final step is integrated into the tool-chain.

V. Synchronisation

Synchronisation of BT system data and VCC system logs in both tools has been an ultimate goal in this research, merging BT system data and VCC system logs from different sources is a big challenge that require complex method [6].

We utilised the existing techniques and attempts to provide a solution to this problem(i.e., merging BTS data and VCC logs from various sources using Bicho and CVSAnalY). Thus, we can merge BTS data into VCC system logs of OSS projects leveraging on the existing techniques.

In this way, we will instantiate the main components we mentioned in section III above in the structure of the frame-

work. That is to say, **Issues** interface and **SCMlog** Interface to enable interaction or connection between Bicho and the supported BT system as well as CVSAnalY and the supported VC system such as GitHub.

The synchronisation was implemented successfully, BT system data not mirrored in *SCMlog* table in CVSAnalY database was synchronised into a newer *SCMlogcvsanaly* table in CVSAnalY created automatically. Similarly, VCC system logs not mirrored in Issues table in Bicho database was synchronised into a newer *Issuesbicho* also created automatically.

VI. Related Work

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

A framework to sync VCC system logs (development logs) and BT system data has been proposed and designed this is by using and improving existing framework developed in the past by [4] [2] [9] [1] [8] [1] [5] Which are all attempts to integrate and identified missing links between VCC system logs and BT system data. Thus, to provide researchers in empirical software engineering with a unified framework for integrating BT system and VC system for mining software repositories. In the same way our novelty, lies but with an attempt to synchronised either the missing VC system log or BT system data of software projects retrieved by these tools (Bicho and CVSAnalY) and stored into their respective databases.

The Buco reporter, developed by Ligu et al. [5], is an extensible framework that mirrors the VCC log and the BT system data, and the tool generates a complete set of evolutionary facts and metrics about a given OSS projects. Buco accurately traces VCC system logs and BT system data, but the tool was not designed and developed to synchronise the missing VCC system log and BT system data if discrepancies were found. The contribution of the presented research is a complete framework tool-chain) to synchronize the missing VCC system log and BT system data, supporting various repositories and bug tracing algorithms and approaches.

The Linkster tool involves a series of steps to retrieve, parse as well as convert and link the data sources [2]. As a result, it requires significant manual effort to analyze recovered links which might be much more accurate. On the other hand, RELINK [9] collects information automatically from the source code repository and BT system, builds the resulting information linked between VCC system logs or BT system data and output the identified links. In general, both these tools require a large amount of interaction but they recover missing VCC system logs and BT system data accurately. Our approach completes these tools by filling the missing VCC system logs and BT system data in either database in an automatic way.

VII. Conclusion and Future Work

Synchronisation of Bicho and CVSAnalY tool sets is suitable for analysing opens source software projects. Even OSS projects with a small number of commits and contributors, provide sufficient data for empirical software engineering research purpose. This automation and synchronisation of VCC system logs and BT system data evict the impediment of incomplete, inconsistent as well as skewed sets of data sets that researchers used for empirical studies.

The framework is easy to implements following the steps and process that involve in mining VCC system logs and BT system data. Although it is a daunting task and very challenging because the origin of such data sets is also not in sync. In this way, the synchronisation will enable flexible cross-analyses of evolutionary aspects of OSS projects since the tools can be able to mine VCC system logs and BT system data from via VC system and BT system. Also, the synchronisation provides a simple query-result mechanism and supports complex data queries for analysis.

The proposed tool-chain will eliminate and avoid repetitive activities in traceability tasks, as well as software maintenance and evolution. as well as provides a solution towards the automation and traceability of BT system data of software projects (in particular, OSS projects) using VCC system logs to complement and tract missing bug data.

References

[1] P. Anbalagan and M. Vouk. On mining data across software repositories. *2009 6th IEEE International Working Conference on Mining Software Repositories*, 2009.

[2] 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.

[3] T. Christiansen and N. Torkington. *Perl cookbook*. " O'Reilly Media, Inc.", 2003.

[4] 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.

[5] E. Ligu, T. Chaikalis, and A. Chatzigeorgiou. Buco reporter: Mining software and bug repositories. page 121, 2013. Retrieved January 23, 2015 from http://ceur-ws.org/Vol-1036/p121-Ligu.pdf.

[6] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. Herraiz. *Tools and Datasets for Mining Libre Software Repositories*, volume 1, pages 24–42. IGI Global, Information Resources Management Association. 701 East Chocolate Avenue, Hershey, PA 17033, Jan. 2011.

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

[8] 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.

[9] 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.