Brunel University London

Improving the quality of bug data in software repositories

A thesis submitted as partial fulfilment of the requirement of Doctor of Philosophy (Ph.D.)

by

BILYAMINU AUWAL ROMO

Computer Science Department

Brunel University London

Friday 15th April 2016

Abstract

Context: Researchers have increasingly recognised the benefit of mining software repositories to extract information. Thus, integrating a version control tool (VC tool) and bug tracking tool (BT tool) in mining software repositories as well as synchronising missing bug tracking data (BT data) and version control log (VC log) becomes of paramount importance, in order to improve the quality of bug data in software repositories. In this way, researchers can do good quality research for software project benefit especially in open source software projects where information is limited in distributed development. Thus, shared data to track the issues of the project are not common. BT data often appears not to be mirrored when considering what developers logged as their actions, resulting in reduced traceability of defects in the development logs (VC logs).

VC system (Version control system) data can be enhanced with data from bug tracking system (BT system), because VC logs reports about past software development activities. When these VC logs and BT data are used together, researchers can have a more complete picture of a bug's life cycle, evolution and maintenance. However, current BT system and VC systems provide insufficient support for cross-analysis of both VC logs and BT data for researchers in empirical software engineering research: prediction of software faults, software reliability, traceability, software quality, effort and cost estimation, bug prediction, and bug fixing.

Aims and objectives: The aim of the thesis is to design and implement a tool chain to support the integration of a VC tool and a BT tool, as well as to synchronise the missing VC logs and BT data of open-source software projects automatically. The syncing process, using Bicho (BT tool) and CVSAnalY (VC tool), will be demonstrated and evaluated on a sample of 344 open source software (OSS) projects.

Method: The tool chain was implemented and its performance evaluated semi-automatically. The SZZ algorithm approach was used to detect and trace BT data and VC logs. In its formulation, the algorithm looks for the terms "Bugs," or "Fixed" (case-insensitive) along with the '#' sign, that shows the ID of a bug in the VC system and BT system respectively. In addition, the SZZ algorithm was dissected in its formulation and precision and recall analysed for the use of “fix”, “bug” or “# + digit” (e.g., #1234), was detected when tracking possible bug IDs from the VC logs of the sample OSS projects.

Results: The results of this analysis indicate that use of “# + digit” (e.g., #1234) is more precise for bug traceability than the use of the “bug” and “fix” keywords. Such keywords are indeed present in the VC logs, but they are less useful when trying to connect the development actions with the bug traces – that is, their recall is high. Overall, the results indicate that VC log and BT data retrieved and stored by automatic tools can be tracked and recovered with better accuracy using only a part of the SZZ algorithm. In addition, the results indicate 80-95% of all the missing BT data and VC logs for the 344 OSS projects has been synchronised into Bicho and CVSAnalY database respectively.

Conclusion: The presented tool chain will eliminate and avoid repetitive activities in traceability tasks, as well as software maintenance and evolution. This thesis provides a solution towards the automation and traceability of BT data of software projects (in particular, OSS projects) using VC logs to complement and track missing bug data.

Synchronising involves completing the missing data of bug repositories with the logs detailing the actions of developers. Synchronising benefit various branches of empirical software engineering research: prediction of software faults, software reliability, traceability, software quality, effort and cost estimation, bug prediction, and bug fixing.

Contents

List of Figures

List of Tables

Acknowledgements

All praise and thanks are due to Almighty Allah, the Glorious and Exalted.
I am heartily thankful and indebted to my principal supervisor Dr. Andrea Capiluppi for his continuous support, guidance and encouragement at all stages of this research. He has been a tremendous mentor to me. His guidance and support helped me in many ways in this research and writing of this Thesis. Without his uncountable advice and support, none of this could have been possible. I would also like to extend my gratitude to my second supervisor Professor Tracy Hall for her insightful comments and advice throughout this research.

I'd like to thank my parents, Auwal Sulieman Romo and Hafsat Auwal Romo. They've instilled in me the appropriate level of curiosity and passion for learning that keeps me writing and researching late into the night. The debt is too large to pay in one lifetime.

I wholeheartedly thank my cherished wife, Zainab Abubukar Jaredi, and my stepmothers Suwaiba Usman Maitambari and Zara'u Jelani Marna—they are the greatest pride and joy of my life. I will always be grateful for their love, patience, and understanding. The incessant curiosity of my family has been a continuous inspiration for my studies on learning. A huge acknowledgment is due to my parents-in-law, Abubakar Sahabi Jaredi and Hauwa'u Sahabi Jaredi; my uncle, Lawali Halidu Romo; my sisters and brothers; and my best friends, Mustapha Abdullahi Sifawa, Hassan Sahabi Isah Gada, Adamu Ibrahim Ruwa-wuri and Fodio Aliyu Wamakko for their continuous encouragement and unconditional support.

I'd be remiss if I didn't acknowledge the manifold support of Drs. Aaron Khan and Andres Baravalle as well as Mal. Hassan Gummi, Mal. Aliyu Dikko Tambuwal and Mal. Yahaya Shehu; they taught me a lot about Computer Science in general. My thanks also are due to Brunel Software Engineering Lab Members, and many other amazing people at Brunel University London and University of East London, whose zeal for studying the way that our brain works trickled down to folks like me and instigated my work on Software Maintenance and Evolution.

To my past and present colleagues, Muhammad Abdullahi Salame, Ali M Said, Nadia Abubakar, Ahmed, Muhammad Al-mualla, Najeeb AA Gambo and Nasiru Daniya, and many others, who encouraged and supported me in many ways: there are only a few lines that I can write here but my gratitude is much larger than that.

Dedication

This Thesis is dedicated with love to my late great-grandfather, Mal. Ahmad Muhammad (Roro Romo) and to Sokoto—the land and the people.

Chapter 1

Introduction

Researchers in empirical software engineering have been mining software repositories for a long time [114]. Version control system logs (i.e, VC log) [84], defect tracking systems [95], mailing lists [18], [100] and the documentation of software artefacts [69] all provide a rich set of data that can be used to understand the inner mechanisms of producing software. The field of mining software repositories (MSR) is similar (but not limited) to the fields of data mining and knowledge discovery: the principle of this field is that empirical investigations of repositories will create new research avenues in software processes and products, including software maintenance and evolution. MSR studies in the past have helped to support the maintenance of software projects, for instance by proposing and validating techniques and novel ideas to help developers report and understand the evolution of software systems [141]; [70]; [48].

Practitioners and researchers in the MSR field have increasingly recognised the benefit of mining software repositories to extract information and have observed an exponential growth. The source of data most commonly used by researchers and practitioners is, by far, VC logs [5]. The VC logs contain the actions (adding, deleting or modification of files/codes) performed by developers throughout the life cycle of a software project: a subset of empirical studies ([43];[42]; [104]; [55]) have been pivotal in understanding how such repositories are mined effectively, and what evidence should be provided by researchers on a software project by mining such activity logs.

Many researchers has recognised the importance of mining VC logs, with the objective, among others, to support software corrective maintenance activities [19], or to improve software design and component reuse [25], [38], [9]. On the other hand, the analysis of VC logs has highlighted the most relevant artefacts that can help software developers in the open-source software (OSS) community to understand and participate in software development projects [40]. For instance, new software developers could contribute to bug-fixing processes and add new feature enhancements to a software project that are relevant to their past expertise [108].

Another commonly mined source of data is bug tracking system (BT system) data. BT data contains information reported by users and developers in the OSS community [104]. However, BT data can be used to design models for predicting software faults and software reliability [45].

Nonetheless, a subset of studies ([41], [137], [5], [105], [126], [139], [67], [36], [34]) have been crucial in shedding light on how BT data can be mined and provide empirical evidence regarding software defects that could be revealed from such data. In addition, BT data has been used to design models for predicting software faults, software artefact can also link to when, how and by whom changes have been made to it [134].

The context of this work is the traceability of software bugs, which is the linking of the software artefact from VC logs to BT data (and vice versa) produced during the development and maintenance cycle of a software system. In simpler terms, traceability of a software bug is the establishment of links between bugs and the changes that software developers made to fix the bug. For this purpose, it is fundamental (i) to provide more efficient automatisation of the traceability of the bug information (i.e., from the BT data) as reflected by the actions of developers (i.e., from the VC log); and (ii) to make use of both VC log and BT data to provide a better understanding of the bug-related activities in software projects.

This chapter is structured as follows. In section 2, the problem statement that underlines the whole thesis is defined; in section 3, the objectives of this thesis are defined, including the appropriate reasoning behind each objective; and in section 4, the contributions of the thesis are highlighted, with respect to the current situation in this field. Section 5 highlights the beneficiaries of this thesis. Finally, section 6 illustrates the contents (structure) of the thesis, and how each chapter is used to build the arguments, or verify the objectives.

1.1 Problem statement

This section illustrates the problem that the thesis aims to solve (or the “why”) and articulated into various chapters of the thesis. These parts reflect the issues that researchers face when attempting to trace bugs in the VC log, and how they need to reconcile the discrepancies between the VC log and the BT data (and vice versa). The steps are as follows:

1. How to select the **appropriate tools** to mine VC logs and BT data.
2. How to **identify** bugs (BT data) in VC logs.
3. How to detect the **discrepancies** (missing data, redundancies and so on) between VC logs and BT data.
4. How to **synchronise** redundant or missing data from one data source by using the traces found in the other source, and vice versa.
5. How to produce a **tool chain** to automatically detect, synchronise and re-engineer missing data and discrepancies in VC logs and BT data.

In the next subsections, the above-mentioned problems are analysed in more detail (i.e., each section looks at “How” to solve the problems).

1.1.1 MSR tools for VC log and BT data

When investigating tools to mine VC log and BT data, there are several tools that are publicly available, but also others that are not publicly available (i.e., commercial tools) for mining VC logs and BT data.

Some of the tools that are publicly available include Linkster [17], ReLink [130] and BuCo Reporter [83]. These tools will be examined in detail in Chapter 2.4.

Tools that are not publicly available include Repoguard [80], SoftChange by [46][1] and Rational ClearCase[2] (these tools are also discussed in detail in 2.6). From the perspective of repeating this research, one would need to use a publicly available tool set.

The second element required to achieve the objectives of this thesis is the integration of the tools. It is important to select a VC system tool and a BT system tool and integrate their functionality in an automated way. Since the tools are designed and their input and output is developed and run independently, the integration of various sources of data, and their input and output, becomes fundamental. In this way, certain criteria have been set in terms of the similarity between the entities and the components stored by the tool set (discussed in Chapter 3.7). Similarly, other features need to be considered, such as ensuring that the tools can support multiple sources of data and software repositories.

1.1.2 How to identify bugs (BT data) in VC logs

Identifying bugs in VC logs requires a large amount of time in software bug traceability [79], and the cost of identifying and fixing bugs has been reported is high in the commercial project [54].

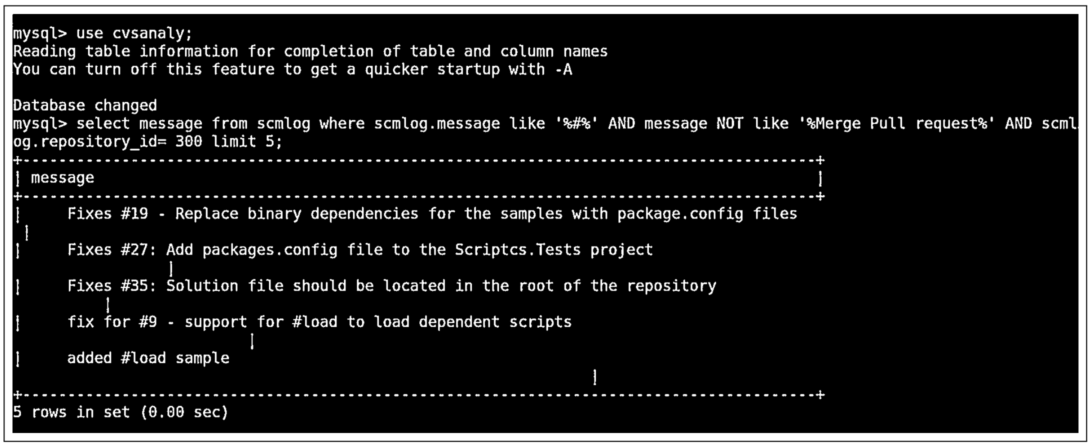
Similarly, developers in the OSS community might spend considerable effort on locating bugs in VC log. After the bugs are identified, a link should be established between the bug associated with the VC log. As a result, researchers have to devise their techniques in how to look for bugs in VC logs effectively when dealing with software bug traceability. Ideally, the aim would be to reduce the cost and time to identify or locate bugs in VC logs.

At present, there are techniques that researchers and practitioners have used to identify bugs in VC logs, from other fields. They include information retrieval (IR) methods [130], machine learning and heuristic-based approaches using sequential-pattern mining [68], [56], [36]. The technique that is most commonly used by software engineering communities is the SZZ algorithm, which looks for patterns such as “#+digit (e.g., #1234)”, “fixed” and “bug” keywords [116], [18]. There are some pitfalls in any algorithm or technique when identifying

[1]Available only on request
[2]`http://www-03.ibm.com/software/products/en/clearcase`

bugs in VC logs. An example of such a pitfall could be deciding the right keywords and how much precision and recall have an effect [56], [97]. The study of Casalnuovo et al [26] demonstrates that researchers in software engineering have difficulty in deciding the right keywords for identifying bugs in VC log. As a result, the researchers have applied nine keywords in retrieving project evolution history related to VC logs: “error”, “bug”, “fix”, “issue”, “mistake”, “incorrect”, “fault”, “defect” and “flaw”. Previous studies have used mainly with two keywords – “bug” and “fix” – to identify bugs in VC logs.

Figure 1.1 depicts use of the keyword “# +digit”. Figure 1.2 depicts a real example of VC logs retrieved using the regular expression “fixed”, and 1.3 depicts a real example of VC logs retrieved using the regular keyword expression “bug”.

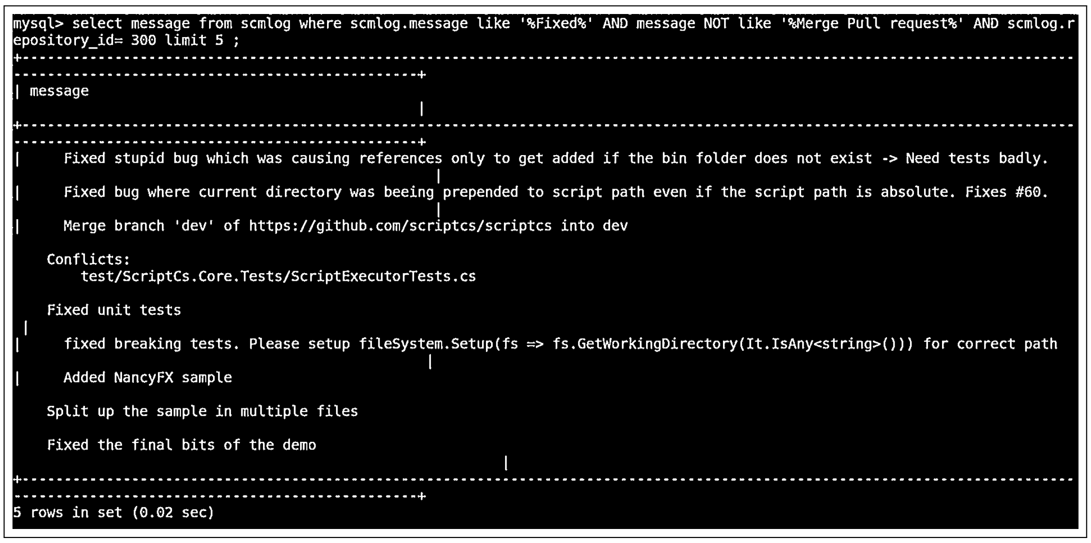


```
mysql> use cvsanaly;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select message from scmlog where scmlog.message like '%#%' AND message NOT like '%Merge Pull request%' AND scml
og.repository_id= 300 limit 5;
+-----------------------------------------------------------------------------------------+
| message                                                                                 |
+-----------------------------------------------------------------------------------------+
|     Fixes #19 - Replace binary dependencies for the samples with package.config files
 |
|     Fixes #27: Add packages.config file to the Scriptcs.Tests project
                 |
|     Fixes #35: Solution file should be located in the root of the repository
          |
|     fix for #9 - support for #load to load dependent scripts
                          |
|     added #load sample
                                                                |
+-----------------------------------------------------------------------------------------+
5 rows in set (0.00 sec)
```

Figure 1.1: Bug data in VC logs using “# + digit” keyword

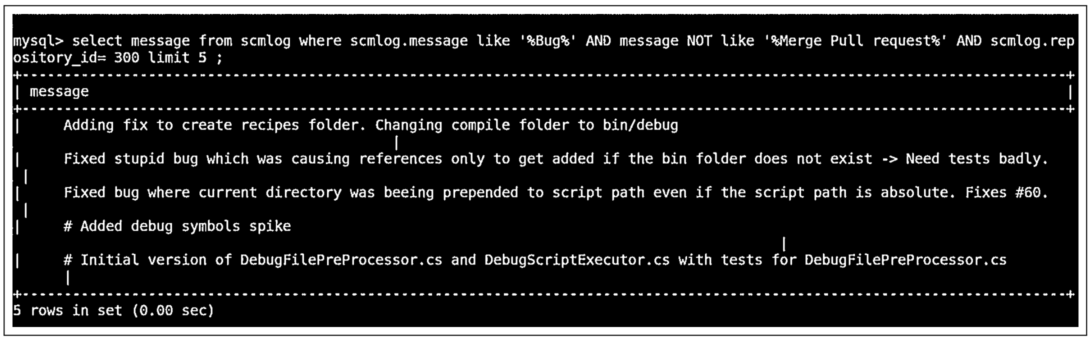
1.1.3 How to find discrepancies between VC logs and BT data

The integration of different tools for software development is challenging when the tools, which should track complementary artefacts, lack consistency in the recording of events [35]. For instance, Robles et al [106] stated that “correlating BT data and VC logs is a big challenge that requires complex methods”. This shows that when bugs are discovered, developers should report their existence in the VC logs when they fix them. Also, they should open the appropriate procedure in the issue tracker, marking it as “open”; similarly, when a bug has been



```
mysql> select message from scmlog where scmlog.message like '%Fixed%' AND message NOT like '%Merge Pull request%' AND scmlog.r
epository_id= 300 limit 5 ;
+-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------+
| message
                                                |
+-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------+
|     Fixed stupid bug which was causing references only to get added if the bin folder does not exist -> Need tests badly.
                                                  |
|     Fixed bug where current directory was beeing prepended to script path even if the script path is absolute. Fixes #60.
                                                  |
|     Merge branch 'dev' of https://github.com/scriptcs/scriptcs into dev

    Conflicts:
        test/ScriptCs.Core.Tests/ScriptExecutorTests.cs

    Fixed unit tests
 |
|     fixed breaking tests. Please setup fileSystem.Setup(fs => fs.GetWorkingDirectory(It.IsAny<string>())) for correct path
                                                 |
|     Added NancyFX sample

    Split up the sample in multiple files

    Fixed the final bits of the demo
                                                          |
+-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------+
5 rows in set (0.02 sec)
```

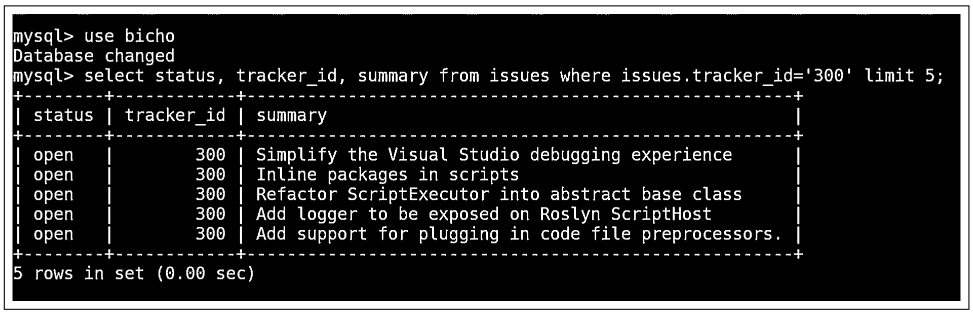
Figure 1.2: Bug data in VC logs using “fixed” keyword



```
mysql> select message from scmlog where scmlog.message like '%Bug%' AND message NOT like '%Merge Pull request%' AND scmlog.rep
ository_id= 300 limit 5 ;
+-----------------------------------------------------------------------------------------------------------------------------+
| message                                                                                                                     |
+-----------------------------------------------------------------------------------------------------------------------------+
|     Adding fix to create recipes folder. Changing compile folder to bin/debug
                                             |
|     Fixed stupid bug which was causing references only to get added if the bin folder does not exist -> Need tests badly.
 |
|     Fixed bug where current directory was beeing prepended to script path even if the script path is absolute. Fixes #60.
 |
|     # Added debug symbols spike
                                                                                           |
|     # Initial version of DebugFilePreProcessor.cs and DebugScriptExecutor.cs with tests for DebugFilePreProcessor.cs
      |
+-----------------------------------------------------------------------------------------------------------------------------+
5 rows in set (0.00 sec)
```

Figure 1.3: Bug data in VC logs using “bug” keyword

fixed, the developers should mention its “fixed” status in the VC logs and mark it as “closed” in the BT data. Figure 1.4 shows a typical example of a bug marked as “open” in BT data.



```
mysql> use bicho
Database changed
mysql> select status, tracker_id, summary from issues where issues.tracker_id='300' limit 5;
+--------+------------+---------------------------------------------------+
| status | tracker_id | summary                                           |
+--------+------------+---------------------------------------------------+
| open   |        300 | Simplify the Visual Studio debugging experience   |
| open   |        300 | Inline packages in scripts                        |
| open   |        300 | Refactor ScriptExecutor into abstract base class  |
| open   |        300 | Add logger to be exposed on Roslyn ScriptHost     |
| open   |        300 | Add support for plugging in code file preprocessors. |
+--------+------------+---------------------------------------------------+
5 rows in set (0.00 sec)
```

Figure 1.4: BT data marked as “Open”

Past research has established the fact that there is inconsistency in how bugs are reported when tracing the issue tracker, and VC logs of a software project [17], which means that bugs in BT systems do not appear in VC systems. Also, it was asserted by [88] and the study of [12] of OSS repositories that studies with few OSS projects “were **not enough** to make any statistical conclusion”. The challenge is to produce empirical evidence on many OSS projects. Since VC logs and BT data are not typically mirrored and synchronised, below are the potential problems when mining VC logs and BT data in empirical studies:

1. **Incomplete data.** This could be a case of the BT system not having all the data, or it could be that the VC logs have incomplete data. This can lead to a biased or non-trustable analysis in empirical software engineering research [112].

2. **Inconsistent and a skewed set of data**. There could be serious consequences when automated algorithms consider only certain VC logs and BT data. Any model designed and produced using an inconsistent and skewed set of data might be severely biased [10]. Therefore, automatisation and completion of missing data is of critical importance in order to avoid biased research [53]. In reality, VC logs should form a superset of all BT data: one would expect the data contained inside BT system to be mirrored in the VC system and developers to record and distinguish between their development and bug-fixing actions.

This thesis is based on the existing tools and techniques found in the literature proposed by [3, 5, 17, 34, 36, 41, 67, 80, 83, 104, 105, 121, 126, 130, 137]. Their past studies report on the traceability links and recovery not to synchronise the recovered links between VC logs and BT data into their respective databases. In this research we want to ensure that the VC logs and BT data that we use in empirical software engineering research are as complete as possible.

Finally, cross-analysing and linking VC logs and BT data can improve the quality of data that we use in empirical software engineering research [74]. Similarly, improving the data accuracy we obtained from OSS projects will provide practitioners in software engineering with more complete data sets [31].

1.1.4 How to synchronise VC logs and BT data

This section looks at how to integrate the tool set and sync the VC logs and BT data when data is missing from one or the other. In other words, when bug data is missing from the BT system, how can we recover BT data using the VC logs information available? Conversely, when VC logs information is missing in the VC system, how can we recover VC logs using the BT data available?

The novelty of the tool chain that needs to be designed and implemented in this thesis, apart from the fact that it supports various OSS software repositories, is that it is able to synchronise missing VC logs (concerning bugs) with data extracted from the BT system, and vice versa. Thus, such a tool chain can assist in mining the complete set of software evolutionary data throughout the entire life cycle of software projects, as well as provide complete VC logs and BT data for posterior analysis.

Figure 1.5 depicts a Venn diagram that shows the current state of BT data and VC log information. Only a small subset of VC log and BT data are present and in sync [109].

Now the idea is to utilise the existing techniques and related publicly available tools in order to design and implement a tool chain not only for "extracting", but also for automatically "syncing" VC logs and BT data, which supports multiple BT system and VC systems. This is because current BT system and VC system databases were designed and structured to extract

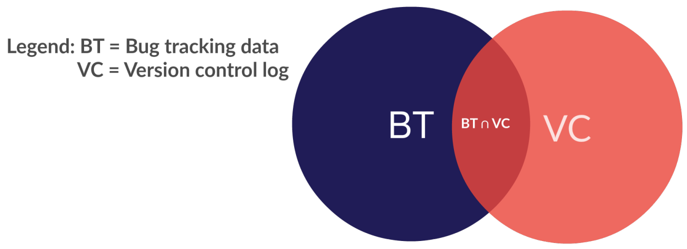


Figure 1.5: Current state of VC log and BT data

and store VC logs and BT data in different tables that exist in different databases [109]. As a result, a newer table might be created (by not intervening in the existing structure of the databases to avoid duplication) in both databases, where all the missing VC logs and BT data are identified and extracted and can be synchronised in an automatic way.

1.1.5 A tool chain for automated detection and syncing of bug-related discrepancies

In this thesis, an approach and tool chain has been introduced for synchronising the VC logs and BT data to form a more complete data set that supports researchers in software engineering, particularly in software maintenance and evolution.

In order to automatically synchronise and detect VC logs and BT data in this thesis, the tool chain will provide an interface that enables researchers in software engineering to cross-analyse and link BT data and VC log data automatically. In this case, the BT tool (Bicho) and VC tool (CVSAnalY) will be re-engineered and implement the tool chain by realigning relevant entities that exist between the Bicho and CVSAnalY tool sets. Thus, automated entries in both tools can occur simultaneously, and common links between VC logs and BT data can be detected automatically.

1.2 Research objectives

Researchers in the past have already studied and established the fact that links between BT data and VC logs are missing and can be recovered [12]. However, improving the quality of bug data in software repository has not been adequately investigated so far. In addition, investigating the techniques and approaches used to extract bugs from the VC logs using regular expressions has been investigated with a large sample of OSS projects in this thesis. Following the parts that compose the problem statement, the **objectives** of this thesis are as follows:

Obj1 [**Tools**] To discover what tools researchers use in mining VC logs and BT data and identify the tools available, as well as describe their data structure.

- **Rationale:** The rationale is to provide state-of-the-art VC tool and BT tool sets. The structures have to be mapped and linked correctly with consistent data formatting between VC logs and BT data in their respective databases.

Obj2 [**Bugs into VC log**] To identify bugs in VC logs, I want to dissect the SZZ algorithm (regular expression) in its basic components and analyse their respective precision and recall. Namely, I want to examine the use of "fix", "bug" and "# + digit" to trace bug IDs from the VC logs of OSS projects stored in both tools.

- **Rationale:** The objective is to dissect the SZZ algorithm in its three main components, or the basic blocks that can be used to identify the presence of bug-fixing commits. These three components will be used in this thesis to isolate all the bug IDs as found in the VC 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".
- **Hypothesis:** Using "# + digit" when tracing bug IDs in VC logs produces higher precision than using the keywords "bug" or "fix".

Obj3 [**Discrepancies between BT data and VC logs**] Use the SZZ technique to analyse BT data and VC logs on a large data set to provide a quantitative study of the traceability issues of 344 OSS projects to quantify the number of BT data and VC logs missing.

- **Rationale**: By identifying bugs in VC logs and BT data, I can establish evidence of the feasibility in traceability link recovery and cross-analyse the VC logs and BT data of 344 OSS projects from open-source software repositories.

- **Hypothesis**: Bugs in VC logs and BT data of 344 OSS projects sampled from GitHub[3] are not mirrored. Most of the OSS projects have an issue of bug traceability[108].

Obj4 [**Synchronisation**] To reconcile and sync the VC logs and BT data of large OSS projects.

- **Rationale:** To isolate database fields that can be used to fill the gaps in one source or the other.

Obj5 [**Tool chain**] To present and propose a tool chain that track and synchronise VC logs and BT data from 344 OSS projects.

- The **rationale** of such a tool chain is to assist in link recovery and the synchronisation of BT data and VC logs. In addition, the goal will be to provide a framework to support multiple BT systems and VC systems in mining software repository links.

1.3 Contributions of this thesis

The contributions of this thesis are as follows:

C1 – Tools. In this thesis, tools that trace **VC logs** and **BT data** for software projects were identified. After selecting Bicho and CVSAnalY to use in this research, this thesis described VC logs and BT data structures of the tools selected. It also identified the fields that linked BT data and VC logs for synchronisation into their respective databases. Thus, researchers in software engineering can trace, mine VC logs and BT data using the identified tools collectively without mining and tracking VC logs and BT data independently.

[3]http://github.com

C2 – Bugs in VC log. The thesis presented an in-depth analysis of VC logs using the SZZ algorithm, which has been used extensively by researchers to identify bugs in VC logs and BT data of software systems. In this thesis, the SZZ algorithm was partitioned in its three core components – the "bug" and "fix" keywords and "# +digit" – with a manual check-up. In addition, this thesis evaluated the precision and recall of the various parts of the SZZ algorithm and presented the precision and recall of each element in detecting bug identifiers in the development logs (VC logs). This thesis suggested using "# + digit" and the bug ID, which largely outperformed the other proxies in finding bugs in VC logs and BT data.

C3 – Discrepancies between BT data and VC logs. This thesis presented the results in a Venn diagram, which suggested that around 1/3 of the total number of VC logs and BT data were mirrored when cross-analysed and linked with BT data. Also, another 1/3 were only present in BT data retrieved by Bicho, while the rest were found in VC log data (CVSAnalY), but never summarised into BT data retrieved by a BT system tool (Bicho). This thesis presented and conducted a large empirical study that mined the VC logs and BT data of 344 OSS projects, hosted on GitHub[4]. Thus, this thesis provided a large and significant statistical conclusion with reasonable evidence in the issue of traceability links recovery and syncing of VC log and BT data from open-source software repositories.

C4 – Synchronisation. The thesis presented a tool chain that synchronised VC logs and BT data, ensuring that data sets held by these tools (Bicho and CVSAnalY) are always complete and enriched effectively. Most importantly: (i) the tool chain avoids the impediment of using incomplete data sets for analysis in empirical software engineering; (ii) VC log and BT data can be identified and retrieved with higher precision; and (iii) consistent and unskewed data sets can be obtained, since the missing information in both tools is tracked and synchronised.

C5 – Tool chain. This thesis proposed and implemented a complete tool chain not only for

[4]https://github.com/

extracting, but also for automatically syncing VC logs (development logs) and bugs of issue data (BT data) – that is, supporting multiple BT system and VC system.

The novelty of the tool chain, apart from the fact that it supports various OSS repositories, is its ability to synchronise missing VC logs (concerning bugs) with data extracted from the BT system, and vice versa. Finally, this tool chain was made available.

1.4 Beneficiaries and impact of this thesis

1. **Open-source software (OSS) community** This thesis benefits OSS community that aim to design and develop tools for retrieving VC logs and BT data collectively that (i) support various BT system and VC system sources; (ii) allow cross-analysis of BT data and VC logs; and (iii) track and synchronise missing BT data and VC logs of software projects, ensuring that complete and consistent data sets are always stored in the database for posterior analysis.

2. **Researchers in software corrective maintenance:** Researchers in software maintenance and evolution benefit from this thesis, since the source of data most commonly used by researchers in software corrective maintenance is, by far, VC logs and BT data. Using the tool chain to extract data from various sources will help researcher by improving the quality of the data sets they used. Similarly, researchers can extract complete VC logs and BT data from various sources, and also understand the inner mechanisms of producing software artefacts that are required for research and analysis in software engineering.

3. **Researchers in empirical software engineering**: The novelty of the tool chain, apart from the fact that it supports various OSS software repositories, is its ability to synchronise missing development logs (concerning bugs) with data extracted from the BT system, and vice versa. As a result, researchers of bugs in empirical software engineering benefit from this thesis by using the tool chain in mining complete sets of evolutionary facts to provide an unbiased data set.

In general, both large OSS and commercial projects can be analysed in order to extract and establish missing links and sync BT data with VC logs (and vice versa) for posterior analysis.

1.5 Structure of the thesis

This chapter introduces the road map for the thesis and the contribution to knowledge.

Chapter 2 presents the state of the art of the existing tools and techniques in the literature. Also, Chapter 2 Section 2.4 present the context of this thesis and discuss the literature related to mining software repositories, software maintenance and evolution, and open-source software.

The approach (methodology) is discussed in Chapter 3.

Chapter 5 presents an empirical study. The results of the empirical study in this thesis, carried out with over 300 OSS projects sampled from GitHub, are presented in Chapter 5.3. The results suggest BT data are not mirrored when compared to the VC logs of the same OSS project.

This thesis presents the SZZ replication with large OSS projects (344) using an improved approach in Chapter 4. Chapter 6 discusses the automating and filling the missing data of 344 OSS projects sampled in this research. Chapter 6.3 details the structure and implementation of the framework proposed by this thesis and describes how to sync and cross-analyse the two sets of bug-related data (VC logs and BT data) and vice versa.

The threat to validity based on our finding in every chapter of this thesis in general are discussed in Chapter 7.5. Finally, the thesis concludes in Chapter 7.7.

Appendix A in Section A.1 presents the tool chain (codes) developed and used in the empirical study reported in this thesis. Table 1.1 summarises the thesis structure, as follows:

Figure 1.6 depicts an overview of this thesis, which will be discussed in more detail in Chapter 6.3. The left-hand side of this figure (as highlighted) is firmly established in the data found in the open repositories, among the ones described in Chapter 2 Section 2.2 of this thesis.

Below is a list of the publications based on this thesis:

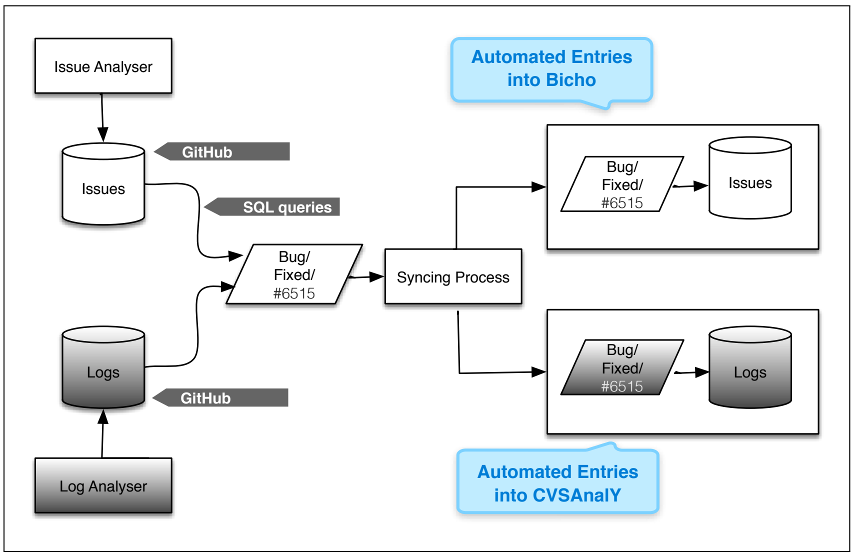


Figure 1.6: Architectural overview of the framework

Chapter	Title	Bibliography
2	State of the Art	[109]
3	Methodology	[108]
4	Locating bugs in VC Logs	[108]
5	Discrepancies between the bug sets from VC logs and BT data	[109] [108]
6	Automating and synchronising the missing data	[109] [108]

Table 1.1: Our publications related to the chapters of this thesis.

1. Romo, B. A. and Capiluppi, A. 2015. 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 (Nanjing, China, April 27 - 29, 2015). EASE '15. ACM, New York, NY, 1-6. DOI= http://doi.acm.org/10.1145/2745802.2745833

2. Romo, B. A., Capiluppi, A., and Hall, T. 2014. Filling the Gaps of Development Logs and Bug Issue Data. In Proceedings of the international Symposium on Open Collaboration (Berlin, Germany, August 27 - 29, 2014). OpenSym '14. ACM, New York, NY, 1-4. DOI= http://doi.acm.org/10.1145/2641580.2641592

Chapter 2

State of the art

In this chapter, we discuss definitions and terms mentioned in this thesis, and we discuss the different types of bugs and their categories. Also, we discuss the state of the art in extracting data from the commit logs and bug tracking issue trackers, and their related tools and techniques.

2.1 Introduction

The "software maintenance" and "software evolution" research fields have become very active and well respected within software engineering research, and the terms software evolution and software maintenance are often used as synonyms. For instance, the International Standards Organisation [63] and [21] emphasise the importance and need for pre-delivery aspects of software maintenance as well as the post- delivery stage (i.e., its evolution).

The IEEE 1219 Standard for Software Maintenance [1] defines software maintenance as "the modification of a software product after delivery to correct software failure. It will improve performance or other attributes or to adapt the product to a modified environment."

On the other hand, software evolution, in general, implies that something in the code base has changed for the better. The Merriam-Webster Dictionary defines evolution as "a process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state". Thus, it captures our intuitive concepts about something that is improving.

The main driving factor of the first conference on software engineering (organised in 1968 by the NATO Science Committee) was to establish and use “sound engineering principles to obtain reliable, efficient and economically viable software”. Similarly, it was mentioned that software maintenance is among the activities of software engineering, and considered as a post-production activity – that is to say, after delivery and deployment of a software project.

Referring to the early definitions of software maintenance, Royce in 1970 shared and proposed the well-known *waterfall life-cycle* process for software development. This process model was inspired as a result of established engineering principles, which include a maintenance phase (but not an evolution one) as the final phase in the life cycle of a software system [110].

The ISO [63] also proposed the following categories for software maintenance:

1. **Corrective** maintenance is the modification of a software project after delivery (at the post-delivery stage) to correct identified faults.

2. **Preventive** maintenance is the modification of software to prevent future faults (at the post-delivery stage).

3. **Adaptive** maintenance is the modification of software projects after delivery (at post/pre-delivery stage) to keep the software system functioning in a different software platform.

4. **Perfective** maintenance is the modification made to software projects to maintain and improve the software perfectiveness and quality (at the post-delivery stage).

The term “software evolution” attracted renewed attention in the 1990s following the classic and insightful work of [81] as well as [82]. In these studies, software evolution has been accepted as an area of research worth studying and an area that poses serious problems and challenges to software projects.

Software evolution has been studied for the past 60 years, and has become an even more prominent area of study since the pioneering works by Lehman and Belady. In most research on software evolution, there is an awareness of the rapidly increasing importance and impact of software projects in many activities of society. In the 1990s, the term software evolution gained widespread acceptance, and the research on software evolution started to become popular [8], [96].

Evolutionary processes such as evolution development [47], the *spiral model* [20] and the *stage model* [13] have shed additional light on how systems evolve and their dynamics.

This research will improve the tool sets that support the maintenance of software projects in the pre-delivery and post-delivery stages of software development [21]. The focus of this work is intended to be specific, because the tools generally used in mining bug-related data, for software maintenance and evolution activities, are not always producing exactly the same sets of bug data (i.e., they are not *in sync*) [109]. In addition, this thesis also focus on linking bug reports to code changes and vice-versa, so that better quality bug data can be mined for researchers that hopefully will benefit practitioners. However, the data stored by BT tool and VC tool could be crucial for research in cost estimation, software quality and fault prediction techniques: therefore, a better understanding of how this data collection could be better achieved is of paramount importance.

The fields of software maintenance and evolution have received renewed attention due to the availability of open-software repositories that allow researchers to mine data to construct models and techniques. In the next subsection, we present the different types of repositories, how they became available to practitioners, and what could be mined by researchers.

2.2 Types of software repositories

Mining software repositories, to extract process and product data, is now considered as a research field, and the term *mining software repositories* (MSR) has been defined to describe a broad class of analyses dealing with the examination of software repositories. In this research, software repositories, in general, are storage websites that hold several of the artefacts designed, produced and archived during software evolution [132] [67] [56] [48] [133] [131].

The most commonly used software repositories allow OSS projects to store the information about bugs and relative to the VC logs. Systems such as Concurrent Versions System, Subversion (SVN) and Git are being commonly used by developers to maintain a log of all the activities done by developers on a software system.

On the other hand, repositories to hold bug data, their date of inception, their resolution date (if any) and so on have long since been made available as issues/bug-tracking systems. Examples of such BT systems include Bugzilla[1], SourceForge[2] and JIRA[3]. Communication archives (e.g., e-mail or mailing lists) have also been used in past research [77], mostly as a means of triangulating other data, for instance to determine the list of authors and developers discussing the issues around code production [101].

Based on their characteristics and uses, the following are the categories of software repositories:

1. Historical repository: This type of repository records the evolutionary facts concerning the progress of a particular software project. Source control repositories and bug repositories are examples of this type of repository. Software repositories are quite often used in software engineering as a way of storing and keeping a record of an open-source software project. For instance, historical repositories are used to track the history of a bug or changes made to a software artefact.

2. Runtime repository: Involves VC logs that contain information about the execution and usage of a software system in both single and multiple deployment sources.

3. Code repository: Host several open-source software projects – for example, Sourceforge.net, Google Code and Codeplex.com – including their source code.

The information stored in these repositories provides software practitioners and researchers with the ability to mine source code, bug-related data and the VC logs. In this thesis, we consider all the repositories mentioned above to mirror important and interesting software artefacts [43].

Information that exists in these repositories remains throughout the entire stages of the software project. This information might represent thousands of versions with years of facts

[1]https://www.bugzilla.org
[2]http://sourceforge.net
[3]https://www.atlassian.com

about the development process of a software system. Software engineering researchers have devised and investigated a wide range of approaches to extract relevant information and revealed relationships and trends from repositories in the context of software evolution [67].

2.3 Definitions and terms

In this subsection we report some definitions adapted from the IEEE Standard Classification for Software Anomalies [2], including *defect, error* and *fault.* The use of these terms differs with respect to each organisation or software project. In this case, the approach is to use the terms bug, defect and errors with respect to a particular definition or term adopted and used in a given software project and organisation.

The definition of a VC log is added to these basic definitions.

- Defect: A defect is an imperfection found in a system where the system does not conforms to its specifications and need be corrected or replaced. (adapted from IEEE Standard 1044-2009 [2])
- Error: Error is a human action that result to incorrect functions defined as per the software requirement specification document and the actual product. (adapted from IEEE Standard 1044-2009 [2])
- Bug: A bug is an error found before the system is delivered to the client. In some cases the terms *bug* and *defect* are used interchangeably to refer to an error found and reported before and after a system has been designed, developed and delivered to the client.
- Fault: A fault is an error that causes a failure to the system which might terminate the whole functionality of a system. (adapted from IEEE Standard 1044-2009 [2]) Faults occur as a result of a discrepancy in source code which can deviate from the system requirements.
- Version control system log: VC log refers to related information provided by software developers regarding changes made in a source code in order to fix *errors*, *bugs*, *defects* and *faults* that occur and are reported in a software project.

The VC log-related information is achieved in a control version system (as discussed in section 3.3) where all the logs pertaining to a given software project can be accessed by practitioners and researchers in software engineering for different purposes.

2.4 Context

Open source software witnessed an exponential growth over the last two decades. Software developers in this community collaborate and volunteer in developing complex software systems. Users of the systems and developers can submit bug reports for fixes or changes, developers in the open source community represent a successful example of software development. Developers in that community participate and collaborate at their convenience and voluntary basis. There exist two notions of software in the literature such as open source software and free software. The open source software is advocated by the Open Source Initiative (OSI). They foster the use, modification, and sharing (in modified or unmodified form) of software by anyone. Open source software might involve a team of developers and it is distributed under a license that comply with their Definition (OSI).

In this thesis, we focus on the Free/Libre Open Source Software projects for the following reasons:

1. The exponential growth and the popularity of OSS projects in commercial industries
2. The free (OSS projects) data and the accessibility to obtain the data from different software repositories in multiple formats for analysis and experiment. [62]
3. The freedom to publish results of OSS projects without breaching confidentiality agreements
4. The ability to provide researchers and practitioners with both OSS community and commercials freedom to replicate our findings and study.
5. The opportunity to contribute and adhere to the benchmarks and terms set-up by FSF and OSI in OSS community.

The OSI [4] is a global non-profit organisation focused on promoting and protecting open source software, development, and communities.

The distribution of open-source software according to OSI must comply with the following terms adapted from OSI [5]:

1. Free Redistribution: The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code: The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. The deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed.

3. Derived Works: The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code: The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups: The license must not discriminate against any person or group of persons.

[4]http://opensource.org/
[5]http://opensource.org/definition

6. No Discrimination Against Fields of Endeavour: The license must not restrict anyone from making use of the program in a specific field of endeavour. For example, it may not restrict the program from being used in business, or from being used for genetic research.

7. Distribution of License: The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product: The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software: The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral: No provision of the license may be predicated on any individual technology or style of interface.

Moving on to this, the Free Software Foundation (FSF) promotes the freedom to defend the right of the free software users. The following terms define four degrees of software freedom:

1. Redistributing copies of the software program

2. Executing a software program, for any private or commercial purpose

3. Distributing modified copies of the software program, also, given the community the accessibility of the opportunity to source code, changes as well as improvement made to the software program.

4. Accessing the source code to revise and peruse software functions and to adapt it to user desire.

The aforementioned degrees define a precise goal and the term *free* and *open source* software. Moreover, the term *Libre* software was coined to refer to the same notion of *free* software and *open source* software thus the ambiguity of the word *free* in English Language.

The FSF emphasises the “why” promotion and defending the freedom of free software or open-source software. While, the OSI focuses on the availability of source code and the process of developing open source software project. Accordingly, the OSI open-code comprises developing software model that involves the freedom of revision by developers in the process of open-source software development project with transparency or openness.

However, the discrepancies of OSI and FSF is regarded as less significant in the context of this dissertation. In this way, the terms Open Source Software (OSS) project is used to exemplify a software system developed base on these two concepts that is to say OSI and FSF.

We use the terms commercial software project to refer non-open source software in this thesis.

2.5 Categories of research around bugs

Past research on bug-related data has mined bug databases containing a wealth of information about software failures and their reports. The research around these topics has centred on how the failure occurred, what part of the system was affected, and how it was fixed. The detailed information on these aspects can be automatically mined from bug tracking systems and version control systems, and researchers in software engineering used this information to predict future occurrences of *defects*, *bugs* and *errors* in software projects. Figure 2.1 depicts a taxonomy of bug research areas, which will be discussed in the subsections below.

The rationale for categorising bugs in this thesis is of course to highlight the importance of BT data and VC logs data sets and how they were applied in each category or branch that we study in this research. The data used to produce this representation was selected using all the relevant papers from the series of international workshops on mining software repositories.

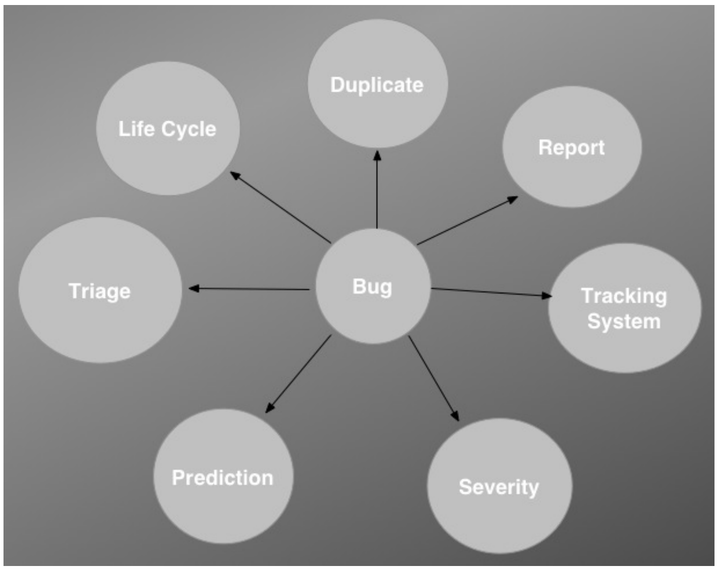


Figure 2.1: Different areas of research on bugs

Each of these research areas is analysed below to highlight the main results based on the related literature.

2.5.1 Bug triaging

Bug triaging in OSS projects is a way of assigning bugs to developers for fixes: the "triager" is a member of a software project who can assign a bug to developers. Also, the triager decides whether a bug is *new*, *unconfirmed* or *reopened*. If the bug has been fixed, then it will be marked as *resolved*. The bug is also marked as resolved if a change or fix is not needed; for instance, when the bug is duplicate, invalid or won't fix, it can be closed. In addition, when the bug is resolved, quality assurance tests take place before marking the bug status to either *verified, closed, reopened* or *unconfirmed* [7].

The following are some of the papers we study and categorise in this research based on **bug triage**.

1. The researchers in [93] propose an approach to help in bug triaging processing by classifying and predicting which developer needs to work on a bug that is reported based on the description that exists in the bug report. The system uses the naive Bayes classifica-

tion to automatically assign the report to a developer. It was evaluated on the Eclipse project and indicates an accuracy of 30%.

2. The researchers in [65] propose a graph model, based on Markov chains, that captures bug reassignment history. The model assists bug triagers to better assign developers to bug reports. The researchers effectively conducted their experiments with 445,000 bug reports. Accordingly, their model reduced bug reassignment events by up to 72% and increased the prediction accuracy by up to 23% when compared to traditional bug triaging techniques.

3. The researchers in [7] present a machine-learning-based technique that creates recommendations that assist development-oriented decisions. The authors present three different kinds of development-oriented recommenders, namely: (i) a developer recommender that suggests which developers might fix a bug; (ii) a component recommender that suggests to which product component a report might pertain; and (iii) an interest recommender that suggests which developers on the project might be interested in fixing the bug. They conducted their study with five OSS projects. The results suggest bug reports are recommended sufficiently, with up to 75% accuracy.

2.5.2 Bug life cycle

The life cycle of a bug describes a sequence which a bug must traverse before it sets to fixed in software projects [33]. Figure 2.2 depicts a life cycle of a bug and shows all the sequences of a bug as well as the transition stages. The statuses –*New, Assigned, Resolved, Verified, Unconfirmed, Reopened and Closed* – represent the stages. All the related information regarding the activities and statuses of a given bug are achieved in bug tracking systems like BugZilla and GitHub.

The life-time of bug describes a sequence of a cycle in which a bug most traverses before it has to be fixed in software projects [33]. The figure 2.2 depicted a life cycle of a bug and showed all the sequence of a bug as well as the transition stages. The statuses such as *New, Assigned, Resolved, Verified, Unconfirmed, Reopened, and Closed* represent the stages. In addition, all the related information regarding the activities and statuses of a given bug are achieved in bug tracking systems like BugZilla and GitHub.

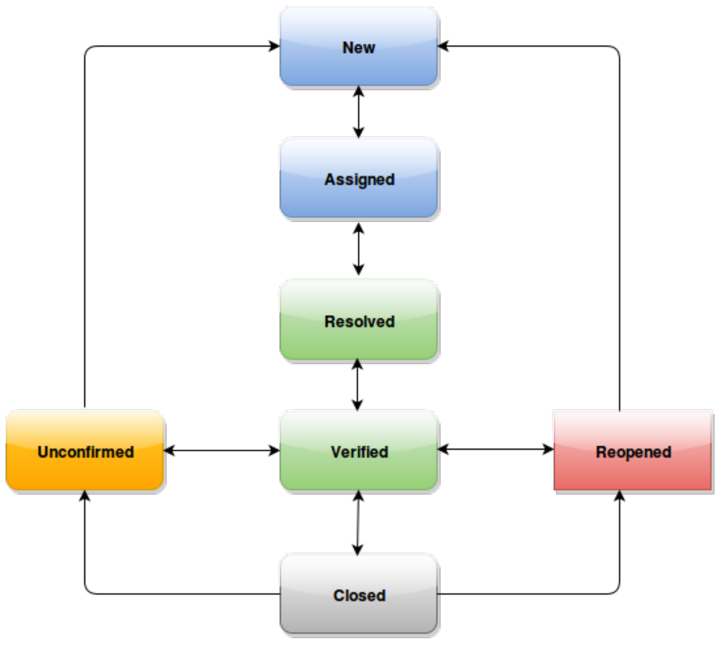


Figure 2.2: Bug life cycle

The following are some of the papers we study in this taxonomy based on the **bug life cycle**.

4. The researchers [98] explored the importance of data mining tools to predict the time to fix a bug using the information obtained at the beginning of a bug life cycle. The result reveals that an accuracy of 34.9% can be achieved. In addition, the researchers are speculating that the higher level of attributes, such as the average lifetime of a bug in specific components or products, may have a greater predictive power.

5. The researchers [33] propose an approach to support the analysis of a bug database, using two visualisations. They highlight the critical parts of the system, such as the components affected by most of the bugs. In addition, the researchers consider bug tracking systems, which store data about bugs reported by users or developers. The researchers briefly introduce the context by reporting on the particularities of the present

data, and then propose two visualisations to render bugs.

2.5.3 Bug reporting

Bug reports are of vital information in any software development project. They allow users of the system to inform developers of the problems encountered while using the system. Bug reports typically contain a description of the problem encountered while using software in natural language text format, which is used by researchers in empirical software engineering and practitioners to automatically assign developers [6] and locations where the bug can be fixed [24]. In addition, bug reports help to recognise bug duplicates [111] and predict correction efforts in an OSS project [126].

The following are some of the papers we study in our taxonomy based on **bug reporting**.

6. The researchers [35] effectively investigated how users of the system are reporting bugs: what information they provided, how frequently, and the consequences of such a bug report.

   The researchers examined the quality and quantity of information provided in 1,600 bug reports drawn from four open-source projects (Eclipse, Firefox, Apache HTTP and Facebook API) that recorded what information users actually provide, how and when users provide the information, and how this affects the outcome of the bug. The results indicate that the observed behaviour and expected results appeared in more than 50% of reports. Accordingly, there is no strong relationship observed between the provided information and the outcome of the bug.

7. The researchers [6] applied a machine-learning algorithm to the open bug repository and learned the kinds of reports each developer resolves. Their approach uses a supervised machine-learning algorithm that is applied to information in the bug repository.

   Their results reached precision levels of 57% on the Eclipse development project. For the Firefox development project, their approach achieved precision rates of over 50%, reaching 64% on one recommendation. For the GCC project, the results were far worse, with a precision rate of only 6% for one recommendation because of the characteristics of the project, such as one developer dominating the report resolution process.

8. The researchers [61] present a descriptive model of bug report quality based on a statistical analysis of surface features of over 27,000 publicly available bug reports for the Mozilla Firefox project. In addition, the model predicts whether a bug report is triaged within a given amount of time.

   The results indicate the model performs significantly better in terms of precision and recall. In addition, the researchers suggest the model can reduce the overall cost of software maintenance in a setting where the average cost of addressing a bug report is more than 2% of the cost of ignoring an important bug report.

9. The researchers [14] conducted a survey to determine the information on bug reports that Eclipse developers used and the problems they frequently encountered. The results suggest that steps to reproduce and stack traces are most useful in bug reports. The most harmful problems they frequently encountered were errors in steps to reproduce, incomplete information and wrong observed behaviour.

10. The researchers [52] conducted a large-scale quantitative and qualitative analysis of the bug reassignment process in Microsoft Windows Vista, using both quantitative and qualitative approaches. The researcher effectively quantified social interactions in terms of both useful and harmful reassignments. Their results suggested that reassignments are useful to determine the best person to fix a bug, contrary to the popular opinion that reassignments are always harmful.

11. The researchers [22] quantitatively and qualitatively analysed the questions asked in a sample of bug reports from the Mozilla and Eclipse projects. The result highlights the importance of engaging with the community effectively and efficiently in bug-fixing activities as well as keeping them up to date about the status of a bug report.

12. The researchers [15] conducted a survey of users and developers of Apache, Mozilla and

Eclipse to find out what makes a bug report. The results indicate that across all three projects the step to reproduce the information contained in a bug report was either incomplete information or wrong observed behaviour. Also the researchers developed a tool that measures the quality of a bug report. The tool was tested by the developers, which indicated a rate of about 41% of a bug report in complete agreement with the developers.

13. The researchers [76] investigated how often users participate in an open-source project and what they contribute. In addition, they further investigated and analysed the reports of Mozilla contributors who report a bug but were never assigned to fix the problem.

    Their result suggests that users are not contributing to OSS projects and that only Mozilla developers do contribute. According to their findings, one can argue that users do contribute in identifying and fixing bugs, though they might not contribute to the effort to fix the bug they reported.

    In this regard, without the users who report a bug or experience a fault in the system, how could a developer be aware that a certain fault exists within a software component after it has been delivered to the client? Even if the bug report is not clear enough to provide developers with the information they need to fix the bug, there are techniques, such as machine learning and text-mining algorithms, that extract the information they need or identify duplicate bug reports that will lead to a better bug report in an open bug reporting system.

14. The researchers [61] present a model that automatically filters a bug report. This model can predict whether a bug report has been triaged within a given period. The empirical evaluation shows that it reduced software maintenance costs by an average of 2% if the average triage cost is not greater than the cost of ignoring the important bug report.

2.5.4 Duplicate bug reports

The following are some of the papers we study in this category based on **duplicate bug report**.

15. The researchers [124] investigated duplicate bug report detection in mining software repositories using natural language and execution information. The experiment was conducted using Firefox and Eclipse bug repositories. The results of the experiment show that use of execution information can detect 67–93% of duplicate bug reports in the Firefox bug repository, while use of natural language information can detect 43–72%.

    The analysis of the results indicates natural language at some stage failed to detect more duplicate bug reports in Eclipse and Firefox repositories, while the execution information detected more duplicate bug reports.

16. The researchers [111] investigated the detection of duplicate bug reports in a case study, analysing defect reports at Sony Ericsson mobile communication using natural language processing techniques. The result of their study shows that about 2/3 (40%) of the duplicates can be found using natural language processing.

17. The researchers [16] presented empirical evidence that duplicate bug reports contain valuable information that helps developers to fix a bug.

18. The researchers [119] propose a new model design to detect duplicate bug reports on a three-bug repository (Open Office, Eclipse and Firefox). The result shows a relative improvement of 17–31%, 22–26%, and 35–43% in Open Office, Firefox and Eclipse data sets respectively.

19. The researchers [118] measured the similarity between two bug reports by introducing a retrieval function that utilised the information available in a bug report, as well as optimised the proposed retrieval function for specific bug repositories, such as Mozilla, Eclipse and Open Office. The results show a 10–27% relative improvement in mean average precision over the previous model – that is, the character N-gram-based model by [120].

20. The researchers [120] compared existing models in detecting duplicate bug reports, where the proposed model indicates a low-level feature to represent the title and detailed description of a bug report.

21. The researchers [122] extended the study of report classification by [64] by utilising the REP, which was recently proposed for report retrieval problems to measure the similarities between a bug report and determine whether they are duplicates or not. There was a new notion of similarity between two bug reports are significant enough. Their preliminary results indicate the approach is effective to increase the true positive rate of 200%.

2.5.5 Bug severity

The following are some of the papers we study in this category based on **bug severity**.

22. The researchers [89] present a new and automated approach called SEVERIS (Severity Issue assignment) that assists test engineers in assigning a severity level to defect reports. They conducted a case study on SEVERIS with data from a NASA project and an issue tracking system was presented. The results indicate that using machine learning and text-mining methods it is possible to automatically predict severity levels from the text provided in the project issue tracking system.

23. The researchers [78] investigated the possibility of predicting the severity and accuracy of a reported bug in Mozilla, Eclipse and Gnome by analysing its textual description using a text-mining algorithm.

    The results suggest it is possible to predict the severity using the information provided in a bug report as well as the textual information describing the bug in the report.

2.5.6 Bug tracking system

The following are some of the papers we study in our taxonomy based on **bug tracking system**.

24. The researchers [138] addressed the concerns of a bug tracking system by proposing four broad directions for improvement with a prototype that demonstrates an interactive bug tracking system.

25. The researchers [66] investigated and analysed the information needed and commonly faced problems with bug reporting. In addition, they conducted a survey on Apache, Eclipse, Mozilla projects and the feedback from 172 developers and users, and suggested a list of seven recommendations for a new design of a bug tracking system.

26. The researchers [64] investigated and proposed a technique to reduce the cost of the software triaging process. The proposed technique uses surface features, textual semantics and graph clustering to predict the duplicate status of a bug report. In addition, the technique is capable of reducing software maintenance costs by filtering out 8% of duplicate bug reports.

Figure 2.3 below is a box plot summarising the number of papers most commonly cited in each category of bug research that we studied in the literature.

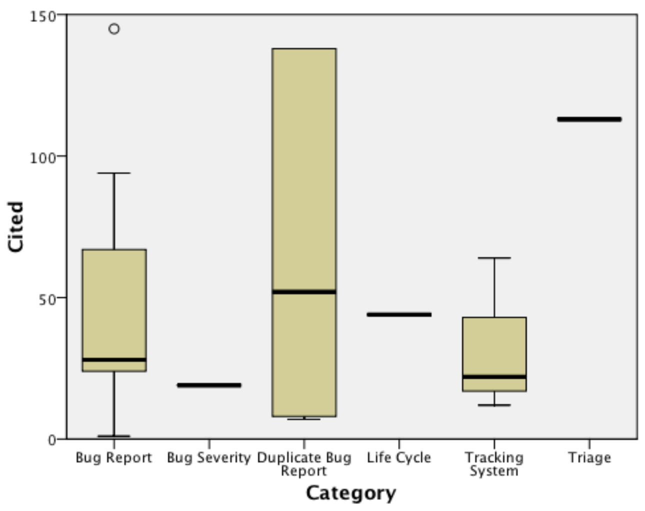


Figure 2.3: Number of papers cited most in bug research categories

Figure 2.4 below summarises the highest number of researches in each category of bug research that we considered in our taxonomy.

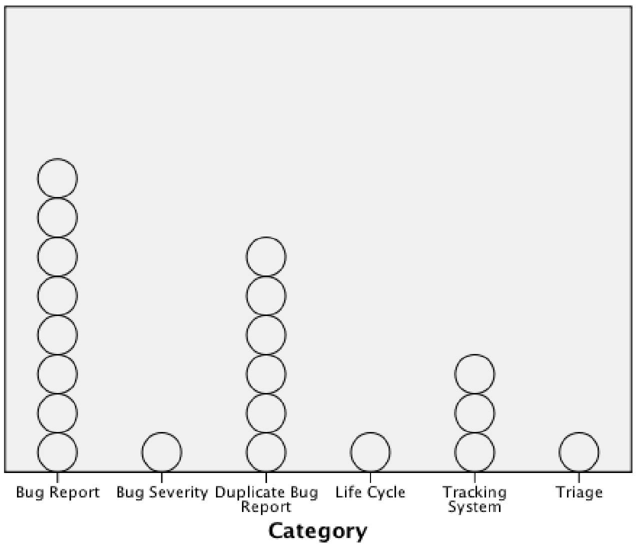


Figure 2.4: Number of researches in each category of bug research

2.6 Related tools

Current solutions (tools) have been devised by [80] [17], [130], [121], [3], [29], [83], which are all attempts to integrate and trace missing links between VC logs and BT data accurately. In this way, it is important to improve these tools by synchronising the recovered links of VC logs and BT data in either database automatically.

In this section, we will discuss the related tools for tracking VC logs and BT data. In addition, we will discuss the related work regarding the techniques that were undertaken to retrieve VC logs and BT data. We will discuss the tools that extract VC logs and BT data, as follows:

1. BucoReporter [6]
2. Bug-Code Analyser [7]

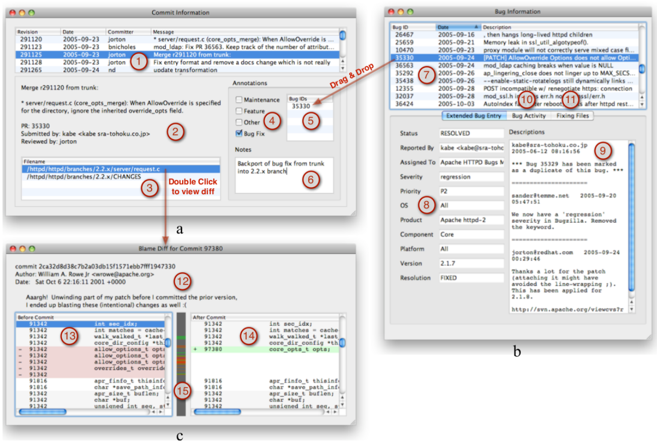
[6]http://java.uom.gr/buco/
[7]http://www.seiplab.riteh.uniri.hr/?page_id=492&lang=en

3. Linkster [8]

4. Relink [9]

2.6.1 Linkster

By way of background, the Linkster tool involves a series of steps to retrieve, parse as well as convert and link data sources [17]. As a result, it requires significant manual effort to track missing links between BT data and VC logs.



Revision	Date	Committer	Message
291120	2005-09-23	jorton	* server/request.c (core_opts_merge): When AllowOverride is ...
291123	2005-09-23	bnicholes	mod_ldap: Fix PR 36563. Keep track of the number of attribut...
291125	2005-09-23	jorton	Merge r291120 from trunk:
291128	2005-09-23	jorton	Fix entry format and remove a docs change which is not really
291265	2005-09-24	nd	update transformation

Merge r291120 from trunk:

* server/request.c (core_opts_merge): When AllowOverride is specified for the directory, ignore the inherited override_opts field.

PR: 35330
Submitted by: kabe <kabe sra-tohoku.co.jp>
Reviewed by: jorton

/httpd/httpd/branches/2.2.x/server/request.c
/httpd/httpd/branches/2.2.x/CHANGES

Maintenance / Feature / Other / [✓ Bug Fix]

Bug IDs
35330

Notes: Backport of bug fix from trunk into 2.2.x branch

Bug ID	Date	Description
26467	2005-09-16	, then hangs long-lived httpd children
25659	2005-09-21	Memory leak in ssl_util_algotypeof().
10470	2005-09-23	proxy module will not correctly serve mixed case fi...
35330	2005-09-24	[PATCH] AllowOverride Options does not allow Opti...
36563	2005-09-24	mod_ldap caching breaks when value is NULL
35292	2005-09-26	ap_lingering_close does not linger up to MAX_SECS...
35438	2005-09-26	--enable-static-rotatelogs still dynamically links ...
12355	2005-09-28	POST incompatible w/ renegotiate https: connection
32037	2005-09-28	mod_ssl.h i[illegible]s err.h no[illegible]ssl/err.h
36424	2005-10-03	AutoIndex fa[illegible]ter reboot[illegible]s after httpd rest...

Status	RESOLVED
Reported By	kabe <kabe@sra-tohc
Assigned To	Apache HTTPD Bugs M
Severity	regression
Priority	P2
OS	All
Product	Apache httpd-2
Component	Core
Platform	All
Version	2.1.7
Resolution	FIXED

```
kabe@sra-tohoku.co.jp
2005-06-12 08:16:56

*** Bug 35329 has been marked
as a duplicate of this bug. ***

====================

sander@temme.net   2005-09-20
05:47:51

We now have a 'regression'
severity in Bugzilla. Removed
the keyword.

====================

jorton@redhat.com   2005-09-24
00:29:46

Thanks a lot for the patch
(attaching it might have
avoided the line-wrapping ;).
This has been applied for
2.1.8.

http://svn.apache.org/viewcvs?r
```

```
commit 2ca32d8d38c7b2a03db15f1571ebb7fff1947330
Author: William A. Rowe Jr <wrowe@apache.org>
Date:   Sat Oct 6 22:16:11 2001 +0000

    Aaargh!  Unwinding part of my patch before I committed the prior version,
    I ended up blasting these (intentional) changes as well :(
```

```
Before Commit
   91342          int sec_idx;
   91342          int matches = cache
   91342          walk_walked_t *last_
   91342          core_dir_config *thi
 - 91342          allow_options_t opts
 - 91342          allow_options_t opts
 - 91342          allow_options_t opts
 - 91342          overrides_t override
 - 91342
   91816          apr_finfo_t thisinfo
   91816          char *save_path_info
   91342          apr_size_t buflen;
   91342          char *buf;
   91342          unsigned int seq, s
```

```
After Commit
   91342          int sec_idx;
   91342          int matches = cache
   91342          walk_walked_t *last_
   91342          core_dir_config *th
 + 97380          core_opts_t opts;



   91816          apr_finfo_t thisinf
   91816          char *save_path_inf
   91342          apr_size_t buflen;
   91342          char *buf;
   91342          unsigned int seq, s
```

Figure 2.5: Linkster tool screenshot adapted from [17]

Figure 2.5 depicts a screenshot of the Linkster tool by [17], which displays three kinds of information on Windows: (i) commit transactions including all the changed files; (ii) bug reports; and (iii) diff and blame information for all of the lines in a file before and after a particular commit. In addition, the tool requires access to VC system and BT system.

2.6.2 Relink

ReLink, developed by [130], collects information automatically from the source code repository and bug tracking system, builds the resulting information linked to bugs/issues or logs and outputs the identified links. Figure 2.6 depicts the overall process of the ReLink tool.

[8]Linkster is not publicly available for download.
[9]`https://code.google.com/p/bugcenter/wiki/ReLink`

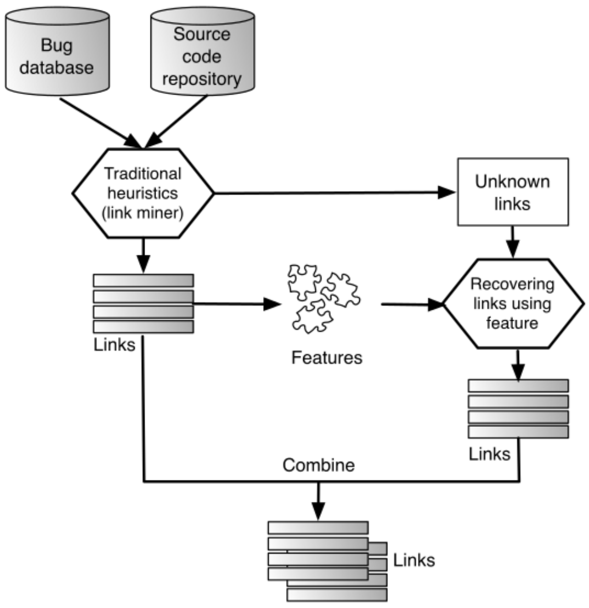


Figure 2.6: Architectural overview of Relink Tool adapted from [130]

ReLink was applied to three open-source projects – ZXing, OpenIntents and Apache – and two simulation studies on Apache and Eclipse MAT [130].

The researchers effectively evaluated the recovered links that are manually recovered and verified links. On average, for the three OSS projects, ReLink recovered links with 78% recall and 89% precision, while traditional heuristics only achieved 64% recall and 91% precision.

In general, the tool requires a large amount of interaction, but recovers missing VC logs and BT data accurately.

2.6.3 Buco Reporter

The BuCo Reporter, developed by [83], is an extensible framework that mirrors development logs and the bug tracking data, and it generates a complete set of evolutionary facts and metrics about a given OSS project. BuCo accurately traces development logs and bugs, but it was not designed and developed to **"synchronise"** the missing development logs and bugs if discrepancies were found. Figure 2.7 depicts the core module of BuCo Reporter.

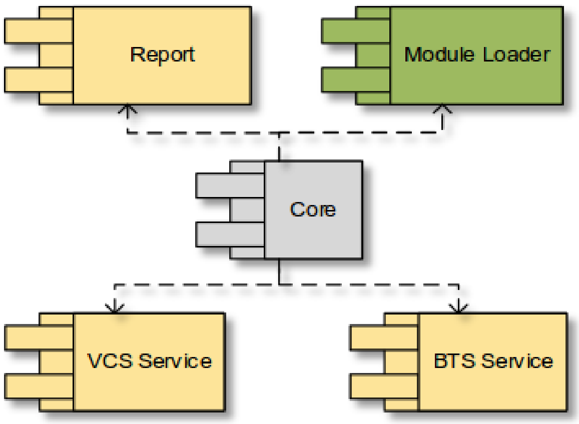


Figure 2.7: Core module of BuCo Reporter adapted from [83]

2.6.4 Buco Analyser

BuCo Analyser was designed and developed to effectively retrieve VC logs and BT data from open-source bug tracking systems and source code management repositories. The tool addressed the issue of linking VC logs and BT data sources, but still failed to synchronise the recovered and linked VC logs and BT data.

Figure 2.8 depicts an architectural overview of BuCo Analyser. The external interfaces that connect BT system and VC system allow the tool to extract VC logs and BT system raw data, as well as perform the calculation on software metrics that exist in OSS projects [87].

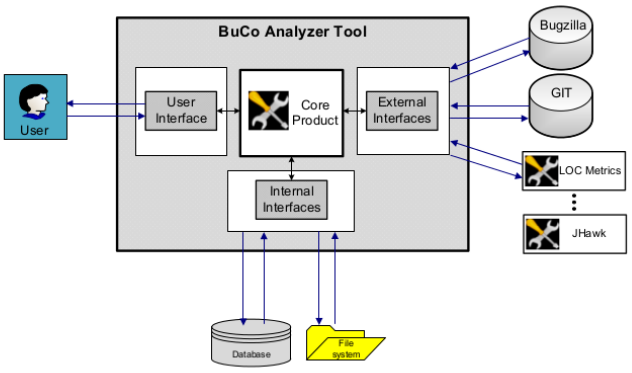


Figure 2.8: Architectural overview of the BuCo Analyser tool adapted from [87]

On the other hand, below are some of the tools that extract VC logs and BT data independently, each of which we will discuss in detail.

1. Bug Locator [10]
2. Bicho [11]
3. CVSAnalY [12]
4. Hipikat [13]
5. SoftChange [14]

2.6.5 Bug Locator

Bug Locator, which was proposed by [136], can automatically search for relevant BT data in the source code based on initial bug reports. It can effectively retrieve BT data by issuing a given query to evaluate BT data localisation performance. In addition, it utilises the relevant information regarding the BT data that exist and have been confirmed as fixed. Bug Locator uses the Vector Space Model to extract effectively relevant BT data. Thus, the tool traces links between VC logs and BT data using the traditional heuristics proposed by [11]. In this way, it is not designed to support the integration and synchronisation of VC logs and BT data. However, the tool effectively performs bug localisation in general.

2.6.6 Bicho

Bicho[15] is a command-line-based tool used to extract BT data from BT systems like Bugzilla, Sourceforge, GitHub and JIRA. Specifically, Bicho extracts bug information such as *Priority,*

[10]https://code.google.com/p/bugcenter/downloads/list
[11]https://github.com/MetricsGrimoire/Bicho
[12]https://github.com/MetricsGrimoire/CVSAnalY
[13]https://www.cs.ubc.ca/labs/spl/projects/hipika/downloads.html
[14]http://sourcechange.sourceforge.net/
[15]Bicho supports the following trackers: Bugzilla (>4) Sourceforge.net (abandoned), Jira (unstable), Launchpad, Allura (unstable), Github (unstable).

Status, Resolution and *Changes* and is automatically stored in a MySQL database. Figure 2.9 is the Bicho schema[16] and depicts relevant entities in the Bicho database.

In this way, the purpose of Bicho is to retrieve BT data from BT system such as BugZilla and GitHub and store the information locally in a MySQL database for posterior analysis.

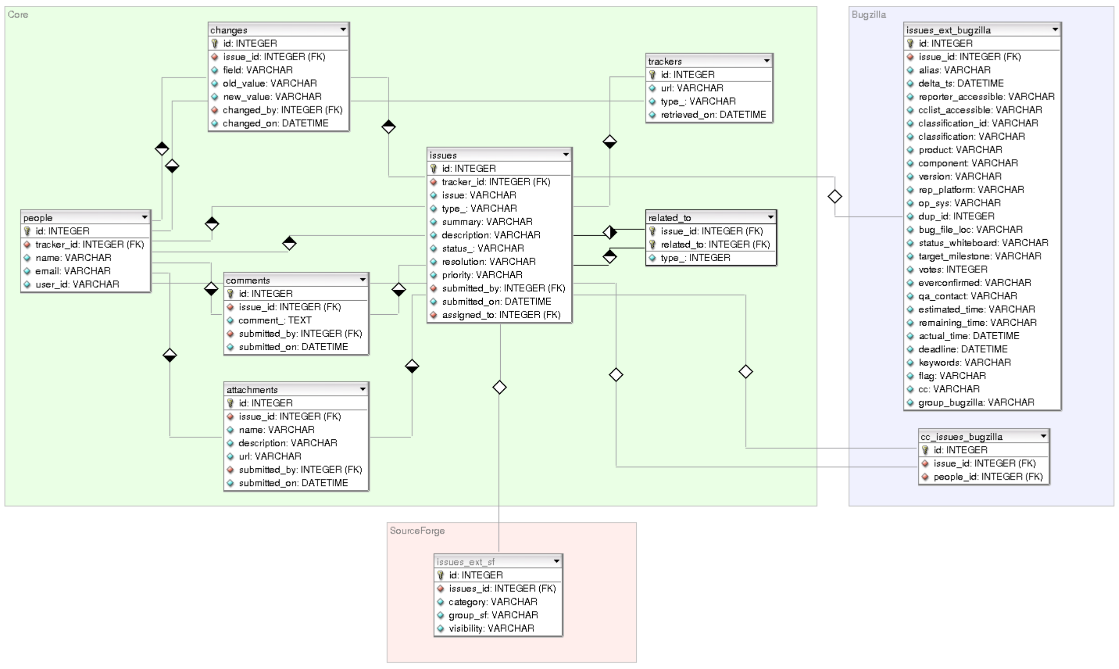


Figure 2.9: Bicho schema

2.6.7 CVSAnalY

CVSAnalY is also a command-line-based tool that extracts data out of logs of repositories and then automatically stores them in a MySQL database for subsequent analyses.

The purpose of CVSAnalY is to analyse the events that occur in the source code. This includes the developers' actions during software corrective maintenance activities from various source code management systems [107] like CV system, SVN or Git and stored VC logs in MySQL databases.

The structure of CVSAnalY was designed with ten entities (tables) and three additional

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

entities. In addition, it is divided into two main parts. The first part consists of the set of entities that represent the history of the project based on the information from the log. CVSAnalY filled these tables during the parsing process exclusively with the information provided by the repository log (SCM). Thus, these tables will always be present in the schema independently of how CVSAnalY is executed.

Also, the second set of tables in CVSAnalY is composed of tables of various extensions. Information provided in these tables depends on every CVSAnalY extension. Figure 2.10 is the CVSAnalY schema[17] including relevant entities that exist in the CVSAnalY database. Also, each table contains a data field and data type where the information extracted by CVSAnalY is held in a MySQL database.

2.6.8 Hipikat

Hipikat, proposed by [30], is a tool designed to form an implicit group memory from the VC logs of software projects and recommend source code from the archives that are relevant for fixes when a new developer in an OSS project is assigned to perform a certain task. Hipikat functions as a plug-in that works within the Eclipse IDE. Thus, it is designed to recommend VC logs of OSS projects and is not suitable to automate and sync VC logs and BT data. Figure 2.11 depicts a screenshot of Hipikat run in Eclipse IDE. In summary, it effectively recommends source code to new developers in software development projects.

2.6.9 SoftChange

SoftChange, by [46], retrieves software artefacts and is designed for the analysis and enhancement of software artefacts retrieved from CV system. In addition, it allows the user to visualise the information effectively. However, the purpose of SoftChange is to analyse the VC logs to help uncover the history and evolution of the software project. In this way, it might be difficult to support the synchronisation and integration of various sources, such as VC logs and BT data, simultaneously, since it is designed to extract information from VC system purposely to visualise the evolutionary aspect of a given OSS project.

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

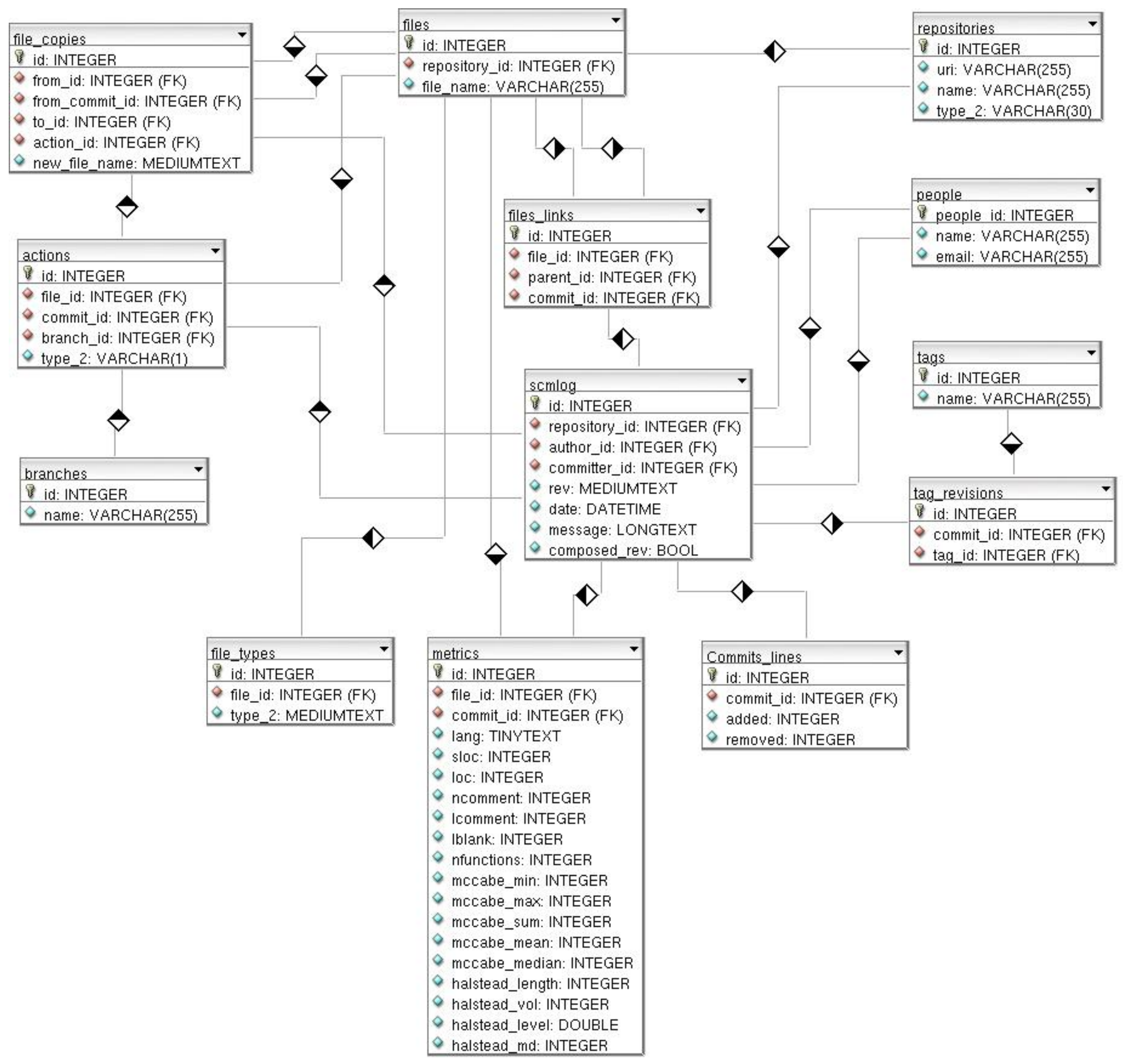
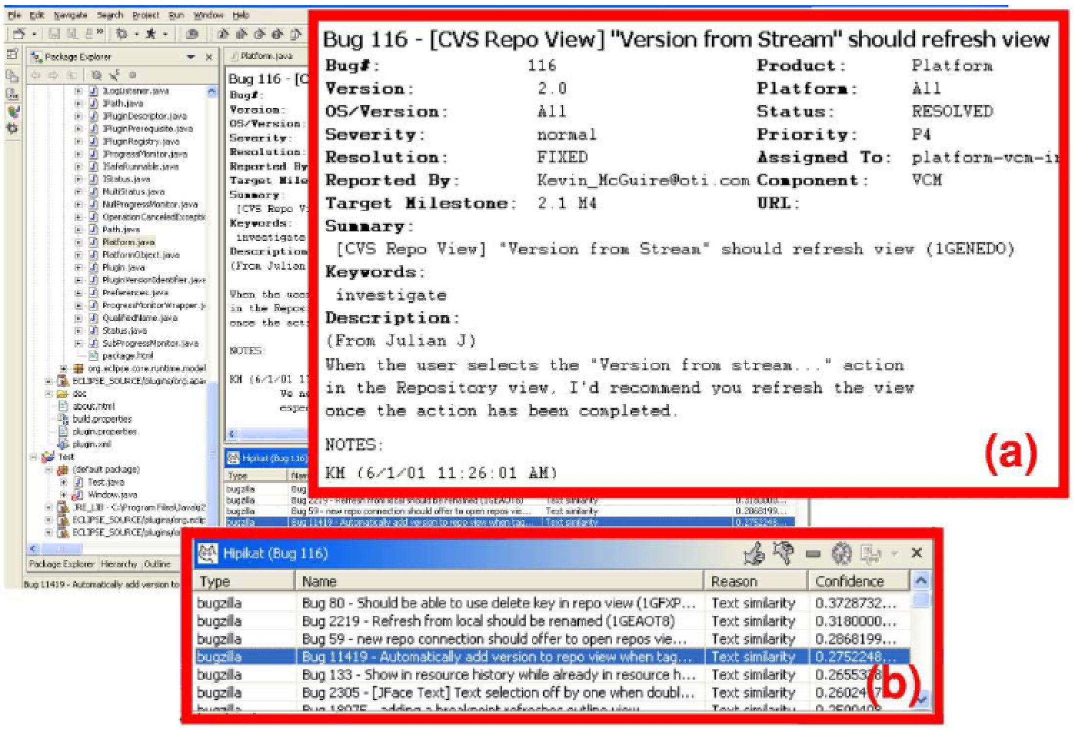


Figure 2.10: CVSAnalY schema



Bug 116 - [CVS Repo View] "Version from Stream" should refresh view

Bug#:	116	**Product:**	Platform
Version:	2.0	**Platform:**	All
OS/Version:	All	**Status:**	RESOLVED
Severity:	normal	**Priority:**	P4
Resolution:	FIXED	**Assigned To:**	platform-vcm-i
Reported By:	Kevin_McGuire@oti.com	**Component:**	VCM
Target Milestone:	2.1 M4	**URL:**	

Summary:
[CVS Repo View] "Version from Stream" should refresh view (1GENEDO)
Keywords:
investigate
Description:
(From Julian J)
When the user selects the "Version from stream..." action
in the Repository view, I'd recommend you refresh the view
once the action has been completed.

NOTES:
KM (6/1/01 11:26:01 AM)

Hipikat (Bug 116)

Type	Name	Reason	Confidence
bugzilla	Bug 80 - Should be able to use delete key in repo view (1GFXP...	Text similarity	0.3728732...
bugzilla	Bug 2219 - Refresh from local should be renamed (1GEAOT8)	Text similarity	0.3180000...
bugzilla	Bug 59 - new repo connection should offer to open repos vie...	Text similarity	0.2868199...
bugzilla	Bug 11419 - Automatically add version to repo view when tag...	Text similarity	0.2752248...
bugzilla	Bug 133 - Show in resource history while already in resource h...	Text similarity	0.26553[illegible]
bugzilla	Bug 2305 - [JFace Text] Text selection off by one when doubl...	Text similarity	0.26024[illegible]

Figure 2.11: Screenshot of Hipikat adapted from [30]

Figure 2.12 depicts an architectural overview of SoftChange.

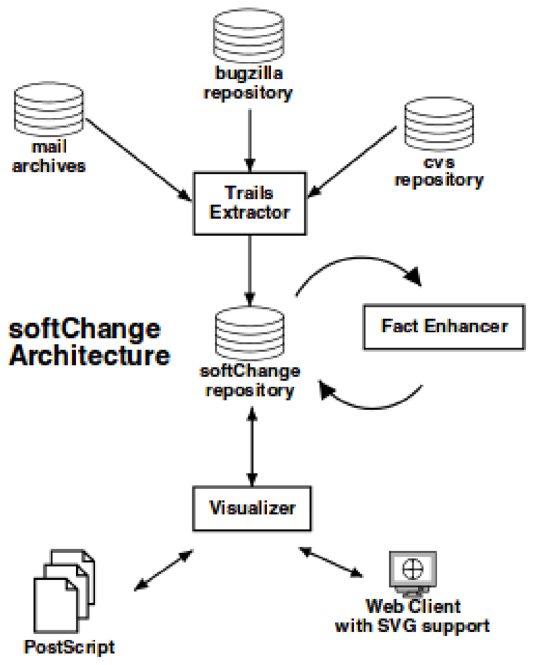


Figure 2.12: Architectural overview of SoftChange, adapted from [46]

However, research by Kim et al [75] and Sliwerski et al [116] has demonstrated and validated manually that the VC logs and bug-related data are referring and pointing to actual fixes using SZZ algorithm (discussed in Section 2.7 of this chapter) as well as automatically and accurately identifying bug-introducing changes. In this thesis, we go one step further and improve the functionality of a VC tool and BT tool we studied and reported in Section 2.6 of this chapter by synchronising (filling) the missing data (i.e., BT data and VC logs) of OSS projects in their respective database and, in an automatic way. Similarly, the study by Sureka et al [121] applies a formal mathematical model to automate the process of identifying missing links between bug-fixing commits in VC logs and their associated bug reports. The model is effective in recovering such missing links: in this research, as mentioned earlier, we proposed to use the identified discrepancies to synchronise the VC logs or BT data when missing links are recovered.

Traceability links are needed to building defect prediction models and [134]. However, the available tools to document VC logs and BT data lack integration [42]. As a result, two independent sets of bug-related data are produced, filling different databases [80] [109]. It has been suggested that using the bug IDs from VC log messages could help to identify and recover missing traceability links [51] [85] [10]. These logs need to be manually or semi-automatically analysed and compared, to determine if logs and IDs from BT system are referring to the same set of bug IDs, or if they refer to disjoint sets.

2.7 Related techniques - SZZ algorithm

Various researchers, including [30] [42] [75] [116] [3] [121], have attempted to integrate and identify missing links between VC logs and BT data into BT Systems and VC Systems. In this thesis, we applied the same technique, but differently with an attempt to **"synchronise"** either the missing VC logs or BT data of software projects using the SZZ algorithm.

However, Mausa et al [88] effectively evaluated the current techniques and approaches to solving the traceability issues in linking of VC logs and BT data of OSS software projects. In general, they found that the use of regular expressions might work well. Also, they suggest that researchers and practitioners should adapt each part of the subset of the SZZ algorithm (i.e., "fix", "bug" and # identifier) to a particular software repository, perhaps to a software project in traceability links. In this way, it is important to dissect the most widely used SZZ regular expressions [116], because it will serve as a guide in syncing complete recovered VC logs and BT data of open-source software projects.

Moreover, the study of Matsuda et al [86] proposes a technique for hierarchically grouping commits that are similar to our approach for BT data to retrieve sets of operations by specifying the granularity of VC logs. Their technique effectively reorganises VC logs by recording editing operations of source code based on types of refactoring.

The approach of mining VC logs and BT data – that is to say, SZZ – is among the widely used algorithms to look for bug-fixing commits, with a set of simple rules [116]. A recent study by Shepperd et al [113] contributes a significant and more appropriate way to clean bug-related data sets for empirical research applied to the NASA case study.

VC logs and BT data have been searched in basically two ways: (i) by using keywords such as “fixed” or “bug” [91]; and (ii) by searching for references to bug reports, for instance the use of the # sign and various numeric values (e.g., #1234) that are linked to the ID of a bug [30] [42] [74] [116]. The SZZ algorithm is an example of an approach that combines keywords and proxies to detect bug-fixing commits. Many have used the SZZ algorithm to detect bug-induced changes in OSS projects, but in this research we use it partially and make it better in detecting VC logs and BT data of OSS projects.

2.8 Summary of the Chapter

In summary, most of the tools reported and found in the literature retrieve and trace VC log and BT data effectively. Thus, in this thesis we will integrate a VC tool and a BT tool as well as identify and synchronise the missing data (BT data and VC logs of OSS project) in their respective databases. The contribution of the presented research is towards a complete framework to synchronise the missing VC logs and BT data, supporting various repositories and bug tracking algorithms and approaches [108].

The evaluation of the current techniques and approaches to solving the traceability issues in linking of VC logs and BT data of software projects by Mausa et al [88] suggests the use of regular expressions might work well. Similarly, they compare their effectiveness with other well-known bug linking techniques and tools, such as ReLink by Wu et al [130] and BuCo Reporter by Ligu et al [83], based on the regular expression. Their results suggest the technique and tools are equally as effective as other proposed techniques in solving the issue of traceability links.

Chapter 3

Methodology

3.1 Introduction

In the previous chapter this thesis discussed the tools and techniques related to the issue of finding bug data in VC logs. In this chapter we will discuss the methodology and techniques that were applied in this thesis to extract the bug-related data from the VC system and from the BT system. Similarly, we will discuss the rationale for selecting tools to store VC logs and BT data. In addition, we will discuss **how** we selected the OSS projects and automated the download, extraction and storage of VC logs and BT data. We will also detail **how** we performed the comparison between the VC logs and BT data.[1]

3.2 Sampling an OSS forge

We impose certain requirements and criteria in sampling the OSS forge. The requirements and criterion itemised below are essential in sampling the required number of OSS projects needed for this research as follows:

- The OSS project must be maintained and remain under active development. This ensures the analysed VC logs and BT data will not be obsolete or irrelevant to our approach.

[1]Parts of this chapter appears in two papers published in OpenSym 2014 and EASE 2015 [109] [108]

- 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 automatic extraction and synchronisation of missing data. Data from these repositories will be extracted by the tool chain developed for this research using the identified tools (i.e., CVSAnalY and Bicho) presented in Section 2.6. This criterion 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, the VC logs must be based on Subversion, VC system or Git – that is, the VC systems supported by CVSAnalY. In fact, both tools sometimes encounter a time-act during data extraction. The issue was resolved by imposing a delay of 15 seconds before sending each request to the repository when mining VC logs and BT data using the tool chain developed for this research. Following this methodology, only repositories that can be processed by CVSAnalY and Bicho have been considered.
- The OSS project BT system repository must be a tracker supported by the Bicho tool[2].

We extracted the projects' data from the GitHub repository through a crawler developed in Perl[3], obtaining the sample of OSS projects. The FLOSSmole project contains the population of GitHub projects as of February 2013, as found in `http://flossdata.syr.edu/data/gh/2013/`. The population on that data dump is 3,640,870 projects. Below is the formula we used in calculating the sample size of the OSS forge:

$$Samplesize = \frac{Z^2 * (p)^*(1-p)}{c^2} \tag{3.1}$$

Where:

Z = confidence level

p = percentage picking a choice

c = confidence interval (or merging error)

[2]Bicho supports the following trackers: Bugzilla (>4) Sourceforge.net (abandoned), Jira (unstable), Launchpad, Allura (unstable), Github (unstable)

[3]Refer to Appendix B in Section A.1 for the script

The sizing of the sample was achieved by considering a 95% confidence level and a 5% confidence interval, resulting in 384 projects. Each project in the sample was given a unique ID and the randomisation process was carried out using Excel. A randomiser extracted 384 random numbers between 1 and 3,640,870. After manual inspection, we found that some 40 projects were empty, hence giving an overall number of 'alive' projects of 344, which is the sample that was studied. The final sample of projects can be found on Figshare.[4]

The justification of the sample size of 344 OSS project in this study was to ensure that a sufficient number of OSS projects is used in the analysis and evaluation of the proposed approach. Thus, it is essential to determine the number of OSS projects likely to be required to avoid a Type II error, which is the likelihood of concluding there is no missing links between BT data and VC logs of OSS projects [117]. In other words, the 344 OSS projects sampled in this thesis will be sufficient to ensure that the thesis has acceptable statistical *power* to support in validating our hypothesis mentioned in Chapter 1 Section 1.2 (i.e., Obj2 and Obj3) of this thesis.

However, after obtaining the list of the projects, we then manually checked each of the 344 selected OSS projects to ensure all the projects had met the criteria we set in Section 3.1 of this chapter, before extracting (i.e., checking out the projects' VC logs and BT data). The data we obtained via FLOSSmole for all the projects from GitHub do not have the URL to extract VC logs and BT data. In this way, a project name was randomly selected from the list of projects. A manual check-up to confirm the URL on the GitHub repository was also carried out before a project was included in the list of 344 OSS projects in this thesis. In addition, some of the projects on the data obtained via FLOSSmole do not exist on the GitHub repository and others were empty. Thus, the projects were excluded from part of the study and the data randomly replaced. That is to say, the list of the projects on GitHub was obtained via FLOSSmole. The text file contains the list of all the project names and their URL, which was used to extract all the VC logs and BT data. The list of the project names and URL can be found on Figshare.[5]

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

3.3 Selecting the most appropriate VC system

VC systems are used to track changes made in the source code by developers at different stages of a software project [123]. The information about the projects is stored and can be accessed later for revision and software corrective activities. Researchers benefit from getting this information by exploring and observing the evolution of software systems to predict which part of the system might be faulty or needs careful consideration before the system fails during normal operation. VC systems are a fundamental tool in any open-source software project, because they help software development teams to keep track of their progress and any other changes made in the source code during software development [60] [37]. VC systems include CVS, SVN, Git and Mercurial.

3.3.1 Distributed and Centralised VC system

VC systems are classified into two types: **distributed** VC systems and c**entralised** VC systems. The difference between distributed and centralised VC systems is that centralised VC systems keep the logs of changes on a central server, from which every member of the software development team can request the latest version of the source code or file and then later push the latest changes/commit they made to the centralised VC system.

In this way, every member of the software development team sharing the server shares the team source code or files. A typical example of a software repository hosting OSS projects that uses a centralised VC system is SourceForge. Figure 3.1 depicts an architectural overview of a centralised VC system.

Conversely, in a distributed VC system, every member of the software development team has a working copy of the entire project and logs (i.e., source code or files) locally. In this case, a member of the software development team is not required to be online to commit or make any changes in the source code. Members of the software development team can *pull* a request with any other team member in the software project. A typical example of a software repository hosting OSS projects that uses a distributed VC system is GitHub. The 344 OSS projects we sampled for this research use distributed VC systems. One advantage is that in

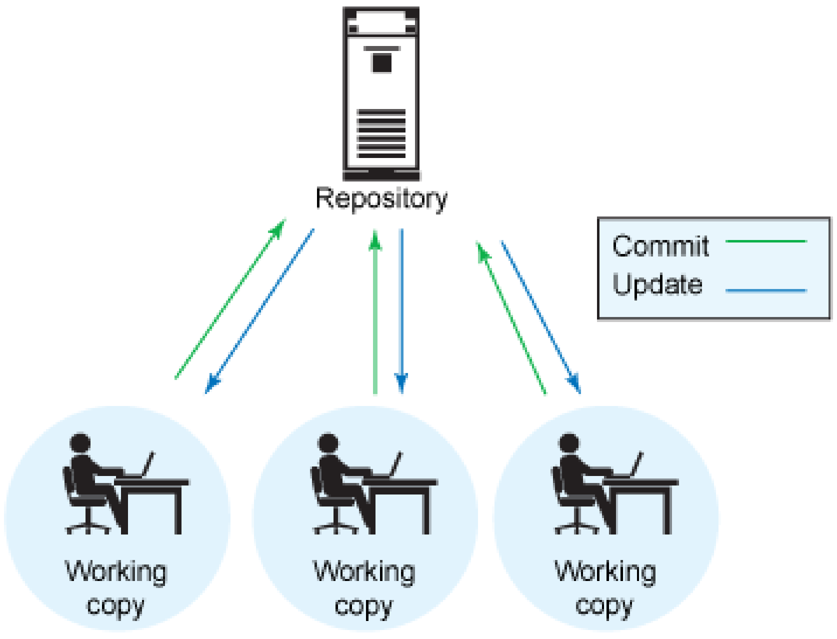


Figure 3.1: Centralised VC system adapted from [73]

distributed VC systems there is no central system in which the entire logs (i.e., source code) are hosted; every member of the software development team has a working copy in case there is a failure or crash in the central VC system server. Figure 3.2 depicts an architectural overview of a distributed VC system.

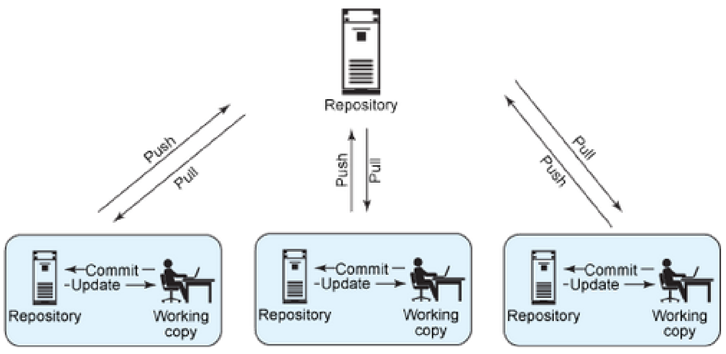


Figure 3.2: Distributed VC system adapted from [73]

GitHub [6] has 10 million people and developers who currently contribute to over 26 million projects; many are active projects [71]. Each repository on GitHub has a dedicated project page that hosts the source code files, commit history, open issues, and other data associated with the project [32]. Developers create permanent URLs to link to specific lines of code in a file. As a result, we chose the GitHub repository and retrieved VC logs and BT data using the identified tools. Regarding the tools that are used to extract data from VC systems, Robles [103] proposed the use of existing tools and data sets when mining software repositories in order to allow other researchers to validate and replicate existing studies. Many researchers in MSR develop their own tools, and some researchers are not making their tools and data sets publicly available for replication and validation [62].

Therefore, we chose Bicho and CVSAnalY to improve and combine their functionality by automating the process of downloading, extraction and storage of VC logs and BT data collectively. These tools are developed as part of the Metrics Grimoire project[7]. In this way, we want to improve and combine their functionality and reuse existing tool sets and data sets for mining software repositories to allow other researchers to replicate and validate our study. In addition, Bicho and CVSAnalY were effectively designed and developed with organised databases for easy querying. Similarly, the data structure in their databases corresponded for cross-database analysis [49].

3.3.2 CVSAnalY

CVSAnalY is a command line-based tool that extracts data out of logs of repositories and then automatically stores them in a MySQL database for subsequent analyses.

The purpose of CVSAnalY is to analyse the events that occur in the source code. The data extracted include the developers' actions during software corrective maintenance activities from various source code management systems [107]. CVSAnalY supports, among others, the CVS, SVN and Git systems and stores the VC logs in a MySQL database.

[6]https://github.com/about
[7]https://metricsgrimoire.github.io/

3.3.3 Description of CVSAnalY schema

Figure 2.10 depicts the schema of CVSAnalY[8]. The structure of CVSAnalY was designed with ten entities (tables) and three additional entities (13 in total). In addition, it is divided into two main parts. The first part consists of the set of entities that represent the history of the project based on the information from the log. CVSAnalY filled these tables during the parsing process exclusively with the information provided by the repository log (SCM). Thus, these tables will always be present in the schema independently of how CVSAnalY is executed. The most up-to-date CVSAnalY is designed with 13 tables. In each table there exists a data field and data type where the information extracted by CVSAnalY is held in a MySQL database. The tables that exist in a CVSAnalY database are shown in Table 3.3.

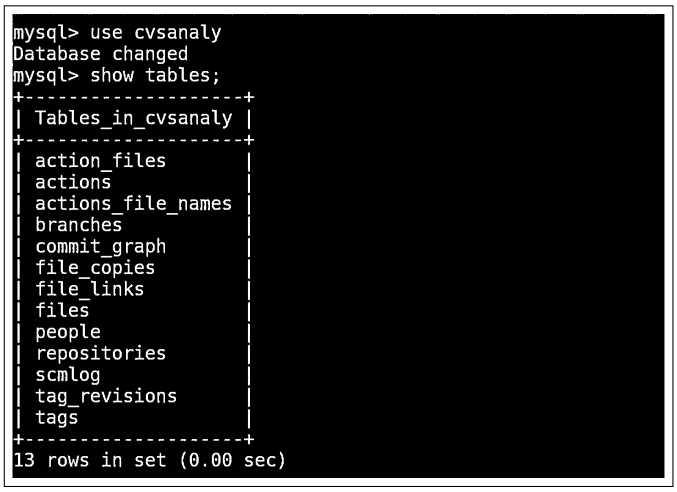

```
mysql> use cvsanaly
Database changed
mysql> show tables;
+--------------------+
| Tables_in_cvsanaly |
+--------------------+
| action_files       |
| actions            |
| actions_file_names |
| branches           |
| commit_graph       |
| file_copies        |
| file_links         |
| files              |
| people             |
| repositories       |
| scmlog             |
| tag_revisions      |
| tags               |
+--------------------+
13 rows in set (0.00 sec)
```

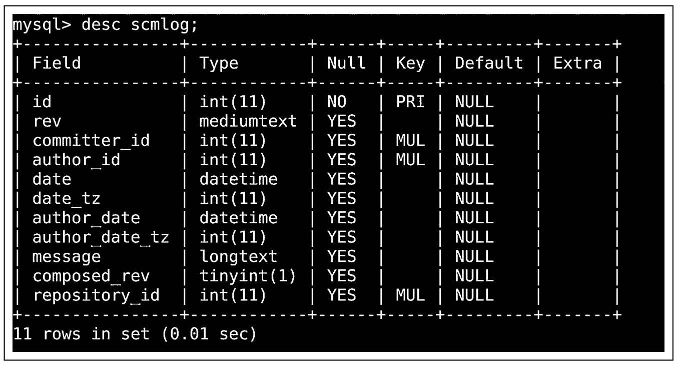
Table 3.3: CVSAnalY Database Tables

In this section, we will discuss some of the tables produced by CVSAnalY that we consider relevant in order to narrow down and carefully examine the issue of how and where to identify VC logs of OSS projects in CVSAnalY. Below we look at the tables *SCMlog*, *repositories* and *people*.

1. The **SCMlog** table contains the information about every VC log in the repository displayed. The table data field consists of the *ID* of the project in the database and

[8]https://github.com/MetricsGrimoire/CVSAnalY/wiki/Database-Schema

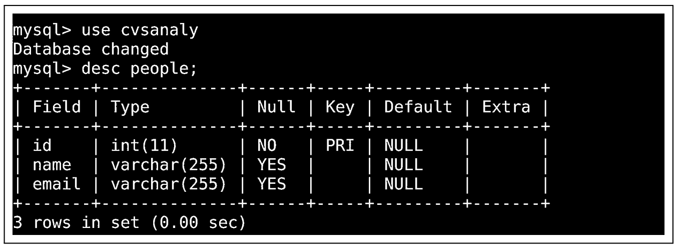
revision (**rev**) of each VC log, as well as the actual commits developers made in bug-fix processes and feature enhancement. In this table, we can track and link who, how and when changes were made in the source code of a given OSS project that we extract. This will help in identifying relevant entities and the data field that are suitable for the automation, integration and synchronisation of VC logs into BT data and vice versa. Thus, we deliberate on the SCMlog table as the entity that will be used in re-aligning the tools. Conversely, we further assess and evaluate the data fields that exist in the table. It's important to ensure the data formats are also carefully examined during the syncing process between the VC logs held by CVSAnalY (database) and the BT data held by Bicho (database) to avoid redundancy and data conflict. In addition, the data fields that we deliberate in the SCMlog table are identified in table 3.12. Table 3.4 is the description of the **SCMlog** table in the CVSAnalY database.



```
mysql> desc scmlog;
+----------------+------------+------+-----+---------+-------+
| Field          | Type       | Null | Key | Default | Extra |
+----------------+------------+------+-----+---------+-------+
| id             | int(11)    | NO   | PRI | NULL    |       |
| rev            | mediumtext | YES  |     | NULL    |       |
| committer_id   | int(11)    | YES  | MUL | NULL    |       |
| author_id      | int(11)    | YES  | MUL | NULL    |       |
| date           | datetime   | YES  |     | NULL    |       |
| date_tz        | int(11)    | YES  |     | NULL    |       |
| author_date    | datetime   | YES  |     | NULL    |       |
| author_date_tz | int(11)    | YES  |     | NULL    |       |
| message        | longtext   | YES  |     | NULL    |       |
| composed_rev   | tinyint(1) | YES  |     | NULL    |       |
| repository_id  | int(11)    | YES  | MUL | NULL    |       |
+----------------+------------+------+-----+---------+-------+
11 rows in set (0.01 sec)
```

Table 3.4: SCMlog table

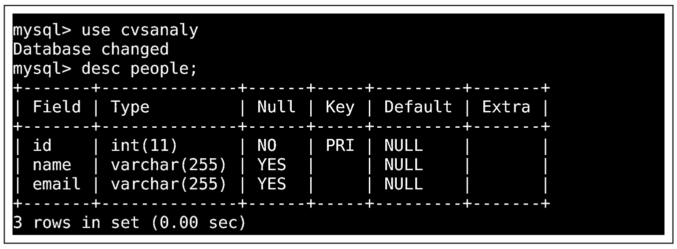
2. The **repositories** table holds the links for all the OSS projects that CVSAnalY extracts and stores in its database. It contains the **ID**, **uri** and the **name** of the OSS project. However, the **repositories** table in CVSAnalY is linked to the **tracker** table in the Bicho database. Interestingly, they hold similar data fields and formats. We narrow the selection of the entities, since we are only interested in the VC system log and BT data held by these tools. In this way, the data structures of Bicho and CVSAnalY appear to be similar, and this indicates and increases the possibility of realigning their functionality. Table 3.5 depicts a description of the **repositories** table in the CVSAnalY database.



```
mysql> use cvsanaly
Database changed
mysql> desc people;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | NO   | PRI | NULL    |       |
| name  | varchar(255) | YES  |     | NULL    |       |
| email | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Table 3.5: structure of the 'repositories' table in CVSAnalY database

3. The **people** table in the CVSAnalY database contains three data fields: **id**, **name** and the **email** address of the developer or a member of the project team. The description of this table is also relevant to the **people** table in the Bicho database. Table 3.6 is a description of the people table in CVSAnalY. Also, each table contains a data field and data type where the information extracted by CVSAnalY is held in a MySQL database.



```
mysql> use cvsanaly
Database changed
mysql> desc people;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | NO   | PRI | NULL    |       |
| name  | varchar(255) | YES  |     | NULL    |       |
| email | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Table 3.6: structure of the 'people' table in the CVSAnalY database

Also, each table contains a data field and data type where the information extracted by CVSAnalY hold in MySQL Database.

3.4 Selecting the appropriate BT system

BT systems are often designed and developed as a database-driven web application with an interface that allows multiple users to submit bug reports simultaneously [138]. These systems can be crawled by automated tools to store the retrieved information in a localised database for analysis.

Furthermore, BT systems are used in most open-source software projects to deal with reporting software bugs, feature enhancement reports from users of the system and developers. Hence, among the most widely used BT systems in the OSS community is Bugzilla [99]. This system allows users to report a bug in addition to managing bug reports and feature requests through a publicly available interface on the web. The BT system data we used in this research was sampled from GitHub.

3.4.1 Bicho

Bicho is a command line-based tool used to extract BT data from BT systems like Bugzilla, SourceForge, GitHub and JIRA. Specifically, Bicho extracts bug information such as *"Priority", "Status", "Resolution" and "Changes"* which is automatically stored in a MySQL database.

In this way, the purpose of Bicho is to extract BT data from bug-tracking systems. Bicho's back end handles each specific BT system, such as Bugzilla or GitHub, and stores the information locally in a MySQL database for posterior analysis.

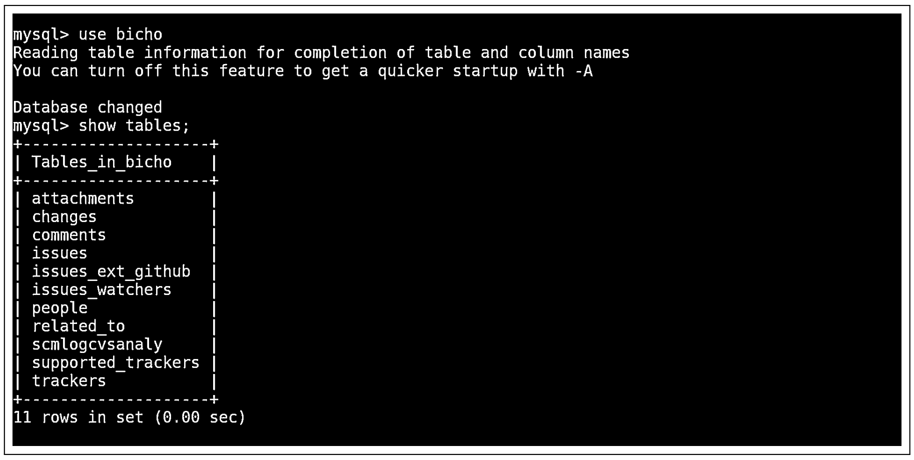
3.4.2 Description of Bicho schema

The structure of Bicho is designed with ten (10) entities (tables), and it has a set of core tables, used by all the back ends, and extended tables, particularly to each BT system.

The most up-to-date version of Bicho is designed with ten tables. Also, in each table there exist data fields and data types where the BT data extracted by Bicho is held in a MySQL database. Figure 2.9 depicts the Bicho database schema.

We will discuss the selected and identified tables – *issues, trackers and people* – produced by Bicho and considered relevant in order to narrow down and carefully examine the issue of how and where to identify BT data of OSS projects in Bicho. Table 3.7 presents a list of tables that exist in the Bicho database.

1. The **issue** table holds the information for each ticket – that is to say, each bug reported by a user or a developer. The information includes the time when it was opened, the reporter, the opener, summary, and description of the problem encountered, as well as the current status, priority and assigned developer. The table has 12 data fields

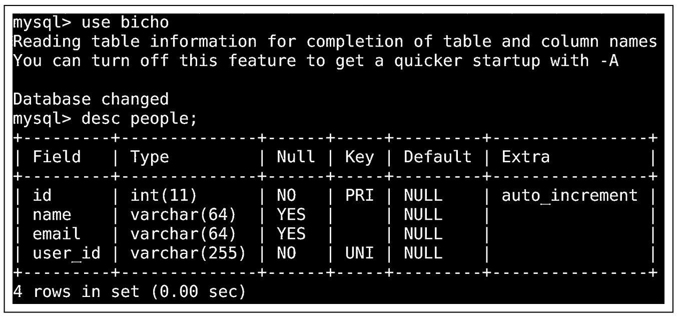


```
mysql> use bicho
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+--------------------+
| Tables_in_bicho    |
+--------------------+
| attachments        |
| changes            |
| comments           |
| issues             |
| issues_ext_github  |
| issues_watchers    |
| people             |
| related_to         |
| scmlogcvsanaly     |
| supported_trackers |
| trackers           |
+--------------------+
11 rows in set (0.00 sec)
```

Table 3.7: The tables in Bicho database

holding relevant information regarding BT data, such as the **tracker__id, issue, status, assigned__to, submitted__by**, and so on. Thus, the Issues table perfectly matches and resembles the ***SCMlog*** table. Because the ***SCMlog*** table data field can be linked to the ***Issues*** data field, as visible in Figure 3.12, the corresponding data fields are linked in Bicho and CVSAnalY. In general, we can trace VC logs in ***SCMlog*** tables in CVSAnalY, as well as trace BT data in the ***Issues*** tables in Bicho. As a result, we can also connect VC logs and BT data of both Bicho and CVSAnalY, since the tools are designed with the relevant data structure. Therefore, ***SCMlog*** tables and ***Issues*** tables are the entities considered for the automation, integration and synchronisation of BT data and VC logs in this research.

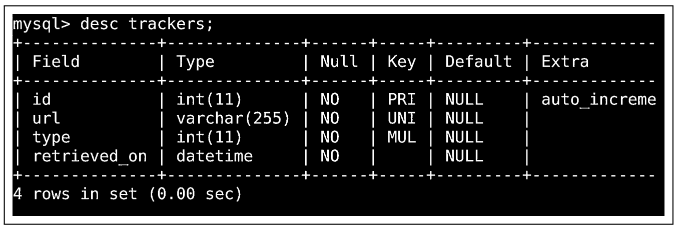
2. The **people** table in Bicho holds the information regarding each developer, like the list of people who participated in the ticketing process. However, in CVSAnalY there is a ***people*** table which holds the **id**, **name** and **email** address of each developer who has made changes to any bug report or feature enhancement in the project. These tables are not considered because we are only interested in VC logs and BT data held by these tools (Bicho and CVSAnalY). Table 3.8 is the description of the people table in the CVSAnalY database.



```
mysql> use bicho
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> desc people;
+---------+--------------+------+-----+---------+----------------+
| Field   | Type         | Null | Key | Default | Extra          |
+---------+--------------+------+-----+---------+----------------+
| id      | int(11)      | NO   | PRI | NULL    | auto_increment |
| name    | varchar(64)  | YES  |     | NULL    |                |
| email   | varchar(64)  | YES  |     | NULL    |                |
| user_id | varchar(255) | NO   | UNI | NULL    |                |
+---------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Table 3.8: People table in Bicho database

3. The **trackers** table in the Bicho database holds information pertaining to the OSS projects from which Bicho retrieved BT data. This included information like the ID of the project, the URL, type and the date it was retrieved. Similarly, CVSAnalY has the same tables, called ***repositories***, which hold the same information regarding the VC logs of OSS projects extracted by CVSAnalY, also with a similar data field. In this way, the only observation in this table is that one cannot query the date a project was retrieved, though it is relevant. Thus, only **id**, **uri**, **type** and the **name** of the project can be queried. Table 3.9 shows the description of the trackers table in the Bicho database.



```
mysql> desc trackers;
+--------------+--------------+------+-----+---------+-------------
| Field        | Type         | Null | Key | Default | Extra
+--------------+--------------+------+-----+---------+-------------
| id           | int(11)      | NO   | PRI | NULL    | auto_increme
| url          | varchar(255) | NO   | UNI | NULL    |
| type         | int(11)      | NO   | MUL | NULL    |
| retrieved_on | datetime     | NO   |     | NULL    |
+--------------+--------------+------+-----+---------+-------------
4 rows in set (0.00 sec)
```

Table 3.9: Trackers table in Bicho database

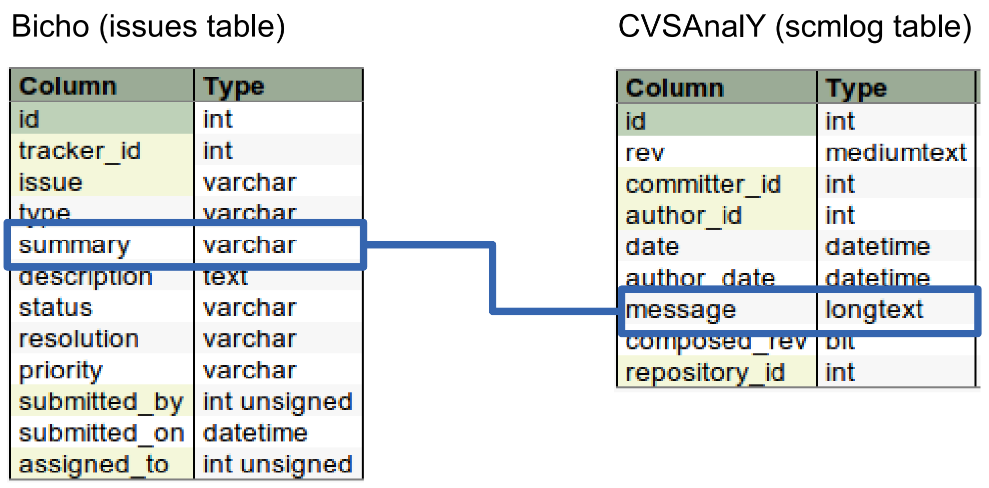
3.5 Locating VC logs and BT Data

In this section, the logic of **how** to retrieve BT data and VC logs we implement the full SZZ algorithm [116]. In its formulation, the algorithm should look for the terms "Bugs" or "Fixed" (case insensitive) in message logs, along with the “#" sign, which shows the ID of a bug. We demonstrate how to locate or identify the bugs that have been addressed both in CVSAnalY and Bicho, by issuing a number preceded by a “#" sign in the following subsections.

3.5.1 Locating bugs in VC logs

The first step was to store the VC logs via the CVSAnalY tool set. Among the tables generated by CVSAnalY, we specifically queried **SCMlog** table, which holds the number and unique IDs of changes in the VC 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 table 3.10 shows the composition of the CVSAnalY table that was used for the extraction of the information referring to bugs.

In order to identify or locate the bugs in VC 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 VC logs also mention 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". This step is also integrated in the tool chain.


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

Table 3.10: Corresponding fields linked in Bicho and CVSAnalY

3.5.2 Locating bugs in BT data

To locate or identify bugs in BT data, we used the Bicho tool to obtain and store all the information contained in the bug trackers of the projects contained in the sample, as well as all the issues reported by the users of a project and confirmed as such by developers. One of the tables created by Bicho is 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. This step is also integrated in the tool chain.

3.6 Worked Example: Brackets project

In this section, we use the *Brackets*[9] project as an example study. This is intended to illustrate the two aspects that the thesis will address in the rest of the chapters: (i) how to identify or locate bugs in VC logs, and (ii) how to detect the discrepancies between VC logs and BT data. The same methodology used in the worked example was used in our extended study to analyse the 344 OSS projects that were sampled and obtained from GitHub via FLOSSmole in this thesis. The *Brackets* project is a "code editor for the web". *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 88 releases have been published.

In the next subsections, we will discuss how to identify bugs in VC logs. Also, we will discuss how to identify and quantify the discrepancies between the bug sets from VC logs and BT data.

3.6.1 Identifying bugs in VC logs

This step involved the cleaning and 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

[9]`https://github.com/adobe/brackets`

log imported with CVSAnalY, produces a large number of false positives. Furthermore, we manually check whether the message field in the SCMlog table of CVSAnalY contains a reference with a "#" sign. A typical example would be for the #3057 bug of the Brackets project during the pilot study. The 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 in- formation in SCMlog referring to the actual 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.

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 in brackets, 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 [130]. In this thesis, we specifically focused on quantifying the bugs/issues, and the logs in Bicho and CVSAnalY that are not linked to bug fixes. Table 3.10 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 are summarised in the Venn diagram in Figure 3.11.

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.

3.6.2 Brackets Project: Discrepancies between the bug sets from VC logs and BT data

This section shows **how we evaluate the discrepancies** between BT systems and VC logs. The bugs from the Brackets VC logs were extracted using the approach highlighted in Subsection 3.5.1; the bugs from the Brackets BT data were extracted using the approach shown in Subsection 3.5.2. A Venn diagram (Figure 3.11) is used to show the subsets of bugs in the *Brackets* project.

As is visible, the number of bug IDs that were found in **both** CVSAnalY and Bicho is around one third (i.e., 167 bug IDs, which 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 third of the bug IDs are only found in Bicho, while the rest of the 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, because our aim was to identify and quantify discrepancies between the two sources in recording and in developer information about bugs.

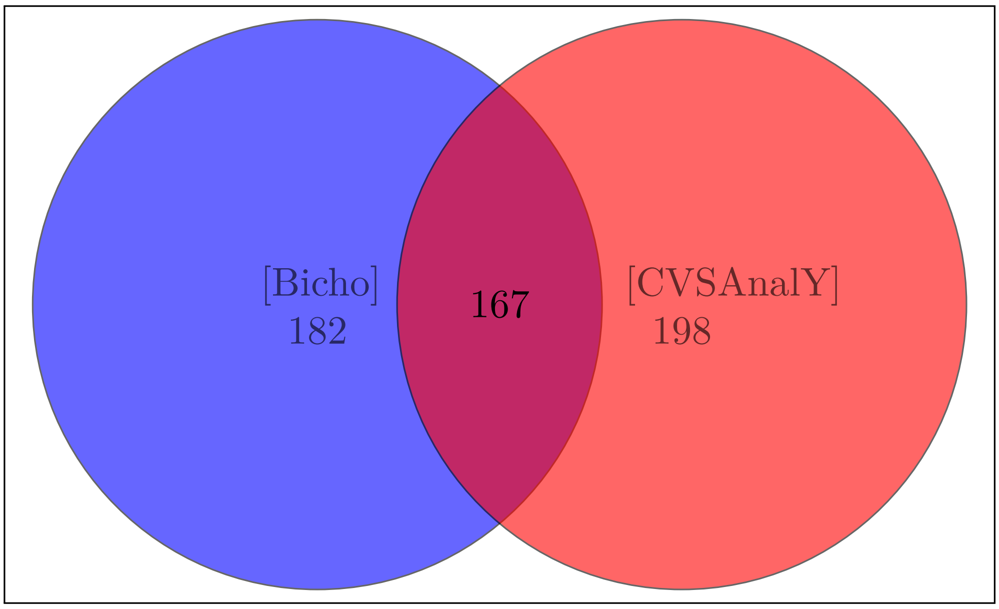


Figure 3.11: Intersection of BT data and VC logs in Bicho and CVSAnalY database for Bracket project

However, the extended study of 344 OSS projects analyses discrepancies between the bug sets from VC logs and BT data. This was carried out using *set theory*, by evaluating the union and intersection of the sets per project. Given a set of bug IDs mentioned in the VC logs, 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. In addition, this was integrated into the tool chain developed for this research (i.e., the thesis).

3.7 Automating and filling the missing data

Observing the tables of Bicho and CVSAnalY (displayed in table 3.10) and their attributes, we propose to use BT data in either database to synchronise 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 synchronise the *summary* and other attributes in the Bicho database. For the purpose of replicating this approach, the full process of tool set-up, data extraction, and automation and synchronisation of the missing data is detailed in the steps below:

1. **Installing Bicho** is easy. It can be installed through a terminal – that is to say, command line prompt. In our case, we installed the tool on Ubuntu. The installation is done using the command *python setup.py install* on the terminal. The complete guide on how to install Bicho can be found here: `http://metricsgrimoire.github.io/Bicho/`.

2. **Installing CVSAnalY** can also be done via a command line prompt (Terminal). Some dependencies need to be installed before installing CVSAnalY. This include Git and RepositoryHandler[10] Once the required dependencies have been installed, CVSAnalY can be installed by running setup.py script via the terminal, like *python setup.py install*. The complete guide on how to install CVSAnalY can also be found at `\http://metricsgrimoire.github.io/CVSAnalY/`.

3. **Run CVSAnalY and Bicho** for one specific project:

[10] A python library for handling code repositories through a common interface.

The two commands that are used for running CVSAnalY and Bicho from the command line (and tailored for the Brackets case) are as follows:

```
‘bicho −−db−user−out=[DB USER] −−db−password−out=[DB PASS] −−db−
    database−out=[DB NAME] −b github −u https://github.com/adobe/brackets/
    issues/ −−backend−user=[GITHUB USER] −−

‘bicho −−db−user−out=root −−db−password−out=YourDB−Password −−db−
    database−out=bicho −b github −u \"https://github.com/adobe/brackets" −−
    backend−user=GitHub−UserName −−backend−password=Your−GitHuB−
    Password −−debug‘;


```

A *Perl* script was developed to join the processes of executing CVSAnalY and Bicho and to extract BT data and VC logs. The algorithm (or pseudocode) illustrated in Algorithm 1 further illustrates the step-by-step procedure we followed to automatically check out the VC logs and BT data of the worked example (i.e., Brackets) as well as of all the 344 OSS projects we sampled in this research (refer to Appendix B in Section A.1 for the *Perl script*).

The algorithm shows that, for every project, the BT data on VC logs and tracker issues are retrieved from the **SCMlog** and the appropriate database tables produced, as linked above in table 3.10. In the case of the GitHub repository, this process is replicable to any other project by replacing the name of the project in the `https://api.github.com/repos/$project//issues` URL to retrieve the BT data (line 27 of the script located in A.1), and replacing the name of the project in the `https://github.com/$project/` URL to retrieve the VC logs (line 20 of the script located in Section A.1).

4. **Find discrepancies in the data sets**: Once the data in the two databases had been obtained and stored, we applied the SZZ algorithm to identify the missing BT data in the **SCMlog** Table of CVSAnalY and the **Issues** Table of Bicho and vice versa.

Algorithm 1 VC logs and BT system data extraction pseudocode

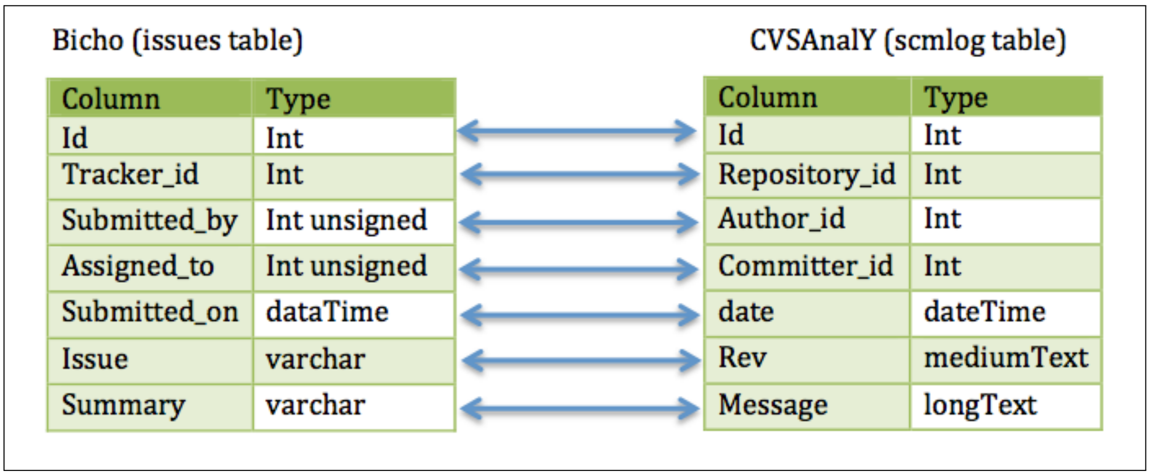
```
1: Read input File
2: for <each line> do                                    ▷ Until end of text file
3:     procedure CHOMP($a, b, c$)                         ▷ Remove trailing and leading strings
4:     end procedure
5:     svn_line ← url
6:     project ← url
7:     PRINT “Executing project A using cvsAnaly"
8:     procedure EXEC($svn\_line$)                        ▷ Extracting VC logs from VC system
9:         Store into db
10:    end procedure
11:    while $timer \neq 0$ do                            ▷ Pause for 15$s$
12:    end while
13:    PRINT “Executing project A using Bicho"
14:    procedure EXEC($project$)                          ▷ Extracting BT data from BT system
15:        Store into db
16:        rm                                             ▷ Clear cache
17:    end procedure
18: end for
```

Using sets of bug IDs, we automatically compared and produced a joint list of bug IDs, and classified them as “missing from the Bicho database”, “lacking in the CVSAnalY database”, or “present in both”.

In the Brackets case, we found 198 bug IDs that were missing in the issue archives, but present in the CVSAnalY database; we also found 182 bug IDs that were in the Bicho database, but not present in the CVSAnalY database. Finally, 167 bug IDs were present in both sets.

5. **Integration of table entries**: In the cases where one bug ID was missing from either database, we proposed using 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 and 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. The full list of matching fields of the

two tables is reported in Chapter 6.8.



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

Table 3.12: Corresponding fields linked in Bicho and CVSAnalY

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 the “Tracker_id" stored (also sequentially) by Bicho. In the *Brackets* case, CVSAnalY stored the log data in our database with a Repository_id=333, while Bicho stored the issues for the same project with a Tracker_id = 333. An extra table might be created to link the two IDs in the databases automatically.

All the fields of CVSAnalY or Bicho from table 3.10 have been mapped to connect the corresponding attributes in both tools. Table 3.12 shows the corresponding fields that have been linked to fill up the missing bug data in either database.

3.8 Summary of the chapter

This chapter presented the procedure we used to extract, compare and synchronise semi-automatically the gaps discovered in either the VC logs or the BT data of OSS projects. We showed that such an approach has been partially automated when partially implementing a well-known algorithm to isolate the bug-fixing commits (i.e., the SZZ algorithm [116]).

Chapter 4

Locating bugs in VC Logs

4.1 Introduction

In the previous chapter we presented the methodology that will be applied in the rest of the work. In this chapter, we will focus specifically on the VC logs, and on various issues that were found when extracting bug information from these data sources.

In this research, VC logs have been searched for bugs in basically two ways: (i) by using keywords such as “Fixed" or “Bug" [91]; 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 [30] [42] [75][116]. The SZZ algorithm is an example of an approach that combines keywords and proxies to detect bug-fixing commits.

Traceability links are needed to perform various software evolutionary activities, for instance to design and build defect prediction models [134]. However, the available tools to document VC logs lack integration [42]. As a result, two independent sets of bug-related data are produced, filling different databases [80] [109]. It has been suggested that using the bug IDs from VC logs could help to identify and recover missing traceability links [51] [85] [10]. These logs need to be manually or semi-automatically analysed and compared, to determine if VC logs and IDs from BT systems are referring to the same set of bug IDs, or if they refer to disjoint sets.

In this analysis 10 OSS projects from GitHub were analysed to pilot and show an approach to extract the complete 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 chapter, 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.

4.2 Definitions

The analysis performed below is an attempt at evaluating 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) to check their precision and recall when isolating the bug IDs in the VC logs.

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

As illustrated above, we partitioned the SZZ algorithm in three components, based on the keywords used. Therefore, given each keyword or identifier, precision is defined as

$$Precision_{\#,p} = \frac{TP_{\#,p}}{TP_{\#,p} + FP_{\#,p}} \tag{4.1}$$

$$Precision_{bug,p} = \frac{TP_{bug,p}}{TP_{bug,p} + FP_{bug,p}} \tag{4.2}$$

$$Precision_{fix,p} = \frac{TP_{fix,p}}{TP_{fix,p} + FP_{fix,p}} \tag{4.3}$$

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

$$Recall_{\#,p} = \frac{TP_{\#,p}}{TP_{\#,p} + FN_{\#,p}} \tag{4.4}$$

$$Recall_{bux,p} = \frac{TP_{bug,p}}{TP_{bug,p} + FN_{bug,p}} \tag{4.5}$$

$$Recall_{fix,p} = \frac{TP_{fix,p}}{TP_{fix,p} + FN_{fix,p}} \tag{4.6}$$

When considering the “Fix" and “Bug" keywords, similar definitions to the ones above apply. All the items were manually checked for the projects composing the sample, and the precision and recall of each are summarised in Table 4.6 (in Chapter 4.4).

4.3 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* project, which we used and showed a worked example of in Chapter 3.6.

4.3.1 Obtaining the complete set of bug IDs

At first, we retrieved all the bug IDs contained in the BT system of this project, and we created the first set (S1), containing over 4,000 bug IDs; after, 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 (VC logs). Only 3,117 logs were obtained when querying the VC 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 source.

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 VC 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 VC logs that were not reported in the BT system. On the other hand, the VC 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 VC logs (development logs) – is 267. Using the "Bug" and "Fixed" keywords also produces further results, as summarised in Table 4.1 below.

Table 4.1: 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

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 VC 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 in this chapter.

4.3.2 Evaluating the precision of each SZZ component

In the second step of our evaluation, we performed a manual analysis of a random sample of 100 VC 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 # identifier are referring to a bug. Regarding the # symbol, we found 58 logs (out of 100) that mentioned #: after a close inspection, we realised that 57 of these 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 don’t have a unique ID attached (i.e., the false negatives). From Table 4.2, 4.3 and 4.4 we present the number of logs that were referring to TP, FP, FN and TN for the *Brackets* project and the remaining nine OSS projects. Given the formulas above, 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 log mentioning “Bug" also referred to a bug ID (i.e., TP), while three 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:

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \tag{4.7}$$

The # symbol gained an F-measure of 0.726, the “Fix" keyword 0.582 and the “Bug"

#Symbol						
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 4.2: Number of logs that were referring to TP, FP, FN and TN for # symbol

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 4.3: Number of logs that were referring to TP, FP, FN and TN for Fixed

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

Table 4.4: Number of logs that were referring to TP, FP, FN and TN for 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. Analysing the unstructured data of the VC 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 traceability purposes.

These findings, if confirmed, will re-enforce 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.

4.4 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 4.5. 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 4.6 below. As also performed in the worked example above, each individual 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 VC 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

Table 4.5: Attributes of the projects selected

S/N	Project Name	URL	Commits	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

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 VC logs (development logs). It is an important finding: VC 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' VC logs; and (ii) the time it took to analyse and detect the presence of bug IDs of all 344 sampled projects.

In terms of the size of the projects' VC 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 VC 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 VC logs for every project in the 344 OSS projects for this research. As a result, the replication of large OSS projects was extended semi-automatically using the tool chain to evaluate the precision of each SZZ component. This will be detailed in the next section of this chapter.

The process followed to extract the precision and recall data was similar to the pilot study: the VC 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

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

Table 4.6: Manual evaluation of 10 OSS projects precision and recall of the three main components of the SZZ algorithm

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.

4.5 Trade-off between recall and precision

The trade-off between precision and recall in the context of this thesis occurs with an increased proportion of # symbol precision, leading to a 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 a 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 4.1 below), the proportion of the three main components of the SZZ algorithm (i.e., # symbol, fixed and bug) were zero (also visible in Table 4.6) because none of the logs retrieved in that project referred to the TP and FP as defined in Section 4.2 of this Chapter. 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 component of the SZZ algorithm.

Previous studies by Buckland et al [23] and Gordan et al [50], regarding the origins of the recall and precision trade-off assumed 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 4.1, and an appropriate balance between the precision and recall of the three main component of the SZZ algorithm evaluated using 10 OSS projects.

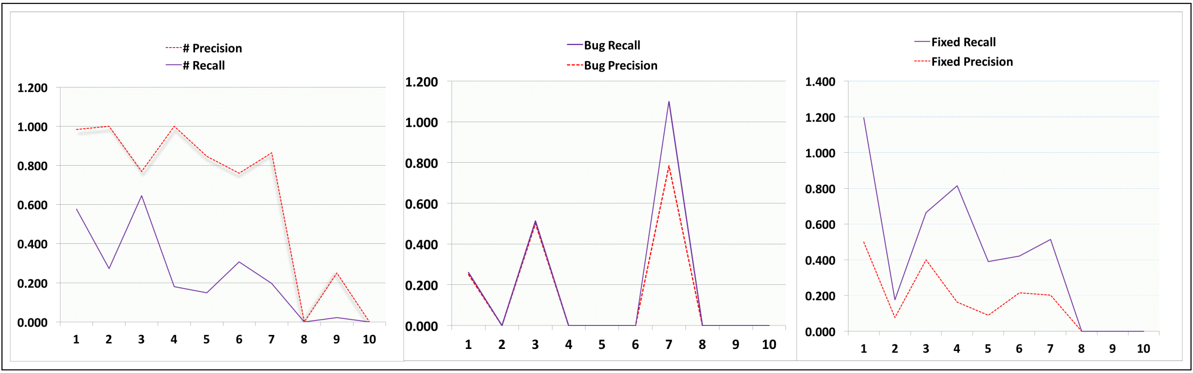


Figure 4.1: Precision and Recall curve of three main component of the SZZ algorithm

4.6 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, recall and F-measure results for each project when using the individual components of the SZZ algorithm of all 344 OSS projects.

In addition, the result of each component was further examined by applying a Mann–Whitney test to further unveil the significance of using each component and prove the scalability of the approach [127]. Similarly, the results of the replication of the 344 OSS projects are shown in Table 4 in Appendix A.7. Also, each individual 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 4.2 of this chapter.

Moreover, the results are reported in table 4 in Appendix A. Section A.7 were 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 4.7 below using WESSA[1]

But in this case, the results varied significantly, considering that the analysis of the 10 OSS projects was carried out manually. Also, in some OSS projects only the top 100 subsets of VC logs were considered when evaluating each components, while the rest of the 10 OSS projects had fewer than 100 VC logs. However, where the proportion of the three main components of the SZZ algorithm (i.e., # symbol, fixed and bug) were zeros from **Table 4** in Appendix A. Section A.7. None of the logs retrieved in that project referred to the TP and FP as mentioned in the previous Section 4.5 and defined in Section 4.2 of this Chapter.

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 VC logs. Iteratively, this is an important finding: VC logs are clearly lagging behind in terms of completeness and traceability, as compared to the BT data.

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									

Table 4.7: Statistical significance of the SZZ algorithm (p-value)

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

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

The matrix table in Table 4.7 shows the significance of the three main components of SZZ algorithm. This was evaluated using the precision, recall and F-measure against each individual 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 VC logs of the projects were extracted semi-automatically using the tool chain by issuing the following SQL query.

```sql
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 worrying for the purpose of bug traceability.

4.7 Summary of the chapter

This chapter outlined an approach to building a complete set of bug IDs that were documented in the evolution of a software system. This comprises the analysis and parsing of both the VC systems and the BT systems: this is required because we found that, commonly, OSS projects hold different sets of bug IDs when interrogating the BT system and the VC system. In addition, the chapter presented an in-depth analysis of the SZZ algorithm, which has been used extensively by researchers to track the bug-fixing commits of software systems. We partitioned the algorithm into 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 VC logs. We found that the guideline of using the # symbol and the bug ID largely outperforms the other proxies to detect bug-fixing commits. 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 VC 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 VC logs, into the BT system. Furthermore, we demonstrated that the process of collecting data related to bugs, when using open-source projects, is far from established or repeatable. 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. The results in this chapter 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 VC logs with the BT data by using the standard # notation for bug IDs.

Chapter 5

Discrepancies between the bug sets from VC logs and BT data

This chapter describes how we analysed the data from BT data sets and VC logs to find discrepancies between the sets of bug IDs cited. The chapter is structured as follows[1]: Section 5.1 presents the introduction. We discuss the background in Section 5.2. Section 5.3 presents the results of 344 OSS projects of our empirical study; the shared bug coverage of 344 OSS projects that we observed in four scenarios in this research is discussed in Section 5.4. We present worked examples of the four scenarios that we observed among the 344 OSS projects in Section 5.5. Section **??** presents our discussion, and the conclusion to this chapter is provided in Section 5.6.

5.1 Introduction

In the previous chapters, we learned that over the past two decades, software engineering researchers have shown significant interest in the analysis and use of empirical data. Open-source software (OSS) projects provide a large amount of process and product data, and several tools are available to mine and analyse this data. As mentioned, utilising this vast amount of data can benefit both OSS and commercial projects: mining and analysing software

[1]Some contents of this chapter have been published in the EASE 2015 proceedings [108]

artefacts – like code, design documents, requirements or bug issues – can offer fundamental contributions for empirical software engineering research. Using the right set of bug IDs, BT data can be used to design models for predicting software faults and software reliability; faults and reliability of a software artefact can also be linked to who, when and how changes were made to it. Similarly, the analysis of VC logs can give important insights into the underlying software quality, by focusing on software developers and their actions and efforts in order to build cost-estimation models. Such logs can also be used for detecting bug-fixing actions, improving bug-prediction techniques and increasing software quality and reliability [134].

In this chapter we argue that the BT and VC data sets are complementary: the extraction of both VC logs and BT data sets requires tools that (i) automatically mine software projects (or software repositories) and (ii) store the extracted data in specifically built databases, for posterior analysis. Data includes not only source code, but also metadata, such as logs, dates and types of actions performed on specific software artefacts. Developers in OSS communities use these tools as a medium of collaboration and communication, such as reporting bugs or mentioning changes that occur as a result of a bug fix, as well as revising all the commits to a software artefact [57].

Combining the two sets of bug-related data (VC logs and BT data) is related to the traceability of bugs within software development. In this chapter we illustrate why bug traceability is complicated by the fact that the sources of bug data are often not in sync or complete [58].

5.2 Background

As discussed in Chapter 3.4, bug-tracking systems such as Bugzilla are the most commonly used in the OSS community to keep track of bugs, features and enhancements. All the BT data that exist inside a BT system are explicitly identified with a hash symbol followed by a number (e.g., #1234), which is the bug ID. In this case, BT systems store, for each tracked bug, its life cycle, with an indication of a “fixed”, “open”, “closed”, “resolved” or “new” status. On the other hand, source code management systems like Control Version System (CVS) are where the tracking of software maintenance and corrective activities on OSS projects is kept.

In this way, VC logs and BT data are recorded inside VC systems and BT systems respectively, in an unstructured way and on different platforms. Thus, it is not possible, for example, to jointly parse an SQL query on a BT system and a VC system to analyse and understand the evolution of OSS projects as well as the action of developers in bug fixes. In this case, Bicho and CVSAnalY tools are designed and developed to bridge this gap – that is, to enable researchers and practitioners in software engineering to retrieve and store VC logs and BT data in a relational database for posterior analysis.

All changes or VC logs are retrieved and stored in the **SCMlog** table (of CVSAnalY database) in the ***message*** text field as single entries.

Each entry might contain various data, including the developer who made the changes, a text message referring to the reasons for the commit, and the list of features added to a piece of a component in the system as well as the date it was added. The BT data are also retrieved and stored in the **Issues** table (of Bicho database) in the ***Summary*** text field as single entries. Similarly, each entry might contain various data, such as the bug ID, which developer was assigned to fix the bug, a summary of the problems reported and encountered by the user, and the date it was reported.

Tools such as CVSAnalY and Bicho are designed to bridge the gap, by allowing researchers to effectively retrieve and record VC logs and BT data in a structured way as well as to store them in their localised databases for posterior analysis. Unfortunately, using the tools (i.e., Bicho and CVSAnalY), one is not able to collectively parse a single query to retrieve or analyse and understand the evolution of a given OSS project as well as the action of developers in bug fixes regarding VC logs and BT data. Thus, there is a need to cross-analyse VC logs stored in CVSAnalY with BT data stored in Bicho in order to obtain and mirror correct BT data and VC logs, matching bugs with the changes related to bug fixes. One way is to analyse the CVS log messages (i.e. the **SCMlog** table in CVSAnalY in the ***message*** text field), to identify commits related to bugs that were fixed. In addition, in cases where VC logs and BT data are missing and recovered, there is a need to synchronise the recovered data automatically to have complete sets of data for empirical studies and other software corrective maintenance activities.

In this research, we first analysed and identified the **SCMlog** table and the text that exists in the ***message*** field in the CVSAnalY database for VC logs, looking for bug IDs. For example, we used keywords such as "#1234", "Fixed #1234" or "Bug #1234". In this study we are only interested in the number of bug IDs reported in the BT system (i.e. in the Bicho Issues table) and the bug IDs appearing in the **SCMlog** table mentioned in CVSAnalY. This is because we want to quantify the discrepancies in the traceability of VC logs and BT data. By that means, we can perform an empirical study with a large number of OSS projects, which consists of thousands of VC logs and BT data from open-source software repositories.

Norman [39] hinted that many empirical studies in the past were conducted with very small systems; in other words, they did not scale up to large systems. In this way, there might still be a minor gap to be filled in the area of traceability issues related to VC logs and BT data in software engineering. A typical example is the empirical study of Bachmann et al [12] with one OSS project. This might not be enough to establish a "ground truth" that VC logs are not mirrored when linked with BT data in OSS software projects. There are hundreds of thousands of OSS projects existing in only one software repository (i.e., GitHub).

The rationale for the empirical study in this chapter of 344 OSS software projects sampled randomly on GitHub is to establish reasonable evidence in relation to traceability links recovery and synchronisation of bug-related data from open-source software repositories.

Several approaches have been proposed in the literature, as discussed in Chapter 3. They are related to recovering traceability links between design and implementation. The study of Murphy et al [94] reported on the software reflexion models to match a design expressed in the Booch notation against its C++. In addition, regular expressions are used and applied in naming conventions and mapping source code model entities onto high-level model entities [125] [4].

Similarly, other techniques have been proposed that use data mining on source code management systems (CVS) [43]; [44]; [135]; [140]. These researchers are among the pioneers to first exploit release data to identify logical coupling between entities. Moreover, they suggest the use of CVS history to detect fine-grained logical coupling between functions, classes and files. Their proposed techniques investigate the historical development of classes, analysing the time when a new class is added onto the system and when an existing class is changed. Similarly, these consist of attributes regarding the changes of an entity, like the author, the date and so on. All this information, including the addition of features enhancement, is analysed in order to reveal evolutionary facts and changes made in different parts of the system in the software development process [44]; [135].

5.3 Results - Overall sample of OSS projects

In this section, we report the analysis of the sample of 344 OSS projects from GitHub. In particular, we report on how many bug IDs are mentioned in the two databases for each project. The two overarching hypotheses that we planned to verify in this research are:

1. bug-related data stored in the issue trackers should be considered as complete; and
2. bug-related data is common and shared in both VC logs and issue trackers.

In order to summarise the findings from the 344 OSS projects, we produced a box plot in 5.1 to display the shared bug coverage (SBC) ratio, defined as follows:

$$SBC = \frac{BugIDs(VCS) \cap BugIDs(BTS)}{BugIDs(VCS) \cup BugIDs(BTS)} \tag{5.1}$$

where BugIDs (VCS) is the set of bug IDs as found in the VC logs (of any project), and BugIDs (BTS) is the set of bug IDs from the issue trackers (of the same project). This ratio was evaluated for each project, and the values were always in the $[0...1]$ interval. We present the results using the box plot, which is considered to be an excellent tool for illustrating the variation in as well as the location of information in large data sets, particularly for detecting and conveying the location and variation of changes between different kinds of data sets in groups [129]. In addition, it is useful for describing the behaviour of the random size of population and distribution at different stages, either at the beginning or the end of the distributions. Similarly, the outliers in the box plot are points that indicate values that are outside the lower and upper distribution of different types of data sets [27].

Table 5.1 presents an excerpt of the results, such as:

- **All in Bicho**: In this set of operations we obtained all the BT data IDs in the Bicho **Issues** tables which might be traced in the CVSAnalY database.
- **All in CVSAnalY**: In this column we present all the VC log IDs in the CVSAnalY **SCMlog** table which might be traced in Bicho.
- **Intersection**: In this column we present all the BT system and VC log IDs present in both the Bicho and CVSAnalY databases – in other words, the IDs of each project that the tool chain was able to trace in both tools (i.e., the common IDs, or the intersection, in Bicho and CVSAnalY).
- **Only in Bicho**: In this column we present all the unique BT data IDs in the Bicho database that are not traced in the CVSAnalY database.
- **Only in CVSAnalY**: In this column we present the unique VC log IDs in the CVS-AnalY database that are not traced in the Bicho database.
- **The union**: This column presents the total number of VC logs and BT data retrieved by the tool chain in the respective Bicho and CVSAnalY databases.
- **Shared bug coverage**: This column presents the ratio of bug coverage in the Bicho and CVSAnalY databases for each project, defined as follows:

  $$SharedBugCoverage = \frac{Union}{intersection} \tag{5.2}$$

  Where the union is the total number of bug IDs retrieved by the tool chain in the respective Bicho and CVSAnalY databases, and the intersection is the common IDs also tracked in the Bicho and CVSAnalY databases.

The results presented in Table 5.1 are an excerpt from the 344 OSS projects evaluated for each project. Refer to Appendix A in Section A.6 for the remaining results of the 334 OSS projects sampled in this research. MySQL dump of Bicho and CVSAnalY database which holds the BT data and VC logs of 344 OSS projects can be found on Figshare.[2]

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

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
1	57	6	6	51	0	57	0.105
2	449	19	19	430	0	449	0.042
3	790	30	30	760	0	790	0.037
4	213	15	14	199	1	214	0.065
5	6	0	0	6	0	6	0
6	101	21	20	81	1	102	0.196
7	18	0	0	18	0	18	0
8	1459	218	202	1257	16	1475	0.136
9	34	2	2	32	0	34	0.058
10	2	1	0	2	1	3	0

Table 5.1: Metrics evaluated for the bug sets from BT data and VC logs (an excerpt from 344 OSS projects)

As visible from the box plot in Figure 5.1, (the Shared bug coverage evaluated per projects presented in Appendix A. Section A.6 for all the 344 OSS projects was used to produce the box plot in Figure 5.1) the set of common bug IDs is in general very low: in around 75% of the projects the common IDs (i.e., the intersection of the sets) is no more than 20% of the overall number of detected bug IDs (i.e., the union of the sets). This could mean that one of the two databases (either VC logs or BT data) contains most of the information on bug IDs, and that information is not mirrored in the other database. It could also mean that there is a common subset of bug IDs, but that most of the other IDs are not shared in the two information sources.

The box plot in Figure 5.1 shows that both Bicho and CVSAnalY have a significant effect on missing data with respect to both VC logs and BT data of all 344 OSS projects we sampled in this research. Thus, the gaps between the different parts of the outliers in the box plot show the degree of dispersion and skewness in the VC logs and BT data, which are not always in sync in both Bicho and CVSAnalY databases. In addition, the outliers in the box plot it indicate 20% of the OSS projects intersection of bug IDs in Bicho and CVSAnalY database.

In addition, the Chord diagram in Figure 5.2 visualises the inter-relationships between BT data and VC logs of 344 OSS projects in Bicho and CVSAnalY respective database. The connections between nodes arrange in circle such as **All in bicho**, **All in CVSAnalY**, **Intersection**, **Only in Bicho**, **Only in CVSAnalY** and the**Union**, displays the proportions of Bug IDs for all 344 OSS projects sampled in this research.

The node is connected to each other represented proportionally by the size of each arc in colours. In this way, the total number of Bug IDs in general for all the 344 OSS projects is

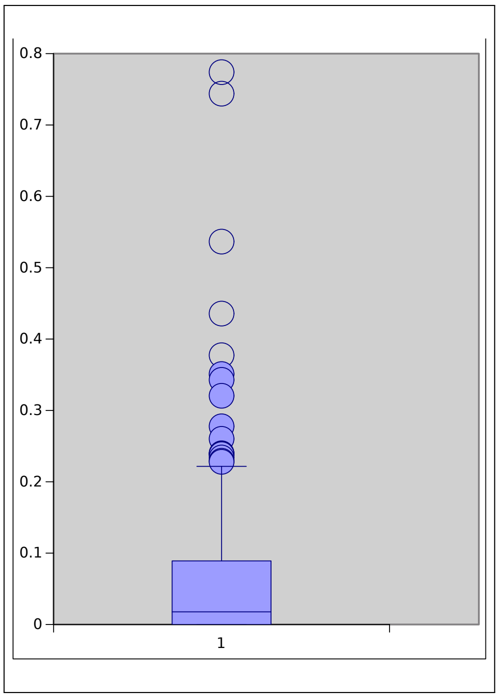


Figure 5.1: Ratio of bug IDs mentioned in both development logs and bug trackers,per project

211,457 (i.e., the **union** which is highlighted in the *red arch*). While the common Bug IDs detected in Bicho and CVSAnalY, is highlighted in the *lime green arch* (i.e., the **intersection** is 12,194). Thus the size of the *lime green arch* in the chord diagram represent a small proportion of bug IDs shared in the node **Only in CVSAnalY** and **All in CVSAnalY**. However, the size of the *lime green arch* in the chord diagram is bigger in the node **Only in Bicho** compared to the node **Only in CVSAnalY** because the proportion of bug IDs detected in Bicho overall are much higher than the bug IDs in CVSAnalY. As we mentioned previously, the set of common bug IDs of 344 OSS projects is in general very low compared with the total number of bug IDs detected in Bicho and CVSAnalY respective databases as visualise in the chord diagram (in Figure 5.2) in the *red arch* (i.e., the **union**) and the *lime green arch* (i.e., **intersection**).

In order to further describe the skewness of VC logs and BT data of the OSS projects we analysed in this Chapter, in the Section 5.4, we formulate four scenarios of bug coverage in the Bicho and CVSAnalY databases.

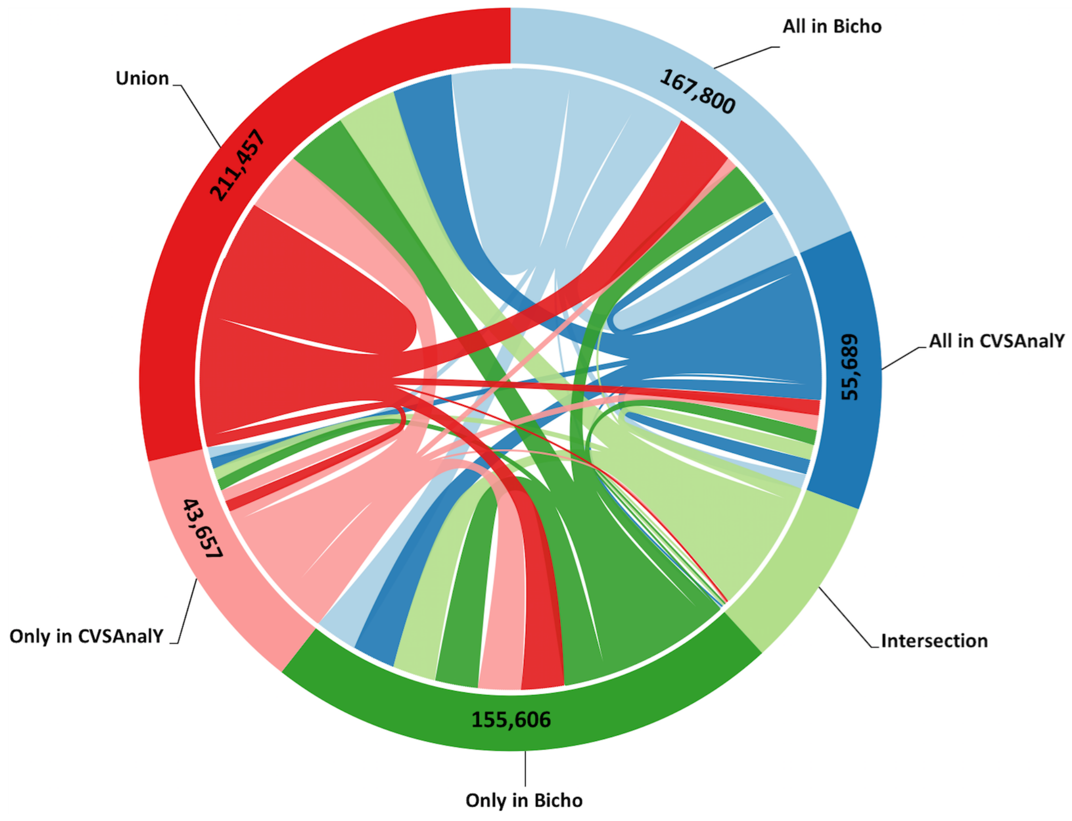


Figure 5.2: BT data and VC logs of 344 OSS projects

5.4 Scenarios of bug coverage

Depending on the configuration found for the two sets of bug IDs, four scenarios can be expected for a software project: they are depicted graphically in the next subsection. In addition, Table 5.1 shows the total number of OSS projects that comply with each scenarios. The projects are further described in the next subsection (refer to appendix A in Section A.8 for the complete 344 OSS projects that comply with each scenario per project)

5.4.1 Scenario 1

The first scenario that we observed was when the set of bug IDs found in the issue tracker database had no intersection with the set of bug IDs coming from the VC logs. We observed this scenario in 25 projects: for the majority of these projects, one of the sets of bug IDs was found in Bicho and CVSAnalY but the intersection of the sets was empty. Figure 5.3 depicts scenario 1 graphically.

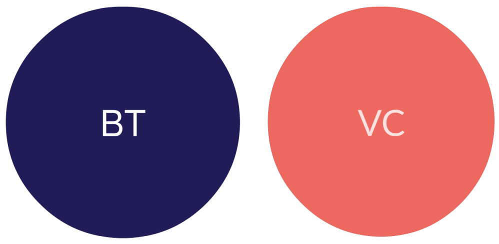


Figure 5.3: Scenario one

5.4.2 Scenario 2

The second scenario was the most common: there was a subset of bug IDs that was common to the two data sources (i.e., the intersection of the sets). Apart from the common IDs, there was also (i) one subset of bug IDs that appeared only in the VC logs, and (ii) another subset of bug IDs that appeared only in the bug-tracking system. Figure 5.4 depicts scenario 2 graphically.

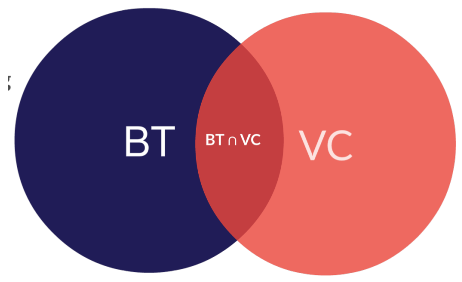


Figure 5.4: Scenario two

5.4.3 Scenario 3

The third scenario that we observed was when all the bug IDs of either of the sets were contained within the other set: in the theory of sets, the cardinality of the union of the sets was the cardinality of the containing set, while the cardinality of the intersection of the sets was the cardinality of the contained set. We found 129 projects that complied with this scenario, which is depicted graphically in Figure 5.5. This represents 42% of the total projects sampled in this research, in which one of the bug ID sets was a subset of the other: for 103 projects, the bug IDs found in the VC logs were a subset of what was found in the BT data. In a further 26 projects, the opposite was the case: the bug IDs found in the BT system were just a subset of what was found in the development logs.

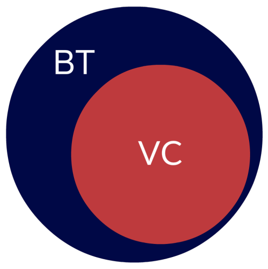


Figure 5.5: Scenario three

5.4.4 Scenario 4

The final scenario was when all the bug IDs were found in the BT system and the development log: in the set theory language, the union of the sets was equal to the intersection of the sets. This was the ideal scenario, because the bugs were being mirrored exactly in the two databases: unfortunately, we observed this scenario in only 8 projects out of 344, and in all these cases the sets of bug IDs from both development logs and bug-tracking issues were empty.

Scenarios	**Number of OSS projects**
1	25
2	182
3	129
4	8
Total number of OSS: 344	

Table 5.2: Total number of OSS projects comply with four scenarios

It is worth noticing that Scenario 4 has very few projects: this would be the ideal situation, when BT data and VC logs are perfectly aligned. In reality this the case, therefore it makes our approach quite meaningful, in the case of OSS projects.

5.5 Worked example: four scenarios

In this section, we further investigate and conduct an in-depth analysis of 10 OSS projects identified in each scenario that we observed in the previous section. The OSS projects were randomly selected. The analysis will provide an insight into the discrepancies that we found between BT data and VC logs of OSS projects sampled in this research, which fall under four different scenarios. In order to provide a comparison between each scenario, the bug IDs of OSS projects discovered in both tools will be examined by measuring their metrics. This includes the number of developers making commits or changes in the VC logs, the total lines of code, and the number of revisions, as well as the total number of files existing in each OSS project.

In the next subsection we will present the values measured across 10 projects in each scenario.

The StatSVN [3] tool was used to produce the metrics in the tables below [72].

The complete statistical report on the 10 OSS projects for all the scenarios was also evaluated using StatSVN. The full statistical report of the 37 total number of OSS projects evaluated for all four scenarios is available on Figshare.[4]

5.5.1 Worked Example – Scenario 1

In this scenario, we analysed 10 OSS projects with the aim of discovering the hidden factor behind the set of bug IDs found in the issue tracker database that had no intersection with the set of bug IDs coming from the VC logs. In the terminology of the set theory, and using BT as the set of bug IDs from the bug trackers, and VC as the set of bug IDs from the development logs:

$$\mathrm{BT} \cap \mathrm{VC} = \emptyset \wedge \mathrm{BT} \not\equiv \mathrm{VC}$$

For the 10 OSS projects reported in Table 5.3, as we mentioned, one of the sets of bug IDs is empty either in Bicho or CVSAnalY, and therefore the intersection of the sets would always be empty. We recall in Section 5.3 of this chapter that report on the number of bug IDs detected in the two databases. In this way, for the 10 OSS projects analysed in this scenario is presented in Table 5.4. The results is an excerpt from the finding of bugs IDs detected: all in Bicho (i.e., in column BT - VC of Table of 5.4), only in Bicho (i.e., in column BT), and all in CVSAnalY (i.e., in column VC - BT), only in CVSAnalY (i.e., in column VC) as well as the Intersection of bug IDs in both tools was evaluated per project.

The second column of Table 5.3 shows the project IDs randomly selected from the 344 OSS projects. The column represents the same tracker (i.e., trackers table) IDs as those in the Bicho database and the same repository (i.e., repositories table) IDs as those in the CVSAnalY database. The third column of Table 5.3 presents the revision of each project as calculated using the StatSVN tool. The *average* number of revisions for the OSS projects in scenario 1 is 653. Project ID=25 has 86 revisions and 5 developers, which is the lowest number of revisions

[3]StatSVN retrieves information from a Subversion repository and generates various tables and charts describing software development project: `http://www.statsvn.org`

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

in this scenario and Project ID=192 has 1 developer which is the lowest number of developers among the 10 OSS projects we observed in this scenario. On the other hand, project ID=207 has the highest number of revisions with 68 developers.

Scenario 1					
S/N	**Project IDs**	**No. Revision**	**Total Files**	**No. Developers**	**Total LOC**
1	10	903	267	29	34607
2	25	86	38	5	203066
3	30	262	5,290	6	2,657,423
4	34	775	475	27	25,052
5	42	1,637	336	201	89,826
6	47	166	46	6	3,740
7	192	88	11	1	884
8	205	317	1978	34	47,298
9	207	1929	159,219	68	159,219
10	239	365	210	34	92,769

Table 5.3: 10 OSS projects observed in scenario 1

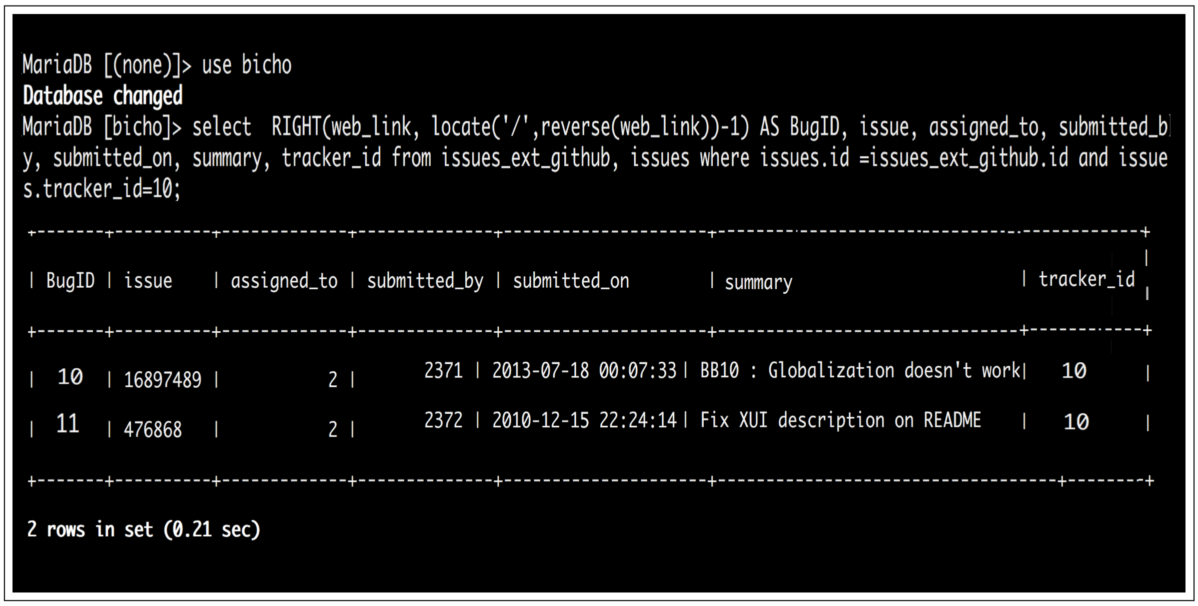
Scenario 1						
		Bicho			CVSAnalY (VC)	
S/N	Project IDs	**BT**	**BT - VC**	**BT ∩ VC**	**VC - BT**	**VC**
1	10	2	2	0	1	1
2	25	11	11	0	3	3
3	30	7	7	0	6	6
4	34	139	139	0	2	2
5	42	52	52	0	202	202
6	47	26	26	0	37	37
7	192	1798	1798	0	5	5
8	205	71	71	0	1	1
9	207	109	5	0	5	109
10	239	1	1	0	18	18

Table 5.4: Metrics evaluated for the bug sets from BT data and VC logs (excerpt)

We present a worked example of what we observed in this scenario on project ID=10. BT data was traced in Bicho, and the VC logs traced in CVSAnalY found. However, the intersection of the sets on this project is empty. Figure 5.6 depicts a typical example of BT data traced in phonegap project [5] (with project ID 10): this project shows there is no common

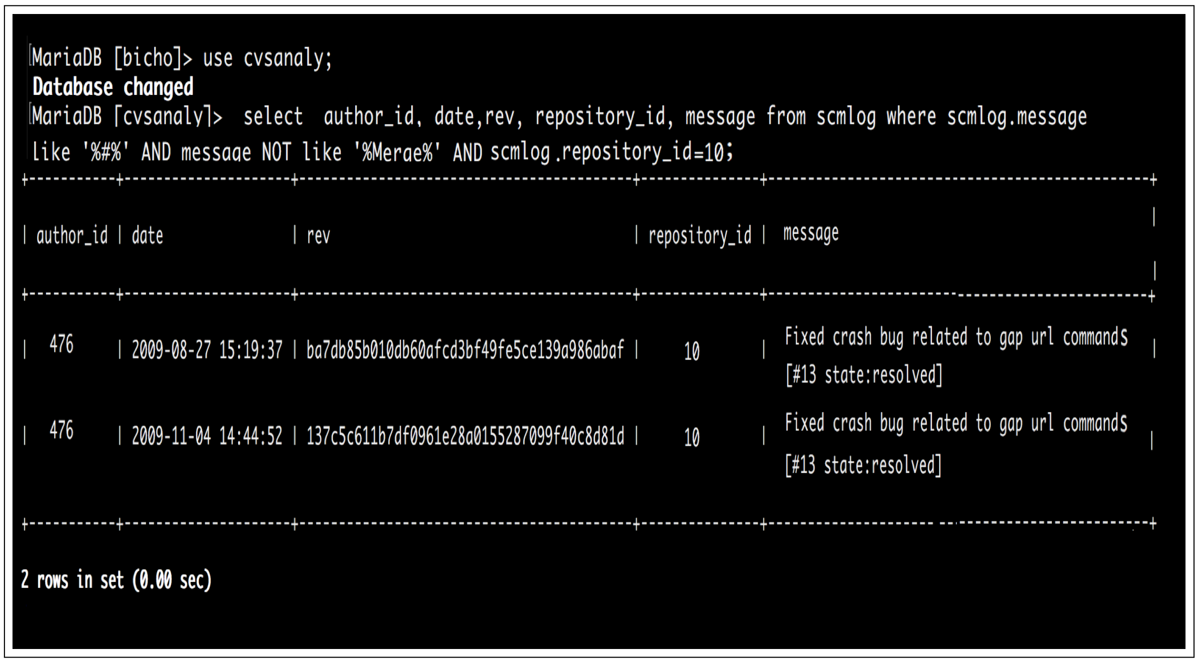
[5]The phonegap project is among the 344 OSS projects we sample on GitHub for this study. https://github.com/sintaxi/phonegap.gita

bug ID traced in CVSAnalY and Bicho, as depicted in figure 5.7.



```
MariaDB [(none)]> use bicho
Database changed
MariaDB [bicho]> select  RIGHT(web_link, locate('/',reverse(web_link))-1) AS BugID, issue, assigned_to, submitted_b
y, submitted_on, summary, tracker_id from issues_ext_github, issues where issues.id =issues_ext_github.id and issue
s.tracker_id=10;
+-------+----------+-------------+--------------+---------------------+------------------------------+------------+
| BugID | issue    | assigned_to | submitted_by | submitted_on        | summary                      | tracker_id |
+-------+----------+-------------+--------------+---------------------+------------------------------+------------+
|  10   | 16897489 |           2 |         2371 | 2013-07-18 00:07:33 | BB10 : Globalization doesn't work|   10   |
|  11   | 476868   |           2 |         2372 | 2010-12-15 22:24:14 | Fix XUI description on README |   10   |
+-------+----------+-------------+--------------+---------------------+------------------------------+------------+

2 rows in set (0.21 sec)
```

Figure 5.6: BT data in Bicho



```
MariaDB [bicho]> use cvsanaly;
Database changed
MariaDB [cvsanaly]>  select  author_id, date,rev, repository_id, message from scmlog where scmlog.message
like '%#%' AND message NOT like '%Merge%' AND scmlog.repository_id=10;
+-----------+---------------------+------------------------------------------+---------------+-------------------------------------------+
| author_id | date                | rev                                      | repository_id | message                                   |
+-----------+---------------------+------------------------------------------+---------------+-------------------------------------------+
|   476     | 2009-08-27 15:19:37 | ba7db85b010db60afcd3bf49fe5ce139a986abaf |      10       | Fixed crash bug related to gap url commandS |
                                                                                              [#13 state:resolved]
|   476     | 2009-11-04 14:44:52 | 137c5c611b7df0961e28a0155287099f40c8d81d |      10       | Fixed crash bug related to gap url commandS |
                                                                                              [#13 state:resolved]
+-----------+---------------------+------------------------------------------+---------------+-------------------------------------------+

2 rows in set (0.00 sec)
```

Figure 5.7: No bug IDs mirrored in CVSAnalY

The graph in 5.8 summarises the results of the findings on discrepancies between VC logs and BT data held by Bicho and CVSAnalY respective databases for the 10 OSS project in scenario 1.

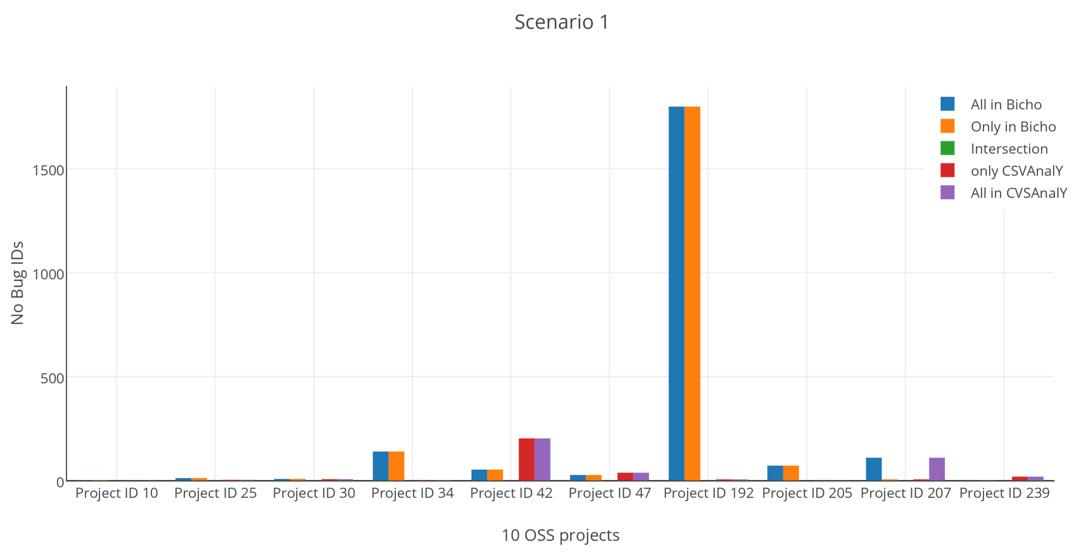


Figure 5.8: Scenario 1: 10 OSS projects

The numbers along the Y-axis in Figure 5.8 represent the number of bug IDs detected in the Bicho and CVSAnalY databases.

The X-axis features one legend to represent the OSS projects IDs in scenario 1, the legend on the right corresponds to four sets of bug IDs detected in Bicho and CVSAnalY: All in Bicho, Only in Bicho, All in CVSAnalY, Only in CVSAnalY and Intersection.

In addition, the graph in Figure 5.8 displays the proportion of bug IDs detected in both tools (i.e., Bicho and CVSAnalY), which indicates the number of bug IDs is by far higher in Bicho than in the CVSAnalY, overall 70% of bug IDs in Bicho for all the 10 OSS projects in this scenario are not mirrored in CVSAnalY. For instance, project ID=192 has 1798 bug IDs found in Bicho. Unfortunately, none of the 1798 bug IDs are ever mentioned in the CVSAnalY database (i.e., VC logs) while only 5 bug IDs were not mentioned in Bicho database. Thus, clearly VC logs are lagging behind, as mentioned in Chapter 1. In reality, VC logs should form a superset of all BT data: one would expect the data contained inside the BT system to be mirrored in the VC system and developers to record and distinguish between their development and bug-fixing actions. Thus, there exist discrepancies in the data sets (i.e., VC logs and BT data of OSS projects we sampled in this research) held by Bicho and CVSAnalY.

As a result, using our approach all the missing VC logs can be circumvented in either Bicho or CVSAnalY (i.e., all the VC logs found only in CVSAnalY database will be converted automatically to entries into the Bicho database; and all the BT data found only in Bicho will be converted into entries to the CVSAnalY database). This will be demonstrated in the next Chapter 6 of this thesis.

5.5.2 Worked Example – Scenario 2

In this scenario, we also analysed 10 OSS projects in order to identify a subset of bug IDs that is common to the two data sources (i.e., the intersection of the bug IDs). In the terminology of the set theory, and using BT as the set of bug IDs from the bug trackers, and VC as the set of bug IDs from the development logs:

$$\mathrm{BT} \cap \mathrm{VC} \neq \emptyset \land \mathrm{BT} \not\equiv \mathrm{VC}$$

We recall in Section 5.3 of this chapter that report on the number of bug IDs detected in the two databases. In this way, for the 10 OSS projects analysed in this scenario is presented in Table 5.6. The results is an excerpt from the finding of bugs IDs detected: all in Bicho, only in Bicho and all in CVSAnalY, only in CVSAnalY as well as the Intersection of bug IDs in both tools was evaluated per project.

The second column shows the project IDs, which represent the same tracker (i.e., trackers table) IDs as those in the Bicho database and also the same repository (i.e., repositories table) IDs as those in the CVSAnalY database. The third column of Table 5.5 presents the revisions of each project, calculated using the StatSVN tool. The *average* number of revisions of the OSS projects in scenario 2 is 1,537. This indicates that a significant number of the OSS projects in this scenario had a large number of revisions. Project ID=338 has 225 revisions and 28 developers, which is the lowest number of revisions and the lowest number of developers among the 10 OSS projects we observed in scenario 2. Project ID=78 has the highest number of revisions, and 54 developers.

Scenario 2					
S/N	**Project IDs**	**No. Revision**	**Total Files**	**No. Developers**	**Total LOC**
1	4	559	159	74	16,665
2	39	2,097	765	108	56,102
3	179	906	45	73	11,708
4	243	1,856	425	310	34,959
5	266	535	123	82	48,916
6	338	225	300	28	531,337
7	339	1,203	1,996	56	175,184
8	340	2,877	851	53	1,010,484
9	341	712	509	45	13,542
10	78	4,404	380	54	345,474

Table 5.5: 10 OSS projects observed in scenario 2

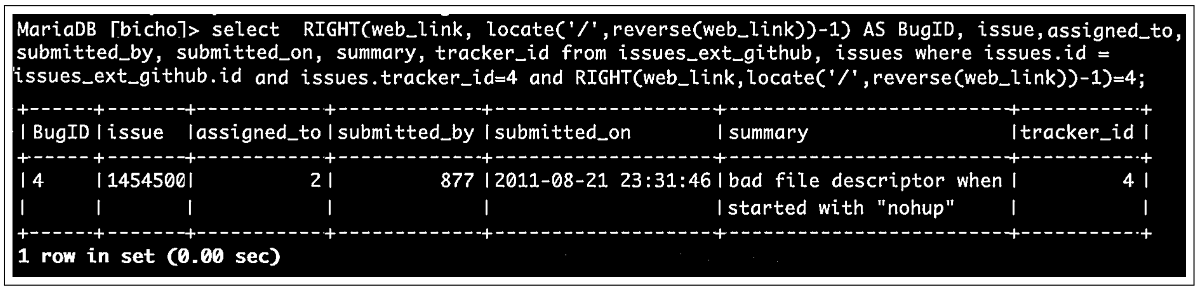
Scenario 2						
		Bicho			CVSAnalY (VC)	
S/N	Project IDs	**BT**	**BT - VC**	**BT ∩ VC**	**VC - BT**	**VC**
1	4	199	213	14	15	1
2	39	840	945	105	122	17
3	179	371	440	69	82	13
4	243	147	209	62	125	63
5	266	38	41	3	23	20
6	338	104	85	19	39	20
7	339	141	162	21	43	22
8	340	630	668	38	40	2
9	341	157	163	6	9	3
10	78	407	667	260	282	22

Table 5.6: Metrics evaluated for the bug sets from BT data and VC logs (excerpt)

Similarly, we present a worked example of what we observed in this scenario. In the 10 OSS projects that we identified in this scenario, bug IDs were traced in Bicho and CVSAnalY. This was observed in the Spark [6] project (project ID=4 in Table 5.5). Figure 5.9 depicts a typical example of BT data traced in the Spark OSS project and bug IDs mirrored in CVSAnalY, as depicted in Figure 5.10. In this case, the common bug IDs are not synchronised into the

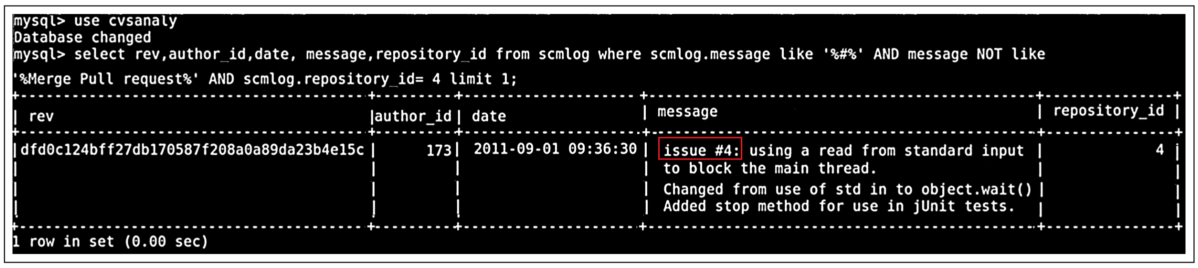
[6] The Spark project is among the 344 OSS projects sample on GitHub for this study: https://github.com/perwendel/spark:

respective Bicho and CVSAnalY databases.



```
MariaDB [bicho]> select  RIGHT(web_link, locate('/',reverse(web_link))-1) AS BugID, issue,assigned_to,
submitted_by, submitted_on, summary, tracker_id from issues_ext_github, issues where issues.id =
issues_ext_github.id and issues.tracker_id=4 and RIGHT(web_link,locate('/',reverse(web_link))-1)=4;
+------+-------+-----------+------------+-------------------+-----------------------+----------+
| BugID | issue  |assigned_to |submitted_by |submitted_on        |summary                 |tracker_id |
+------+-------+-----------+------------+-------------------+-----------------------+----------+
| 4     |1454500|          2 |         877 |2011-08-21 23:31:46 |bad file descriptor when |         4 |
|       |       |            |             |                    |started with "nohup"    |           |
+------+-------+-----------+------------+-------------------+-----------------------+----------+
1 row in set (0.00 sec)
```

Figure 5.9: BT data in Bicho



```
mysql> use cvsanaly
Database changed
mysql> select rev,author_id,date, message,repository_id from scmlog where scmlog.message like '%#%' AND message NOT like
'%Merge Pull request%' AND scmlog.repository_id= 4 limit 1;
+-----------------------------------------+---------+-------------------+---------------------------------------------+--------------+
| rev                                      |author_id| date               | message                                      | repository_id |
+-----------------------------------------+---------+-------------------+---------------------------------------------+--------------+
|dfd0c124bff27db170587f208a0a89da23b4e15c |      173| 2011-09-01 09:36:30 | issue #4: using a read from standard input |             4 |
|                                          |         |                    | to block the main thread.                   |               |
|                                          |         |                    | Changed from use of std in to object.wait() |               |
|                                          |         |                    | Added stop method for use in jUnit tests.   |               |
+-----------------------------------------+---------+-------------------+---------------------------------------------+--------------+
1 row in set (0.00 sec)
```

Figure 5.10: Bug IDs mirrored in CVSAnalY

The graph in 5.11 summarises the results of the findings on discrepancies between VC logs and BT data held by Bicho and CVSAnalY respective databases for the 10 OSS project in scenario 2.

The numbers along the Y-axis in Figure 5.11 represent the number of bug IDs detected in the Bicho and CVSAnalY databases.

The X-axis features one legend to represent the OSS projects IDs in scenario 2, the legend on the right corresponds to four sets of bug IDs detected in Bicho and CVSAnalY: All in Bicho, Only in Bicho, All in CVSAnalY, Only in CVSAnalY and Intersection.

In addition, the graph in Figure 5.11 displays the proportion of bug IDs detected in both tools in this scenario(i.e., Bicho and CVSAnalY), which indicates there are common (i.e., intersection of bug IDs) bug IDs for all the 10 OSS projects in Bicho and CVSAnalY database, In addition, 75% of the bug IDs in Bicho are much more than the bug IDs found in CVSAnalY for all the 10 OSS project in this scenario. For instance, project ID=39 and 340 have much

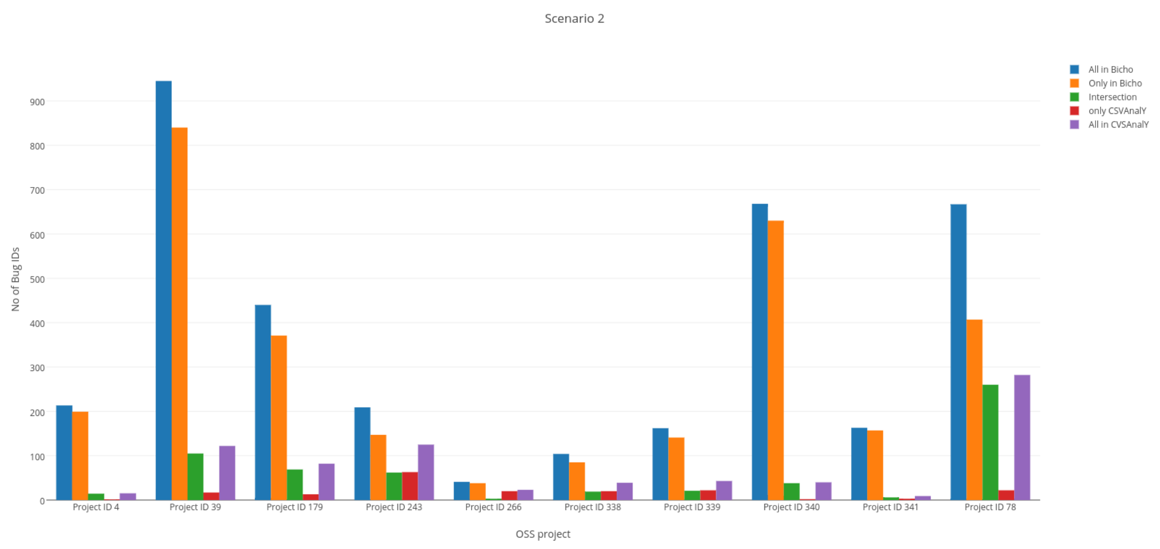


Figure 5.11: Scenario 2: 10 OSS projects

more bug IDs found only in Bicho. In this case, not all the bug IDs is detected in Bicho and CVSAnalY database. Thus, the missing bug IDs along with the meta data will be synchronise automatically into Bicho and CVSAnalY database vice versa.

5.5.3 Worked Example – Scenario 3

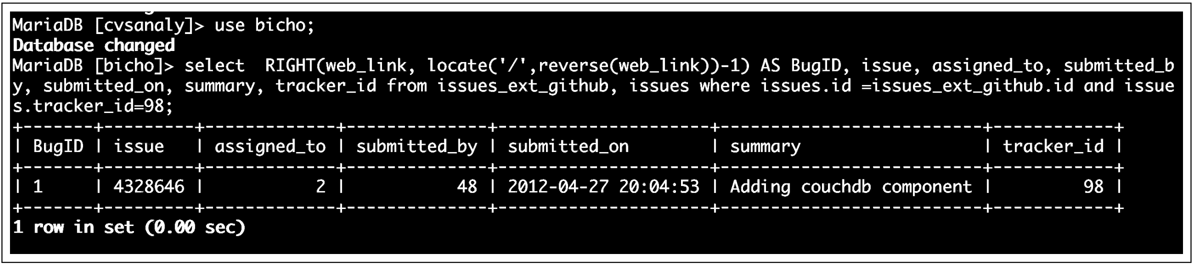
In this scenario, 10 OSS projects were analysed to identify when all the bug IDs of either of the sets were contained within the other set. In the terminology of the set theory, and using BT as the set of bug IDs from the bug trackers, and VC as the set of bug IDs from the development logs:

$$\mathrm{BT} \subseteq \mathrm{VC} \wedge \mathrm{BT} \cap \mathrm{VC} \neq \emptyset$$

We recall in Section 5.3 of this chapter that report on the number of bug IDs detected in the two databases. In this way, for the 10 OSS projects analysed in this scenario is presented in Table 5.8. The results is an excerpt from the finding of bugs IDs detected: all in Bicho, only in Bicho and all in CVSAnalY, only in CVSAnalY as well as the Intersection of bug IDs in both tools was evaluated per project.

The first column in Table 5.7 shows the project IDs, which also represent the same tracker (i.e., trackers table) IDs as those in the Bicho database and the same repository (i.e., repositories table) IDs as those in the CVSAnalY database. The second column of Table 5.7 presents the revisions of each project, calculated using the StatSVN tool. The *average* number of revisions of the OSS projects in scenario 3 is 497. This suggests that half of the OSS projects in this scenario have >100 revisions, while 2 projects in this scenario have <100 revisions. Project ID=46 has 2 revisions and 1 developer, which is the lowest number of revisions and the lowest number of developers in this scenario. On the other hand, project ID=275 has the highest number of revisions – 2,068 – and 108 developers.

In this scenario, bug IDs in the BT system were just a subset of what was found in the VC logs. This was observed in the Breze project[7] project. Figure 5.12 depicts a typical example of BT data traced in the Breze OSS project; 1 bug ID (i.e., BT data) was found in Bicho database as a subset of 15 VC logs only mirrored in CVSAnalY database in the project, as depicted in Figure 5.13.



```
MariaDB [cvsanaly]> use bicho;
Database changed
MariaDB [bicho]> select  RIGHT(web_link, locate('/',reverse(web_link))-1) AS BugID, issue, assigned_to, submitted_b
y, submitted_on, summary, tracker_id from issues_ext_github, issues where issues.id =issues_ext_github.id and issue
s.tracker_id=98;
+-------+---------+-------------+--------------+---------------------+--------------------------+------------+
| BugID | issue   | assigned_to | submitted_by | submitted_on        | summary                  | tracker_id |
+-------+---------+-------------+--------------+---------------------+--------------------------+------------+
| 1     | 4328646 |           2 |           48 | 2012-04-27 20:04:53 | Adding couchdb component |         98 |
+-------+---------+-------------+--------------+---------------------+--------------------------+------------+
1 row in set (0.00 sec)
```

Figure 5.12: BT data in Bicho

The graph in 5.14 summarises the results of the findings on discrepancies between VC logs and BT data held by Bicho and CVSAnalY respective databases for the 10 OSS project in scenario 3.

The numbers along the Y-axis in Figure 5.14 represent the number of bug IDs detected in the Bicho and CVSAnalY databases.

The X-axis features one legend to represent the OSS projects IDs in scenario 3, the legend

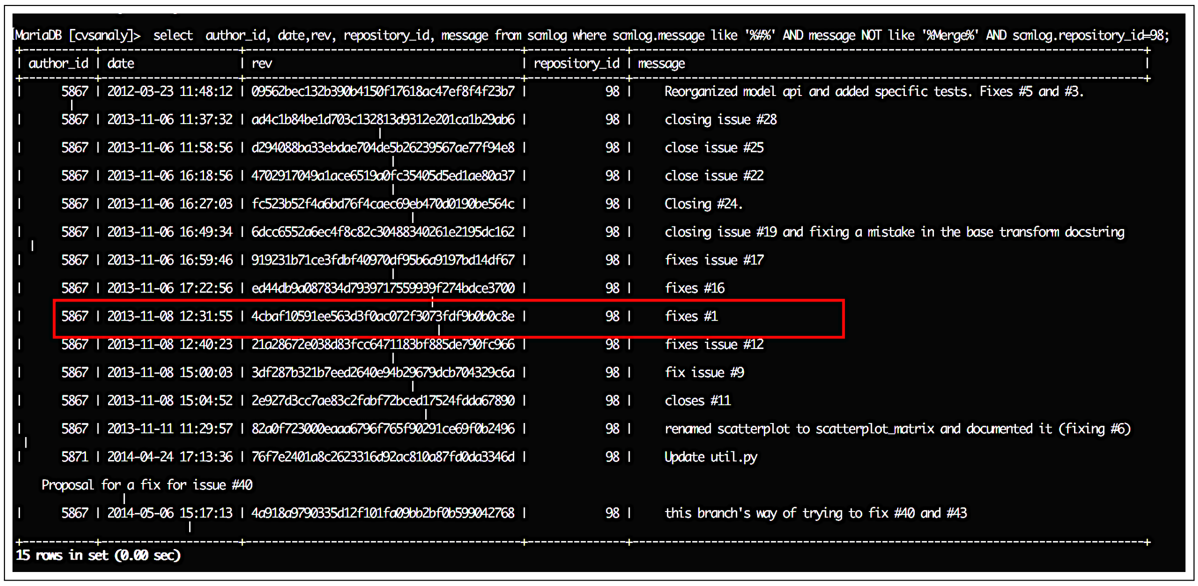
[7]The Breze project is among the 344 OSS projects we sample on GitHub for this study: https://github.com/breze-no-salt/breze.git

Scenario 3					
S/N	**Project IDs**	**No. Revision**	**Total Files**	**No. Developers**	**Total LOC**
1	1	129	38	17	5,812
2	46	2	1	1	4
3	9	112	29	10	9,379
4	137	253	106	55	40,935
5	290	90	35	18	7,376
6	275	2,068	352	108	30,614
7	293	162	11	14	2,025
8	278	148	21	10	5,224
9	98	1,240	237	10	54,399
10	299	765	2034	34	1,590,527

Table 5.7: 10 OSS projects observed in scenario 3

Scenario 3						
		Bicho			CVSAnalY (VC)	
S/N	Project IDs	**BT**	**BT - VC**	**BT ∩ VC**	**VC - BT**	**VC**
1	1	51	57	6	0	6
2	46	106	107	1	0	1
3	9	32	34	2	0	2
4	137	3044	3050	6	0	6
5	290	80	83	3	0	3
6	275	698	749	51	0	51
7	293	45	47	2	0	2
8	278	2550	2562	12	0	12
9	98	0	1	1	15	16
10	299	140	154	14	0	14

Table 5.8: Metrics evaluated for the bug sets from BT data and VC logs (excerpt)



```
MariaDB [cvsanaly]>  select  author_id, date,rev, repository_id, message from scmlog where scmlog.message like '%#%' AND message NOT like '%Merge%' AND scmlog.repository_id=98;
+-----------+---------------------+------------------------------------------+---------------+---------------------------------------------------------------------------------------+
| author_id | date                | rev                                      | repository_id | message                                                                               |
+-----------+---------------------+------------------------------------------+---------------+---------------------------------------------------------------------------------------+
|      5867 | 2012-03-23 11:48:12 | 09562bec132b390b4150f17618ac47ef8f4f23b7 |            98 |    Reorganized model api and added specific tests. Fixes #5 and #3.
|      5867 | 2013-11-06 11:37:32 | ad4c1b84be1d703c132813d9312e201ca1b29ab6 |            98 |    closing issue #28
|      5867 | 2013-11-06 11:58:56 | d294088ba33ebdae704de5b26239567ae77f94e8 |            98 |    close issue #25
|      5867 | 2013-11-06 16:18:56 | 4702917049a1ace6519a0fc35405d5ed1ae80a37 |            98 |    close issue #22
|      5867 | 2013-11-06 16:27:03 | fc523b52f4a6bd76f4caec69eb470d0190be564c |            98 |    Closing #24.
|      5867 | 2013-11-06 16:49:34 | 6dcc6552a6ec4f8c82c30488340261e2195dc162 |            98 |    closing issue #19 and fixing a mistake in the base transform docstring
|      5867 | 2013-11-06 16:59:46 | 919231b71ce3fdbf40970df95b6a9197bd14df67 |            98 |    fixes issue #17
|      5867 | 2013-11-06 17:22:56 | ed44db9a087834d7939717559939f274bdce3700 |            98 |    fixes #16
|      5867 | 2013-11-08 12:31:55 | 4cbaf10591ee563d3f0ac072f3073fdf9b0b0c8e |            98 |    fixes #1
|      5867 | 2013-11-08 12:40:23 | 21a28672e038d83fcc6471183bf885de790fc966 |            98 |    fixes issue #12
|      5867 | 2013-11-08 15:00:03 | 3df287b321b7eed2640e94b29679dcb704329c6a |            98 |    fix issue #9
|      5867 | 2013-11-08 15:04:52 | 2e927d3cc7ae83c2fabf72bced17524fdda67890 |            98 |    closes #11
|      5867 | 2013-11-11 11:29:57 | 82a0f723000eaaa6796f765f90291ce69f0b2496 |            98 |    renamed scatterplot to scatterplot_matrix and documented it (fixing #6)
|      5871 | 2014-04-24 17:13:36 | 76f7e2401a8c2623316d92ac810a87fd0da3346d |            98 |    Update util.py

   Proposal for a fix for issue #40
|      5867 | 2014-05-06 15:17:13 | 4a918a9790335d12f101fa09bb2bf0b599042768 |            98 |    this branch's way of trying to fix #40 and #43
+-----------+---------------------+------------------------------------------+---------------+---------------------------------------------------------------------------------------+
15 rows in set (0.00 sec)
```

Figure 5.13: Bug ID mirrored in CVSAnalY

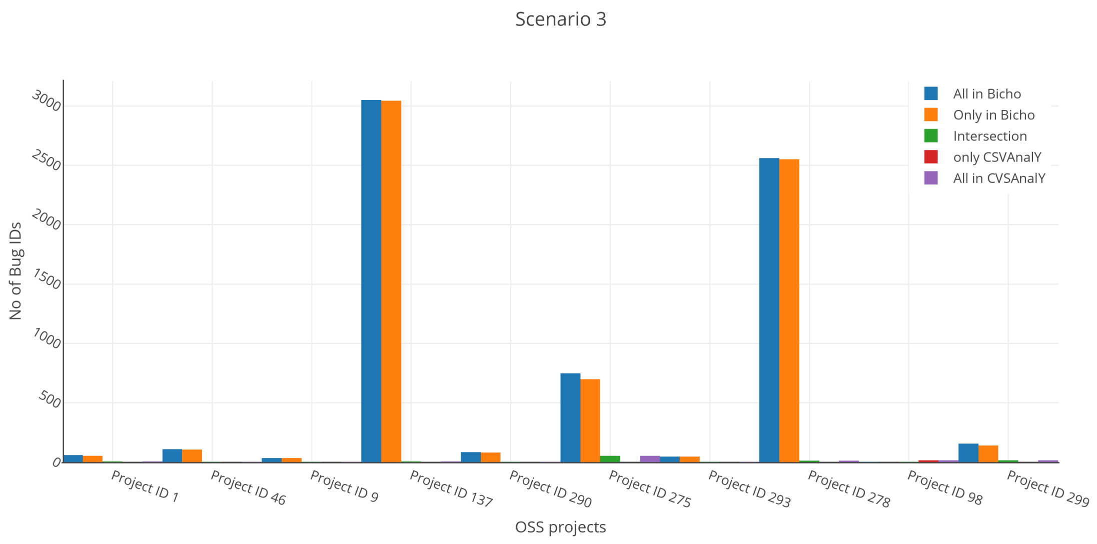


Figure 5.14: Scenario 3: 10 OSS projects

on the right corresponds to four sets of bug IDs detected in Bicho and CVSAnalY: All in Bicho, Only in Bicho, All in CVSAnalY, Only in CVSAnalY and Intersection.

In addition, the graph in Figure 5.14 displays the proportion of bug IDs detected in both tools in this scenario (i.e., Bicho and CVSAnalY databases), which indicates bug IDs found in the CVSAnalY database were a subset of what was found in the Bicho database. For instance, in project IDs=46, 137, 275 and 278, small number of bug IDs was found as a subset of BT data in Bicho database. However, 15 bug IDs in project ID=98 was found as a subset of VC logs in CVSAnalY database. In this case, bug IDs that were not found as a subset of the other (i.e., either in Bicho or CVSAnalY database) will be synchronises automatically into their respective databases.

5.5.4 Worked Example – Scenario 4

The final scenario was when all the bug IDs were found in the BT system and the development log. In the terminology of the set theory, and using BT as the set of bug IDs from the bug trackers, and VC as the set of bug IDs from the development logs:

$$\mathrm{BT} \equiv \mathrm{VC} \wedge \mathrm{BT} \cap \mathrm{VC} = \mathrm{BT} \cup \mathrm{VC}$$

This is the ideal scenario, when all the bugs IDs are being mirrored exactly in the two databases. We classified 8 projects into this scenario, since they comply with the set theory definition above. Unfortunately, these 8 projects have no big IDs

Below we provide an analysis of their characteristics, as done for this scenario previous 3 Scenarios. The second column in Table 5.9 shows the project IDs, which also represent the same tracker (i.e., trackers table) IDs as those in the Bicho database and the same repository (i.e., repositories table) IDs as those in the CVSAnalY database. The third column of Table 5.9 presents the revisions of each project, calculated using the StatSVN tool. The *average* number of revisions of the OSS projects in scenario 4 is 430. Project ID=271 has 32 revisions and 7 developers, which is the lowest number of revisions among the 10 OSS projects we observed in scenario 4, while project ID=130 has the lowest number of developers and 208 revisions. Project ID=89 has the highest number of revisions (1,970). The average number of developers for the OSS projects sampled in this scenario is 6.

Scenario 4					
S/N	**Project IDs**	**No. Revision**	**Total Files**	**No. Developers**	**Total LOC**
1	38	311	40	13	3,743
2	45	85	387	4	45,332
3	89	1,970	493	4	25,954
4	103	147	86	5	68,804
5	125	258	112	8	4,984
6	130	208	91	3	4,229
7	271	32	12	7	747

Table 5.9: 10 OSS projects observed in scenario 4

5.5.5 Worked example - Summary of the four scenarios

Table 5.10 summarises the findings for the four scenarios, showing an average of revisions, total files, number of developers and the total number of lines of code for the sample of 37 OSS projects we observed in the four scenarios.

Metrics were measured for the sample of 37 OSS projects using the StatSVN tool. The OSS projects in scenario 2 produced a significant number of metrics, including the highest number of revisions and the highest number of developers. Most of the common bug IDs detected in Bicho and CVSAnalY were, by far, in scenario 2. The metrics produced by the OSS projects in scenario 1 had the second-highest number of revisions and developers, and those in scenario 3 had the third highest. On the other hand, for most of the OSS projects we observed in scenario 4, bug IDs were empty in Bicho and CVSAnalY. Thus, most of the OSS projects in scenario 4 had the lowest number of revisions and the lowest number of developers among the 37 OSS projects we observed.

In general, Table 5.10 shows the mean (average) values evaluated for the 37 OSS projects in four scenarios we sampled out of 344 OSS projects. The results in the Table 5.10 indicates fewer revisions in software projects might result in fewer commits or changes in the software development process. Conversely, having fewer developers in the software development process might result in low commits and revisions. Depending on the experience of the developers

Scenario	No. Revision	Total Files	No. Developers	Total LOC
1	653	16,787	41	331,388
2	1,537	555	88	224,437
3	497	286	28	193,387
4	430	174	6	21,970

Table 5.10: Mean (average) values evaluated for the 37 OSS projects in four Scenarios

and the size of the project. This might affect the traceability of BT data and VC logs of the OSS projects we observed in the scenario where there is no intersection between the sets of bug IDs in the two tools, or where only a small number of common IDs were detected.

In this section, we learned that traceability links between VC logs and BT data are derived from changes in the source code [68]. Thus, the commits in the files might be the source of information available for recovering links between BT data and VC logs produced by the developers in the project. In this case, having a small number of revisions in a given OSS project might reduce the possibility of tracking BT data and VC logs in the OSS projects we sampled in this research.

5.6 Summary of the chapter

This chapter presented the results of an extended quantitative analysis of how bug-related data is stored in the VC logs and the BT data in a sample of 344 OSS projects. The set of bug IDs from the VC logs was compared to the set of bug IDs found in the BT systems. The objective of this research was to ascertain how much discrepancy is visible when considering these two sources of information, and whether either could be considered as a complete and credible set of data regarding bug issues.

We found that over half of the projects sampled have a portion of bug IDs mentioned in one source (either the development logs or the bug-tracking logs), but not in the other. We also found that the intersection of "common" or shared bug IDs is very low (around 20% for some 75% of the projects in the sample), while in some extreme cases projects held distinct sets of IDs in either database that were not shared between them.

Chapter 6

Automating and synchronising the missing data

In this section, we report in detail the proposed structure of the framework and the implementation.

This chapter is structured as follows. In Sections 6.1 and 6.2 we discuss the concept and the background for this chapter. Section 6.3 details the structure of the framework proposed. We discuss the implementation in Section 6.4. In Section 6.5 and 6.6 we discuss the quantification of VC logs and BT data as well as re-engineering the tool sets. We report on the synchronisation in Section 6.7. In addition, section 6.8 discuss re-aligning Bicho and CVSAnalY and finally provide a conclusion in Section 6.9.

6.1 Introduction

Open-source software project development data is stored in a VC system, which contains valuable information, such as the evolution of a software project. This information includes the history of the development process and information about developers who have contributed to producing the source code.

Developers in the open-source software (OSS) community derive benefits from VC systems by getting access to the copies of the source code of different software projects, and even by contributing to the source code in the repository, thereby learning and sharing knowledge of different types of software development processes. In addition, VC systems offer an interface for performing data analysis pertaining to the software project and conducting empirical studies in software engineering.

However, such data cannot be queried through the standard interface of the VC system. CVSAnalY and other related tools mentioned in Chapter 2.6 are designed to offer such functionality, with the limitation of not being able to synchronise the recovered missing VC logs.

Conversely, BT systems also hold crucial information regarding issues reported that may require the immediate attention of developers in OSS or commercial projects. In this way, getting access to BT systems is enabled via HTTP, and the issue reports can be viewed in HTML format and can also be retrieved in XML format. The retrieval of BT data is required for analyses and predicting future defects in the system. Similarly, before a developer can add or make changes in the source code, the primary source of information and guide they can refer to is the information stored in the BT system. In this way, the retrieval of issues reports is done using Bicho for such analysis by querying relevant entities that exist in the Bicho database.

The concept of the syncing process in this research is to provide an interface that enables practitioners and researchers in software engineering to cross-analyse and sync BT data and VC logs data automatically. In the case where discrepancies were discovered, syncing the recovered link between VC logs and BT data would be advantageous in the databases of either Bicho or CVSAnalY for posterior analyses.

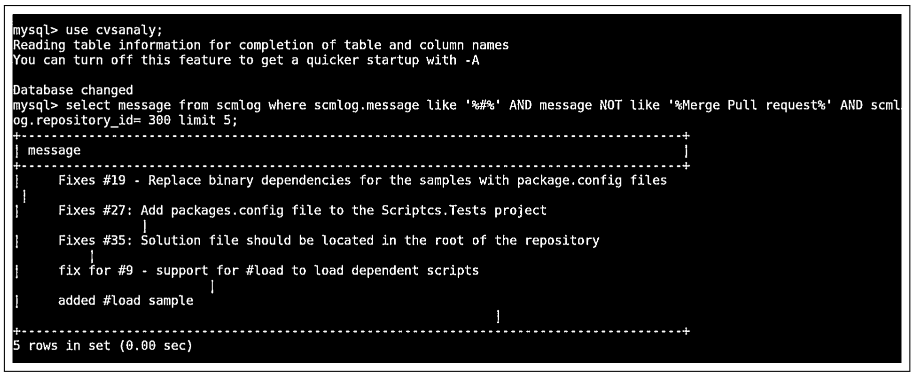
The process of syncing VC logs and BT data will improve the quality of bug data we use for validation of techniques and analysis in empirical software engineering. As a result, VC logs and BT data can be queried with higher precision, consistent data sets can be obtained (since the missing information in both tools is tracked and synchronised), and complete data sets of software projects can be obtained for posterior analysis.

6.2 Background

VC log data may be enhanced with the data from BT systems that report past software corrective maintenance activities. When these are exploited together, extensive tools that enable analyses and prediction of the future evolution of OSS projects are enriched effectively. Unfortunately, Bicho and CVSAnalY provide insufficient support for cross-analyses of both VC logs and BT data.

6.2.1 VC log

Version commit logs refer to actions of developers left in text format, which detail all the revisions and commits (changes) made to a software artefact. CVSAnalY retrieves VC logs and stores them to a database (MySQL) automatically. The VC logs information can be retrieved by issuing a query in the database, specifically in the **SCMlog** table that exists in the CVSAnalY database. The specification of additional parameters in the query (SQL query) allows the retrieval of information about a particular VC log. The SQL query in Figure 6.1 depicts a typical example of a query for a VC log file from one of the projects (the *Scripts* project[1]) we sampled in this research, detailing several contextual descriptions of the changes or revisions in the source code.



```
mysql> use cvsanaly;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select message from scmlog where scmlog.message like '%#%' AND message NOT like '%Merge Pull request%' AND scml
og.repository_id= 300 limit 5;
+-----------------------------------------------------------------------------------------+
| message                                                                                 |
+-----------------------------------------------------------------------------------------+
|     Fixes #19 - Replace binary dependencies for the samples with package.config files
 |
|     Fixes #27: Add packages.config file to the Scriptcs.Tests project
                 |
|     Fixes #35: Solution file should be located in the root of the repository
          |
|     fix for #9 - support for #load to load dependent scripts
                          |
|     added #load sample
                                                                 |
+-----------------------------------------------------------------------------------------+
5 rows in set (0.00 sec)
```

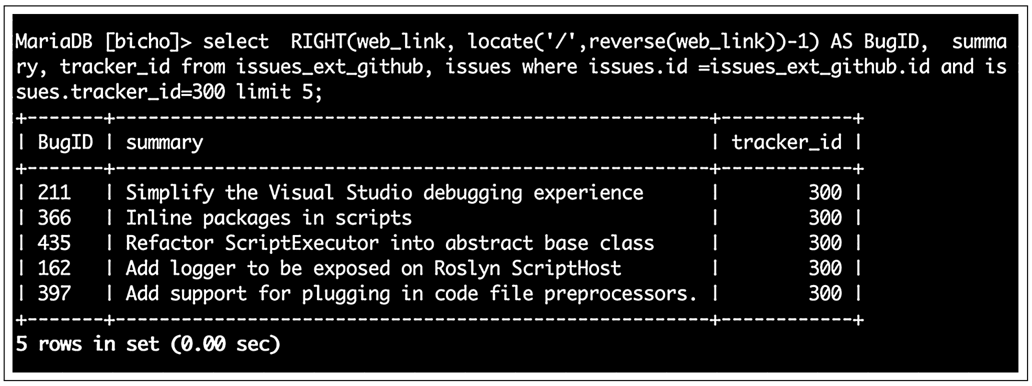
Figure 6.1: VC log

[1]https://github.com/scriptcs/scriptcs

6.2.2 BT data

BT data consists of two parts: a set of feature enhancement reports which describe metadata, such as the report and which component of the system it pertains to; and a textual description of the problems regarding the system.

In other words, BT data is considered as a bug report referring to a contextual description of a software problem or request for an additional feature enhancement to the software. For instance, the SQL query in Figure 6.2 depicts a typical example of an issue (BT system data) reported for the Scripts project[2]. The resulting table shows the first 5 bug IDs retrieved by the Bicho tool for the *Scripts* project.



```
MariaDB [bicho]> select  RIGHT(web_link, locate('/',reverse(web_link))-1) AS BugID,  summa
ry, tracker_id from issues_ext_github, issues where issues.id =issues_ext_github.id and is
sues.tracker_id=300 limit 5;
+-------+------------------------------------------------------+------------+
| BugID | summary                                              | tracker_id |
+-------+------------------------------------------------------+------------+
| 211   | Simplify the Visual Studio debugging experience      |        300 |
| 366   | Inline packages in scripts                           |        300 |
| 435   | Refactor ScriptExecutor into abstract base class     |        300 |
| 162   | Add logger to be exposed on Roslyn ScriptHost        |        300 |
| 397   | Add support for plugging in code file preprocessors. |        300 |
+-------+------------------------------------------------------+------------+
5 rows in set (0.00 sec)
```

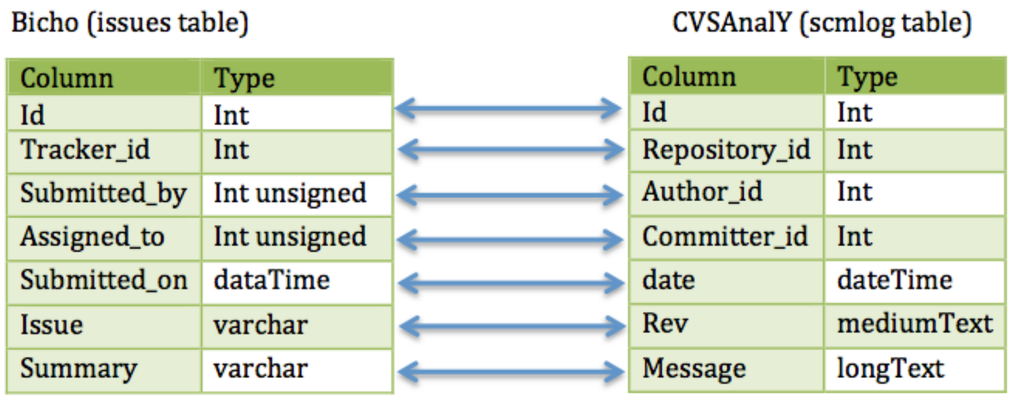
Figure 6.2: BT data

The BT system has a valuable and useful web interface mechanism that provides other systems, like Bicho, with access to BT data. In addition, the data that is retrieved from the BT system is generated from a template. All the BT data within the Bicho tool set have the same structure and format in the database.

Synchronising VC logs and BT data in a localised database provides the possibility of querying most of the interesting analysis problems that are missing in their respective databases. A simple database query statement ensures interoperability between Bicho and CVSAnalY tools. Thus, the evolution of OSS projects related to VC logs and BT data can be achieved collectively if they are mirrored in their respective databases.

[2]https://github.com/scriptcs/scriptcs

Observing the tables of Bicho and CVSAnalY and their attributes, we propose using bug-related data in either database to fill the missing data 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 VC log data with BT data (and vice versa) will require the use of metadata contained in the "**SCMlog**" table (populated by CVSAnalY) to be copied in the "**Issues**" table (populated by Bicho). Table 6.3 shows that attributes could be used from either table to fill the gaps in the other table.



Bicho (issues table) | CVSAnalY (scmlog table)

Column	Type		Column	Type
Id	Int	↔	Id	Int
Tracker_id	Int	↔	Repository_id	Int
Submitted_by	Int unsigned	↔	Author_id	Int
Assigned_to	Int unsigned	↔	Committer_id	Int
Submitted_on	dataTime	↔	date	dateTime
Issue	varchar	↔	Rev	mediumText
Summary	varchar	↔	Message	longText

Table 6.3: Corresponding fields linked in Bicho and CVSAnalY

6.3 Structure of The Framework

The structure of the framework comprises six modules: BT system, VC system, Bicho, CVS-AnalY, SCMlog and Issues. Figure 6.4 depicts the components in a UML notation that can be instantiated in the final implementation.

On the other hand, Figure 1.6 shows the architectural overview of the framework. The next subsections describe the main components, and what has been achieved to date.

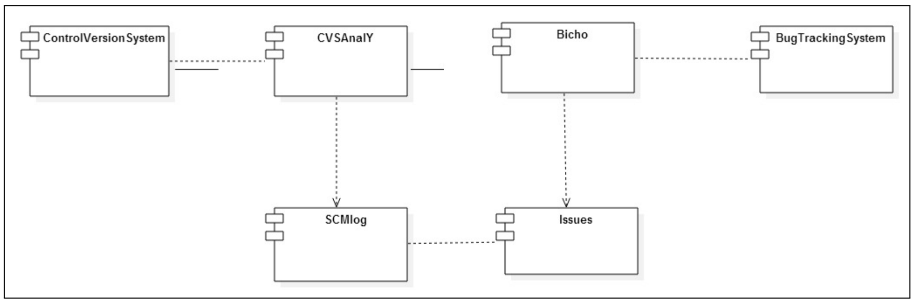


Figure 6.4: UML diagram of components

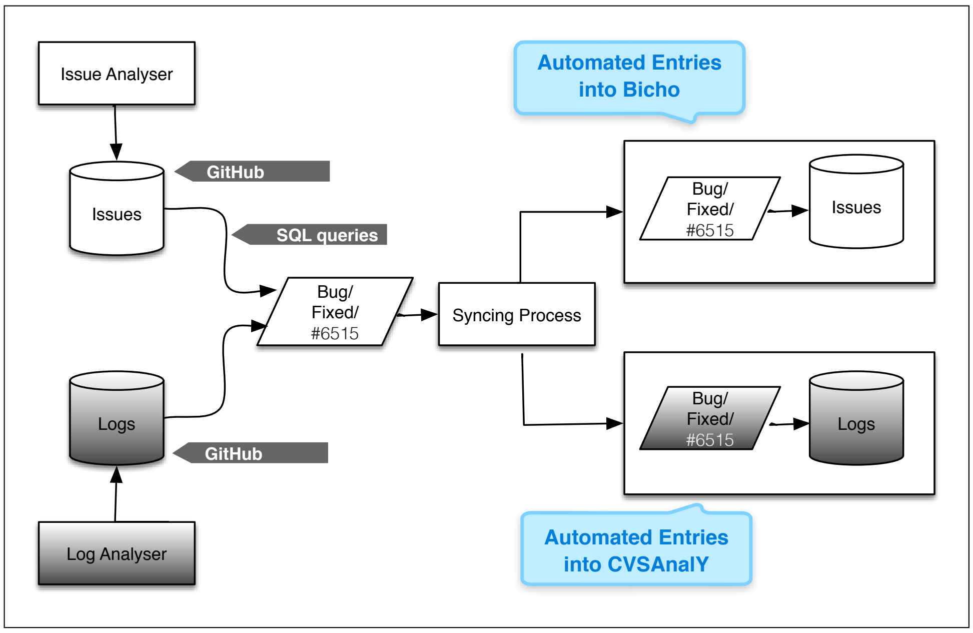


Figure 6.5: Architectural overview of the framework

6.3.1 VC Log Parser Via CVSAnalY

This component defines the interaction between CVSAnalY and any VC systems. Also, the SCMlog component serves as the main entry point where development logs are stored as extracted by CVSAnalY. In this way, the SCMlog interface must be implemented, to allow communication with any VC system. Currently, the framework supports the interaction with Git.

However, one of the main obstacles among the supported CVS is that Git requires authentication by the client or user before CVSAnalY can point to a repository in Git to extract and store VC logs. A username and password need to be entered, to allow communication between CVSAnalY and Git. As a result, this thesis implemented this framework only in its static interaction with Git: users need first to specify their logging credentials for authentication in GitHub in order to extract data by CVSAnalY from the remote VCS and stored development logs into a database generated by CVSAnalY. Refer to appendix A. Section A.1 for the complete codes.

6.3.2 BT data Parser Via Bicho

This component provides an interface in which the interaction between Bicho and any BT 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 will enable the interaction between Bicho and the supported bug-tracking systems. The framework currently supports 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. Refer to appendix A in Section A.1 for the complete codes.

6.4 Implementation

We implement the majority of the framework in Perl programming language, whose strengths are text manipulation, portability, fast development capabilities and a rapid development cycle

[28]. In addition, Perl has an impressively broad range of standard libraries. The Perl DBI package makes the automation and integration of databases very easy.

In this thesis, we make use of the SZZ algorithm [116] and track bugs and logs of the OSS projects sample from GitHub. In our formulation, we only looked for bugs described by the “#" sign and various numeric values (e.g., #1234) which are linked to the ID of a bug. In its original formulation, the SZZ algorithm also searches for keywords like “Bug", “Fixed" and others.

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 an intersection and union of sets, and synchronising the identified missing VC logs and BT data into their respective databases automatically.

6.4.1 Retrieving VC Logs

Obtaining the development logs: the tool is capable of interfacing with and executing CVS-AnalY and Bicho commands, in order to parse logs and bugs at once. CVSAnalY and Bicho automatically create databases and tables with metadata, storing all the development logs (VC logs) and BT data of the sample. Among the tables generated by CVSAnalY, we then specifically queried the *message* text field in **SCMlog** table, which mentions the number and unique IDs of changes in the VC system. In the presence of a bug ID, the VC logs also mention the bug ID with the #1234 format. For the purpose of this research, we were only interested in bug IDs that were being mentioned by developers: bug IDs did not necessarily need to be “fixed" or “resolved". This step was integrated into the tool that was developed for this research (Refer to appendix A in Section A.2 for the complete code)

6.4.2 Retrieving BT Data

Obtaining the BT data: the second phase in our data preparation process was to execute the Bicho tool to obtain and store all the information contained in the bug trackers of the projects as well as all the issues reported by the users of a project and confirmed as such by developers. One of the tables created by Bicho is 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. We queried specifically the **Issues_ext_bugzilla** table to obtain the set of unique numbers and IDs of bugs reported and confirmed by the developers table. This step was integrated into the tool that was developed for this research (Refer to appendix A in A.2 for the complete code).

6.4.3 Data Cleaning

Data cleaning: false positives and true positives – the cleaning step, before isolating the bug numbers and IDs for both CVSAnalY and Bicho. The query for the “#" sign followed by numeric values in the VC logs imported with CVSAnalY produced a large number of false positives in the pilot study carried out in Chapter 3.6. The messages refer to the pattern searched for by the “#" sign, but they are all linked to a request of pulling a merge from another distributed repository into the original one under GitHub. In this case, for the rest of the 344 OSS projects we obtained from GitHub, the same pattern was filtered out automatically using the SQL query integrated into the tool that was developed for this research. Line 2 (BT data) and lines 7–11 (VC logs) of the fragment of code in Code 6.1 indicate how we removed the trailing strings.

6.4.4 Isolating The Bug IDs

Isolating the bug numbers and IDs: after cleaning the data and removing all the trailing strings and white spaces, we composed two sets of bug IDs: one from the VC logs, and the other from the issue tracking systems. In the VC 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 assigned by the developers to the issues reported as bugs. These steps were performed within the developed tool, by querying the appropriate tables and cleansing the results. Line 2-4 (Bicho) and line 6-12 (CVSAnalY) of the fragment of code in Code 6.1 indicate how we composed the two sets of IDs of both tools.

Code 6.1: Cleaning VC logs and BT data

```
# Removing trailing white spaces.
$data[0] =~ s/\s+//;

push (@output_bicho, $data[0]);  # Isolating and assigning BT data

# Split VC logs into smaller section and removing trailing white spaces and strings
@tokens = split(/\s/, $data[0]);
    for($j=0; $j<=$#tokens; $j++){
        if ($tokens[$j] =~ /#\d+/){
            $tokens[$j] =~ s/(\.|\,|\;|\:)//;
            $tokens[$j] =~ s/.*#//;
            push (@output_cvs, $tokens[$j]);  # Isolating and assigning VC logs
        }
    }


```

6.4.5 Evaluation

Evaluating the union and intersection of the sets: the penultimate step was to evaluate the union and intersection of the sets, for each project. Given a set of bug IDs mentioned in the **SCMlog** table and the list of bug IDs stored in the issue trackers of a project, we evaluated the intersection (i.e., the common bug IDs) of these two sets (as visible at lines 2–3 in the fragment of code in Code 6.2), 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. This final step is integrated into the tool as depicted in the fragment in Code 6.2 (Evaluation of VC logs and BT data).

We randomly picked a few of the sample of 344 OSS projects (Discussed in Sub-section

4.3.2 and Section 4.4 of Chapter 4) obtained from GitHub and manually analysed each of the remaining bugs in the 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 were always related to bug IDs.

Code 6.2: Evaluation of VC logs and BT data

```
# CREATE SETS, USE SETS
$s1 = Set::Scalar->new (@output_bicho);
$s2 = Set::Scalar->new (@output_cvs);

```

6.5 Quantification of VC logs and BT data

After evaluating the sets of VC logs and BT data using set theory, we obtained the common, union and intersection of sets of all 344 OSS projects we sampled in this research. We utilised the shared bug coverage metric that we used in evaluating the sets of IDs, such that all the VC log IDs found only in CVSAnalY, and not in Bicho, could be inserted into the **Issues** table of Bicho. The IDs would be used as references in the **SCMlog** table and a query would be parsed to retrieve relevant metadata inside the table (**SCMlog**). For instance, the relevant metadata would include all the identified links which we mapped in Figure 6.3. Codes 6.3 and 6.4 depict a typical example of the SQL queries we used in both tools, which quantified all the VC logs and BT data. However, we applied the full SZZ algorithms and quantified BT data into VC logs present in Bicho and CVSAnalY, for example the use of the “#" symbol, “Fixed" and “Bug" keywords in the CVSAnalY database (we used the same query and obtained the results presented in Section A.6 at Appendix A).

Code 6.3: SQL query to retrieve BT Data

```
select RIGHT(web_link, locate('/',reverse(web_link))-1) from issues_ext_github, issues
    where issues.id = issues_ext_github.id and issues.tracker_id = 1;
```

Code 6.4: SQL query to retrieve VC log

```
select message from scmlog where scmlog.message like '%#%' AND message NOT like '%
    Merge␣Pull␣request%' AND scmlog.repository_id= 1;

select message from scmlog where scmlog.message like '%Fixed%' AND message NOT like '
    %Merge␣Pull␣request%' AND scmlog.repository_id= 1;

select message from scmlog where scmlog.message like '%Bug%' AND message NOT like '%
    Merge␣Pull␣request%' AND scmlog.repository_id= 1;
```

Code 6.5 shows how we quantified VC logs and BT data for each project using the set operations, as follows:

$$SharedBugCoverage = \frac{Intersection}{Union} \tag{6.1}$$

- **Only in Bicho**: In this set of operations we obtained all the unique BT data IDs in the Bicho database that are not traced in the CVSAnalY database (the operation of the set is visible at line 8 in the fragment of code in Code 6.5).
- **All in CVSAnalY**: In this operation we obtained all the VC log IDs in the CVSAnalY **SCMlog** table which might be traced in Bicho (the operation of the set is visible at line 9 in Code 6.5, which is a fragment of the tool chain developed for this thesis).
- **Only in CVSAnalY**: In this set of operations we obtained the unique VC log IDs in the CVSAnalY database that are not traced in the Bicho database (the operation of the set is visible at line 10 in Code 6.5).
- **All in Bicho**: In this set of operations we obtained all the BT data IDs in the Bicho **Issues** tables which might be traced in the CVSAnalY database (the operation of the set is visible at line 11 of Code 6.5, which is a fragment of the tool chain developed in this thesis).
- **Common BT system and VC logs in Both tools (intersection)**: In this operation we obtained all the BT system and VC log IDs present in both the Bicho and CVSAnalY

databases – in other words, the IDs the tool chain was able to trace in both tools (the operation of the sets is visible at line 12 in Code 6.5).

- **The union**: This operation obtained the total number of VC logs and BT data retrieved by the tool chain in Bicho and CVSAnalY respective databases. Also, the operation of the union is visible at line 13 in Code 6.5.

After the sizing of VC logs and BT data in both tools, we display the results for each of the 344 OSS projects sampled in this research. Line 16 of Code 6.5 prints the number of VC logs and BT data IDs based on the set of operations we itemised between lines 8 and 13 of Code 6.5, which is a fragment of the tool chain we developed in this research. (Refer to Section A.3 in Appendix A for the codes and Section A.6 in Appendix A for the results.)

Code 6.5: Quantification of VC logs and BT data

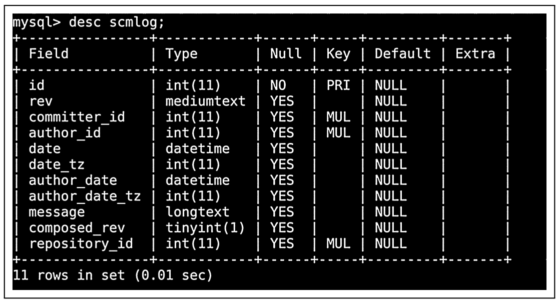
```
 1
 2  # CREATE SETS, USE SETS
 3  $s1 = Set::Scalar−>new (@output_bicho);
 4  $s2 = Set::Scalar−>new (@output_cvs);
 5
 6  # OPERATIONS ON SETS
 7
 8  $only_in_bicho  = $s1 − $s2;     # only in bicho
 9  $in_cvs  = $s2;             #  in cvsanaly
10  $only_in_cvs  = $s2−$s1;         # only in cvsanaly
11  $in_bicho = $s1;          # in bicho
12  $common  = $s1 ∗ $s2;          # common
13  $total  = $s1 + $s2;            # union
14
15
```

6.6 Re-engineering Bicho and CVSAnalY

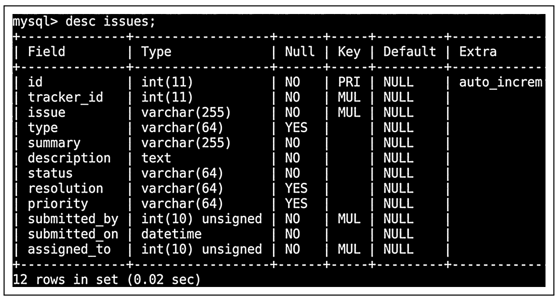
The first part of the process to change and improve the design of Bicho and CVSAnalY is to identify the relevant entities that hold BT data and VC logs. In this way, the entities will be examined to ensure the changes we made and the design conforms to the entities and referential integrity of both databases. The referential integrity is where the foreign key contains a value that refers to the existing valid row in another relation [102]. In this case, we observed that the identified entities – that is to say, the **SCMlog** and Issues tables – might need to be altered to allow cross-linking of VC logs and BT data correctly, since the data type that exists in each data field of the tables in both tools varies slightly. The *rev* column in the **SCMlog** table data type is *medium text* while in the *issue* column in the **Issues** table the data type is *VarChar*. Such a typical discrepancy could result in missing data or return an invalid entry during the automated entry in both tools.

In addition, the data structure of the **Issues** table in Bicho – for instance, the ID field (i.e. the primary key in the **Issues** table) is set to be *auto_increment*. However, the primary key – that is, the ID field – in the **SCMlog** table is not set to *auto_increment*. Therefore, we also expected some difficulty in automating the entries of BT data into the **SCMlog** table, like duplicate entries for the primary key, since it is not set to *auto_increment*. The tables are depicted in table 6.9.

Other integrity rules that are enforced in the Bicho and CVSAnalY databases are the NOT NULL and UNIQUE constraints. The NOT NULL constraints are placed in the **SCMlog** table of CVSAnalY in each column except the *id* column (i.e., the primary key) to ensure that every row in the table has a value for that column during the insertion of BT data traced in Bicho, but not in CVSAnalY. Conversely, the UNIQUE constraint restriction is also placed in each column to ensure that no duplicate values exist in the **SCMlog** and **Issues** tables of both tools (Bicho and CVSAnalY). In addition, we will demonstrate some typical examples where such constraints were encountered during the synchronisation process. Table 6.6 below depict the SCMlog table in CVSAnalY and the Issues table in Bicho respectively.



```
mysql> desc scmlog;
+----------------+------------+------+-----+---------+-------+
| Field          | Type       | Null | Key | Default | Extra |
+----------------+------------+------+-----+---------+-------+
| id             | int(11)    | NO   | PRI | NULL    |       |
| rev            | mediumtext | YES  |     | NULL    |       |
| committer_id   | int(11)    | YES  | MUL | NULL    |       |
| author_id      | int(11)    | YES  | MUL | NULL    |       |
| date           | datetime   | YES  |     | NULL    |       |
| date_tz        | int(11)    | YES  |     | NULL    |       |
| author_date    | datetime   | YES  |     | NULL    |       |
| author_date_tz | int(11)    | YES  |     | NULL    |       |
| message        | longtext   | YES  |     | NULL    |       |
| composed_rev   | tinyint(1) | YES  |     | NULL    |       |
| repository_id  | int(11)    | YES  | MUL | NULL    |       |
+----------------+------------+------+-----+---------+-------+
11 rows in set (0.01 sec)
```



```
mysql> desc issues;
+--------------+------------------+------+-----+---------+-----------
| Field        | Type             | Null | Key | Default | Extra
+--------------+------------------+------+-----+---------+-----------
| id           | int(11)          | NO   | PRI | NULL    | auto_increm
| tracker_id   | int(11)          | NO   | MUL | NULL    |
| issue        | varchar(255)     | NO   | MUL | NULL    |
| type         | varchar(64)      | YES  |     | NULL    |
| summary      | varchar(255)     | NO   |     | NULL    |
| description  | text             | NO   |     | NULL    |
| status       | varchar(64)      | NO   |     | NULL    |
| resolution   | varchar(64)      | YES  |     | NULL    |
| priority     | varchar(64)      | YES  |     | NULL    |
| submitted_by | int(10) unsigned | NO   | MUL | NULL    |
| submitted_on | datetime         | NO   |     | NULL    |
| assigned_to  | int(10) unsigned | NO   | MUL | NULL    |
+--------------+------------------+------+-----+---------+-----------
12 rows in set (0.02 sec)
```

Table 6.6: SCMlog table in CVSAnalY database

6.7 Synchronisation

Merging BT data and VC logs from different sources is a big challenge that requires complex methods [106], since we are correlating and merging information from various sources.

In this research, we utilised the existing techniques and attempts to provide a resolution to this dilemma (i.e., merging BT data and VC logs from various sources using Bicho and CVSAnalY). Thus, we will merge BT data into VC logs of OSS projects using a simplified approach.

In this way, we will instantiate the main components we mentioned in Section 6.3 above – that is to say, the *Issues* interface and the *SCMlog* interface – in the structure of the framework to enable an interaction or connection between Bicho and the supported BT system as well as CVSAnalY and the supported distributed VC system, such as GitHub.

6.7.1 T1: Bicho – CVSAnalY

In this section, we synchronised the BT data and VC logs of two entities – that is to say, the **SCMlog** table and the **Issues** table of CVSAnalY and Bicho respectively. We streamlined the synchronisation in two forms: in Test 1 (T1) we attempted to sync missing BT data in the **Issues** table of Bicho into the **SCMlog** table of CVSAnalY, and we attempted to sync missing VC logs in the **SCMlog** table of CVSAnalY into the **Issues** table of Bicho.

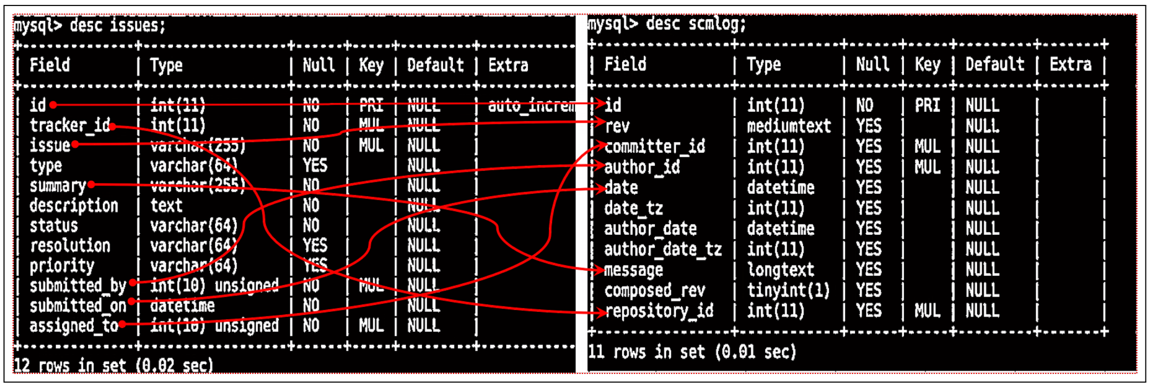
We began by instantiating and implementing an interface (i.e., issues table) that enabled the interaction with BT data retrieved by Bicho from the supported BT system. However, a connection needed to be established with CVSAnalY to allow an asynchronous exchange of data between Bicho and CVSAnalY.

In this way, we used a set operation function in which we implemented and obtained the BT data existing **only in Bicho** and already retrieved from the supported BT system and merged it with VC logs in the CVSAnalY *SCMlog* interface (table). BT data was merged by composing two sets of BT data IDs and VC log IDs in Bicho and CVSAnalY. The evaluation of the sets of IDs was carried out automatically and integrated in the tool chain developed for this thesis. The missing IDs were isolated and mapped as a set of elements (using a scaler set in Perl) separately (i.e. **only in Bicho** and only in CVSAnalY) using the set operation function (as visible in Code 6.5 in Section 6.5 of this chapter).

BT data in the *Issues* interface (table) of Bicho that was not mirrored in the *SCMlog* interface (table) of CVSAnalY could be synchronised using the set operation function. We called the set operation function and initiated a loop ***($only_in_bicho−>element)*** to construct the full list of elements (i.e. BT data IDs) existing only in Bicho and checked through each BT system. Thus, we query the **Issues** table serving as the interface in Bicho, select the identified columns and insert them into the **SCMlog** table serving as the interface in the CVSAnalY database to merge BT data not mirrored in CVSAnalY using the full list of elements (i.e., BT data IDs existing only in Bicho and obtained in the set operation (***$only_in_bicho***)). The arrows in table 6.7 graphically shows the attributes and the metadata contained in the “**Issues**" table (populated by Bicho) to be copied in the “**SCMlog**" table (populated by CVSAnalY).

6.7.2 T1: CVSAnalY – Bicho

In this section, we instantiated and implemented an interface (i.e., **SCMlog** table) that enabled the interaction with the VC logs retrieved by CVSAnalY from the supported CVS. Conversely, we established a connection with CVSAnalY to initiate and allow an asynchronous exchange of data between CVSAnalY and Bicho. As a result, we could also use the set operation function in which we implemented and obtained the VC logs existing only in CVS and already retrieved from the supported VC system and merged them with BT data in the



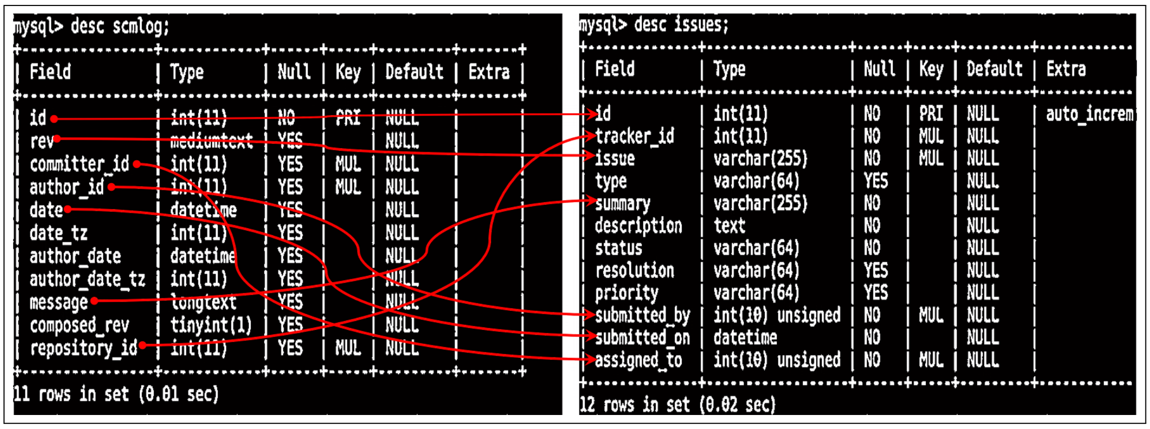
```
mysql> desc issues;
+--------------+------------------+------+-----+---------+------------
| Field        | Type             | Null | Key | Default | Extra
+--------------+------------------+------+-----+---------+------------
| id           | int(11)          | NO   | PRI | NULL    | auto_increm
| tracker_id   | int(11)          | NO   | MUL | NULL    |
| issue        | varchar(255)     | NO   | MUL | NULL    |
| type         | varchar(64)      | YES  |     | NULL    |
| summary      | varchar(255)     | NO   |     | NULL    |
| description  | text             | NO   |     | NULL    |
| status       | varchar(64)      | NO   |     | NULL    |
| resolution   | varchar(64)      | YES  |     | NULL    |
| priority     | varchar(64)      | YES  |     | NULL    |
| submitted_by | int(10) unsigned | NO   | MUL | NULL    |
| submitted_on | datetime         | NO   |     | NULL    |
| assigned_to  | int(10) unsigned | NO   | MUL | NULL    |
+--------------+------------------+------+-----+---------+------------
12 rows in set (0.02 sec)
```

```
mysql> desc scmlog;
+----------------+------------+------+-----+---------+-------+
| Field          | Type       | Null | Key | Default | Extra |
+----------------+------------+------+-----+---------+-------+
| id             | int(11)    | NO   | PRI | NULL    |       |
| rev            | mediumtext | YES  |     | NULL    |       |
| committer_id   | int(11)    | YES  | MUL | NULL    |       |
| author_id      | int(11)    | YES  | MUL | NULL    |       |
| date           | datetime   | YES  |     | NULL    |       |
| date_tz        | int(11)    | YES  |     | NULL    |       |
| author_date    | datetime   | YES  |     | NULL    |       |
| author_date_tz | int(11)    | YES  |     | NULL    |       |
| message        | longtext   | YES  |     | NULL    |       |
| composed_rev   | tinyint(1) | YES  |     | NULL    |       |
| repository_id  | int(11)    | YES  | MUL | NULL    |       |
+----------------+------------+------+-----+---------+-------+
11 rows in set (0.01 sec)
```

Table 6.7: Issues —> SCMLog

Bicho *Issues* interface (table). VC logs were merged by composing two sets of BT data IDs and VC log IDs in Bicho and CVSAnalY. The evaluation of the sets of IDs was carried out automatically and integrated in the tool chain developed for this thesis. The missing IDs were isolated and mapped as a set of elements (using a scaler set in Perl) separately (i.e. **only in CVSAnalY** and **only in Bicho**) using the set operation function (as visible in Code 6.5 in Section 6.5 of this chapter).

VC logs in the *SCMlog* interface (table) of CVSAnalY not mirrored in the *Issues* interface (table) of Bicho could be synchronised using our set operation function. we called the set operation function and initiated a loop ($only_in_cvs–>element) to construct the full list of elements (i.e. VC log IDs) appearing only in the **SCMlog** table of CVSAnalY database, specifically in the ***message*** column (using the “#" symbol), and then checked each VC log ID and processed them one element at a time. Thus, we query the **SCMlog** table in the ***message*** column serving as the interface in Bicho, select the identified columns and insert them into the **Issues** table serving as the interface in the Bicho database to merge VC logs not mirrored in Bicho using the full list of elements(i.e., VC log IDs existing only in CVSAnalY and obtained in the set operation ($only_in_cvs)). Also, the arrows in table 6.8 graphically shows the attributes and the metadata contained in the “**SCMlog**" table (populated by CVSAnalY) to be copied in the “**Issues**" table (populated by Bicho).



```
mysql> desc scmlog;
+---------------+------------+------+-----+---------+-------+
| Field         | Type       | Null | Key | Default | Extra |
+---------------+------------+------+-----+---------+-------+
| id            | int(11)    | NO   | PRI | NULL    |       |
| rev           | mediumtext | YES  |     | NULL    |       |
| committer_id  | int(11)    | YES  | MUL | NULL    |       |
| author_id     | int(11)    | YES  | MUL | NULL    |       |
| date          | datetime   | YES  |     | NULL    |       |
| date_tz       | int(11)    | YES  |     | NULL    |       |
| author_date   | datetime   | YES  |     | NULL    |       |
| author_date_tz| int(11)    | YES  |     | NULL    |       |
| message       | longtext   | YES  |     | NULL    |       |
| composed_rev  | tinyint(1) | YES  |     | NULL    |       |
| repository_id | int(11)    | YES  | MUL | NULL    |       |
+---------------+------------+------+-----+---------+-------+
11 rows in set (0.01 sec)
```

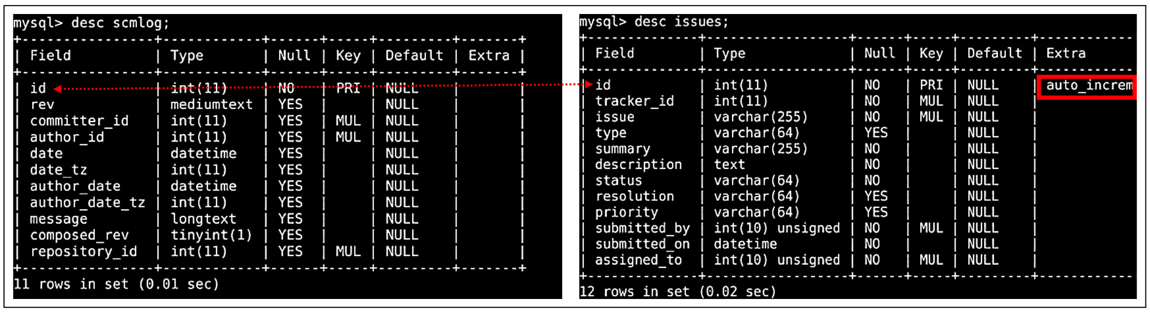
```
mysql> desc issues;
+--------------+------------------+------+-----+---------+-----------
| Field        | Type             | Null | Key | Default | Extra
+--------------+------------------+------+-----+---------+-----------
| id           | int(11)          | NO   | PRI | NULL    | auto_increm
| tracker_id   | int(11)          | NO   | MUL | NULL    |
| issue        | varchar(255)     | NO   | MUL | NULL    |
| type         | varchar(64)      | YES  |     | NULL    |
| summary      | varchar(255)     | NO   |     | NULL    |
| description  | text             | NO   |     | NULL    |
| status       | varchar(64)      | NO   |     | NULL    |
| resolution   | varchar(64)      | YES  |     | NULL    |
| priority     | varchar(64)      | YES  |     | NULL    |
| submitted_by | int(10) unsigned | NO   | MUL | NULL    |
| submitted_on | datetime         | NO   |     | NULL    |
| assigned_to  | int(10) unsigned | NO   | MUL | NULL    |
+--------------+------------------+------+-----+---------+-----------
12 rows in set (0.02 sec)
```

Table 6.8: SCMlog —> Issues

6.7.3 Issues with synchronisation

The synchronisation was implemented, as we envisage the occurrence of duplicate entries of bug IDs to be synchronised into Bicho database. As we mentioned in Section 6.6, there might be duplicate entries for primary keys in the ID column of the **Issues** and **SCMlog** Table in either databases as depicted in Figure 6.9. Figure 6.10 shows the failed synchronisation of VC logs from CVSAnalY to Bicho in T1 during the implementation.

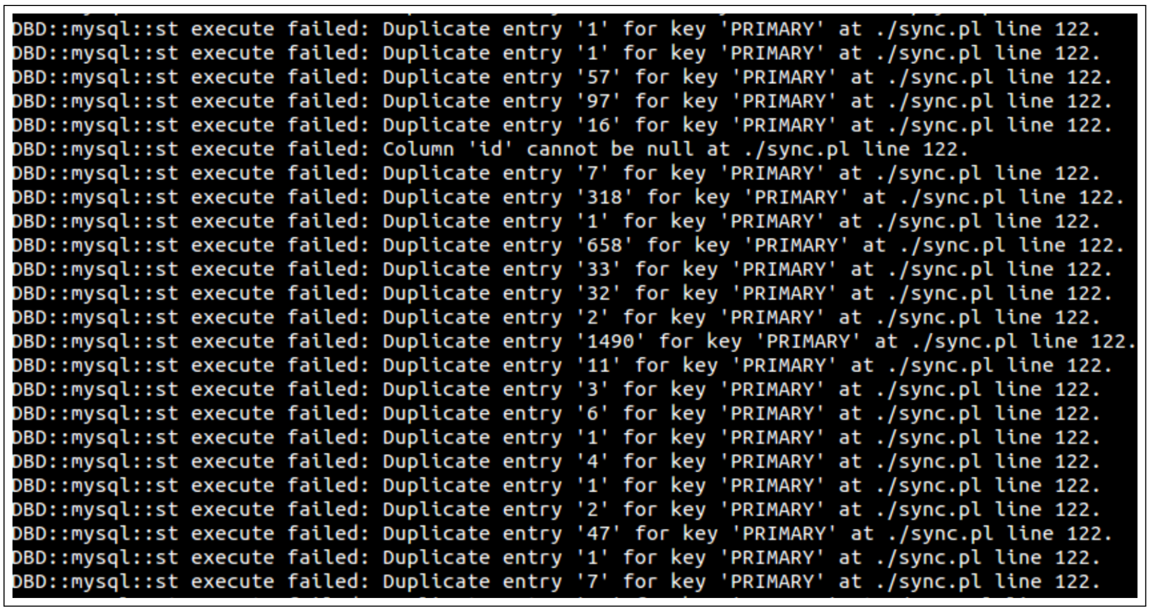
Therefore, we preferred not to intervene in the existing databases (Bicho and CVSAnalY) and Tables (**Issues** and **SCMlog**) at this stage, but rather to work on two different tables, to be integrated within CVSAnalY and Bicho, respectively.



```
mysql> desc scmlog;
+---------------+------------+------+-----+---------+-------+
| Field         | Type       | Null | Key | Default | Extra |
+---------------+------------+------+-----+---------+-------+
| id            | int(11)    | NO   | PRI | NULL    |       |
| rev           | mediumtext | YES  |     | NULL    |       |
| committer_id  | int(11)    | YES  | MUL | NULL    |       |
| author_id     | int(11)    | YES  | MUL | NULL    |       |
| date          | datetime   | YES  |     | NULL    |       |
| date_tz       | int(11)    | YES  |     | NULL    |       |
| author_date   | datetime   | YES  |     | NULL    |       |
| author_date_tz| int(11)    | YES  |     | NULL    |       |
| message       | longtext   | YES  |     | NULL    |       |
| composed_rev  | tinyint(1) | YES  |     | NULL    |       |
| repository_id | int(11)    | YES  | MUL | NULL    |       |
+---------------+------------+------+-----+---------+-------+
11 rows in set (0.01 sec)
```

```
mysql> desc issues;
+--------------+------------------+------+-----+---------+-----------
| Field        | Type             | Null | Key | Default | Extra
+--------------+------------------+------+-----+---------+-----------
| id           | int(11)          | NO   | PRI | NULL    | auto_increm
| tracker_id   | int(11)          | NO   | MUL | NULL    |
| issue        | varchar(255)     | NO   | MUL | NULL    |
| type         | varchar(64)      | YES  |     | NULL    |
| summary      | varchar(255)     | NO   |     | NULL    |
| description  | text             | NO   |     | NULL    |
| status       | varchar(64)      | NO   |     | NULL    |
| resolution   | varchar(64)      | YES  |     | NULL    |
| priority     | varchar(64)      | YES  |     | NULL    |
| submitted_by | int(10) unsigned | NO   | MUL | NULL    |
| submitted_on | datetime         | NO   |     | NULL    |
| assigned_to  | int(10) unsigned | NO   | MUL | NULL    |
+--------------+------------------+------+-----+---------+-----------
12 rows in set (0.02 sec)
```

Figure 6.9: T1 Demo: Primary *ID* field in Issues Table (Bicho) is set to be Auto increment

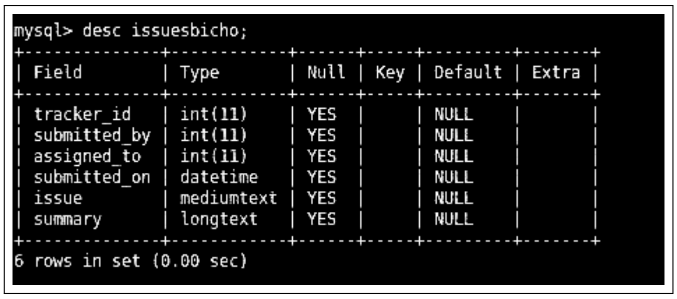


```
DBD::mysql::st execute failed: Duplicate entry '1' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '1' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '57' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '97' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '16' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Column 'id' cannot be null at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '7' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '318' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '1' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '658' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '33' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '32' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '2' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '1490' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '11' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '3' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '6' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '1' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '4' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '1' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '2' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '47' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '1' for key 'PRIMARY' at ./sync.pl line 122.
DBD::mysql::st execute failed: Duplicate entry '7' for key 'PRIMARY' at ./sync.pl line 122.
```

Figure 6.10: T1 and T2 Demo

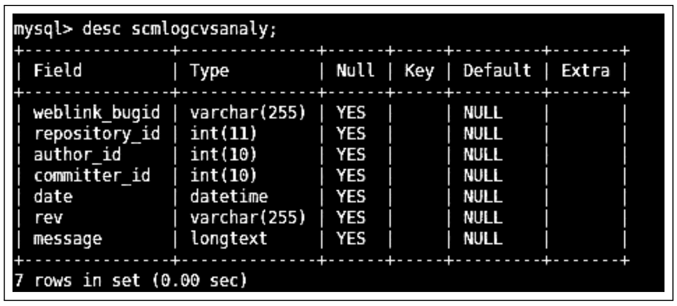
6.8 Re-aligning CVSAnalY and Bicho

We realigned the Bicho and CVSAnalY databases by adding two extra Tables in their respective databases – namely the **SCMlogCVSAnalY** Table in CVSAnalY and the **IssuesBicho** Table in Bicho – as depicted in Table 6.11 and 6.12.



```
mysql> desc issuesbicho;
+--------------+------------+------+-----+---------+-------+
| Field        | Type       | Null | Key | Default | Extra |
+--------------+------------+------+-----+---------+-------+
| tracker_id   | int(11)    | YES  |     | NULL    |       |
| submitted_by | int(11)    | YES  |     | NULL    |       |
| assigned_to  | int(11)    | YES  |     | NULL    |       |
| submitted_on | datetime   | YES  |     | NULL    |       |
| issue        | mediumtext | YES  |     | NULL    |       |
| summary      | longtext   | YES  |     | NULL    |       |
+--------------+------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Table 6.11: IssuesBicho table



```
mysql> desc scmlogcvsanaly;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| weblink_bugid | varchar(255) | YES  |     | NULL    |       |
| repository_id | int(11)      | YES  |     | NULL    |       |
| author_id     | int(10)      | YES  |     | NULL    |       |
| committer_id  | int(10)      | YES  |     | NULL    |       |
| date          | datetime     | YES  |     | NULL    |       |
| rev           | varchar(255) | YES  |     | NULL    |       |
| message       | longtext     | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Table 6.12: SCMlogCVSAnalY table

6.8.1 T2 Bicho – IssuesBicho table

Similarly, we realigned the Bicho databases by adding an extra table called **IssuesBicho** and inserting VC logs not mirrored in the Bicho database. This is automated and integrated into the tool chain developed, as visible in the fragment of code in Appendix A Section A.4 between lines 28-39.

6.8.2 T2: CVSAnalY – SCMlogCVSAnalY table

In this Table, we synced BT data not mirrored in CVSAnalY by creating the **SCMlogCVS-AnalY** table if it did not exist (integrated in the tool chain) and inserting the missing BT data into the **SCMlogCVSAnalY** Table created automatically, as visible in the fragment of code in Appendix A Section A.4 between lines 72-86.

6.8.3 Implementation of Auxiliary Tables

The synchronisation was implemented successfully. The BT data not mirrored in the **SCMlog** Table in the CVSAnalY database was synchronised into the **SCMlogCVSAnalY** table of CVSAnalY automatically, and without duplication. Similarly, VC logs not mirrored in the **Issues** table in the Bicho database was synchronised into **IssuesBicho** automatically. Figure 6.16 shows the automatic syncing process in both tools for all 344 OSS projects we sampled in this research.[3] for the complete code.

[3]Refer to Appendix A in A.4

The table in 6.1 present the percentage of BT data and VC logs recovered and synchronised in the auxiliary tables of Bicho and CVSAnalY databases per project. The columns in Table 6.1 such as **Only in Bicho** and **Only in CVSAnalY** for **Project ID=42** in Bicho and CVSAnalY database shows the number of BT data and VC logs only tracked in **Project ID=42**. Thus, the 52 BT data found **only in Bicho** (i.e., 20.47% of the BT data in project ID=42) was recovered and synchronised into CVSAnalY auxiliary (SCMlogCVSAnalY) Table automatically. On the other hand, a significant bug IDs was found in the VC log. In this case, the 202 number of bug Ids tracked in VC logs (i.e., 79.53% of bug IDs) in column **only in CVSAnaly** for project ID=42 in the table 6.1 was recovered and synchronised into Bicho auxiliary (Issuesbicho) table automatically. This was evaluated using the tool-chain (Refer to appendix A.5 for the complete code) automatically for all the 344 OSS projects sampled in this research. The percentage of bug data recovered and synchronised for the rest of the 300 OSS projects can also be found in Appendix A Section A.8 (Mysql dump of Bicho Delta and CVSAnalY Delta (database) which holds the recovered and synchronised BT data and VC logs of 344 OSS projects can be found on Figshare.[4]

The box plot in Figure 6.13, shows the set of bug IDs only found in CVSAnalY is in general very low: in around 75% of projects bug IDs not found in CVSAnalY (i.e., only in Bicho) is synchronised in the CVSAnalY database. This means Bicho delta contains less information on bug IDs in Issuesbicho (i.e., the table in Bicho database that holds the VC logs synch from CVSAnalY).

In addition, the box plot in Figure 6.13 shows that some of the OSS projects we sampled on GitHub for this research, there is significant effect on missing data with respect to VC logs. Thus,the outliers in the box-plot indicate in around 25% of all the 344 OSS projects in CVSAnalY database the only bug Ids presence was synchronised into bicho delta (i.e., issuesbicho table in Bicho database).

On the other hand, the box plot in Figure 6.14, shows the set of bug IDs only found in Bicho is in general very high: in around 75% of projects bug IDs was found in Bicho was synchronised into newer SCMlogcvsanaly table of CVSAnalY database created in the database. This mean

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

Project Ids	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Bicho Delta	CVSAnalY Delta
1	57	6	6	51	0	57	0.00%	89.47%
2	449	19	19	430	0	449	0.00%	95.77%
3	790	30	30	760	0	790	0.00%	96.20%
4	213	15	14	199	1	214	0.47%	92.99%
5	6	0	0	6	0	6	0.00%	100.00%
6	101	21	20	81	1	102	0.98%	79.41%
7	18	0	0	18	0	18	0.00%	100.00%
8	1459	218	202	1257	16	1475	1.08%	85.22%
9	34	2	2	32	0	34	0.00%	94.12%
10	2	1	0	2	1	3	33.33%	66.67%
11	18	3	2	16	1	19	5.26%	84.21%
12	29	1	1	28	0	29	0.00%	96.55%
13	14	541	1	13	540	554	97.47%	2.35%
14	2257	554	544	1713	10	2267	0.44%	75.56%
15	195	40	19	176	3	198	1.52%	79.80%
16	494	56	54	440	2	496	0.40%	88.71%
17	0	13	0	0	13	13	100.00%	0.00%
18	1	0	0	1	0	1	0.00%	100.00%
19	792	31	5	787	26	818	3.18%	96.21%
20	33	1	1	32	0	33	0.00%	96.97%
21	321	6	6	315	0	321	0.00%	98.13%
22	40	1	1	39	0	40	0.00%	97.50%
23	1	0	0	1	0	1	0.00%	100.00%
24	6	0	0	6	0	6	0.00%	100.00%
25	11	3	0	11	3	14	21.43%	78.57%
26	166	107	3	163	104	270	38.52%	60.37%
27	121	17	17	104	0	121	0.00%	85.95%
28	325	68	68	257	0	325	0.00%	79.08%
29	3	0	0	3	0	3	0.00%	100.00%
30	7	6	0	7	6	13	46.15%	53.85%
31	232	192	185	47	7	239	2.93%	19.67%
32	8	0	0	8	0	8	0.00%	100.00%
33	21	0	0	21	0	21	0.00%	100.00%
34	139	2	0	139	2	141	1.42%	98.58%
35	364	28	28	336	0	364	0.00%	92.31%
36	27	1	1	26	0	27	0.00%	96.30%
37	47	6	6	41	0	47	0.00%	87.23%
38	0	0	0	0	0	0	0.00%	0.00%
39	945	122	105	840	17	962	1.77%	87.32%
40	27	2	2	25	0	27	0.00%	92.59%
41	236	35	4	232	31	267	11.61%	86.89%
42	52	202	0	52	202	254	79.53%	20.47%
43	421	94	93	328	1	422	0.24%	77.73%
44	24	3	3	21	0	24	0.00%	87.50%
45	0	0	0	0	0	0	0.00%	0.00%
46	107	1	1	106	0	107	0.00%	99.07%
47	26	37	0	26	37	63	58.73%	41.27%
48	68	5	5	63	0	68	0.00%	92.65%
49	55	9	8	47	1	56	1.79%	83.93%
50	51	0	0	51	0	51	0.00%	100.00%

Table 6.1: Synchronisation of BT data and VC log using # Symbol of the SZZ Algorithm - 344 OSS Projects

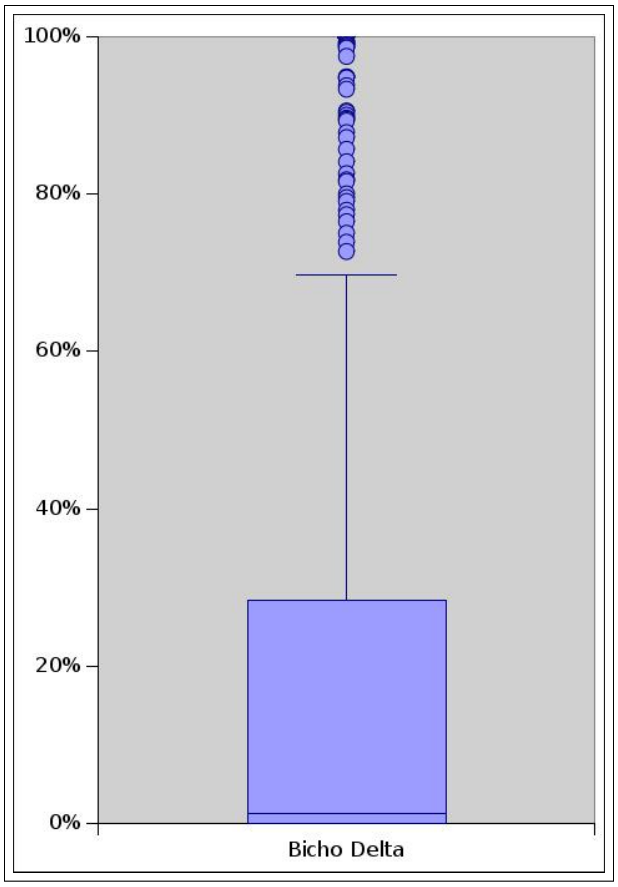


Figure 6.13: Bicho delta

CVSAnalY delta will contains more information on bug IDs that were circumvented in the SCMlogcvsanaly table of CVSAnalY database found only in bicho. Thus, there are no outliers in the upper quartile of the box plot. In most of the 344 OSS projects we sampled in this research, the majority of missing BT data and VC log were circumvented automatically using our approach in both Bicho and CVSAnalY respective databases.

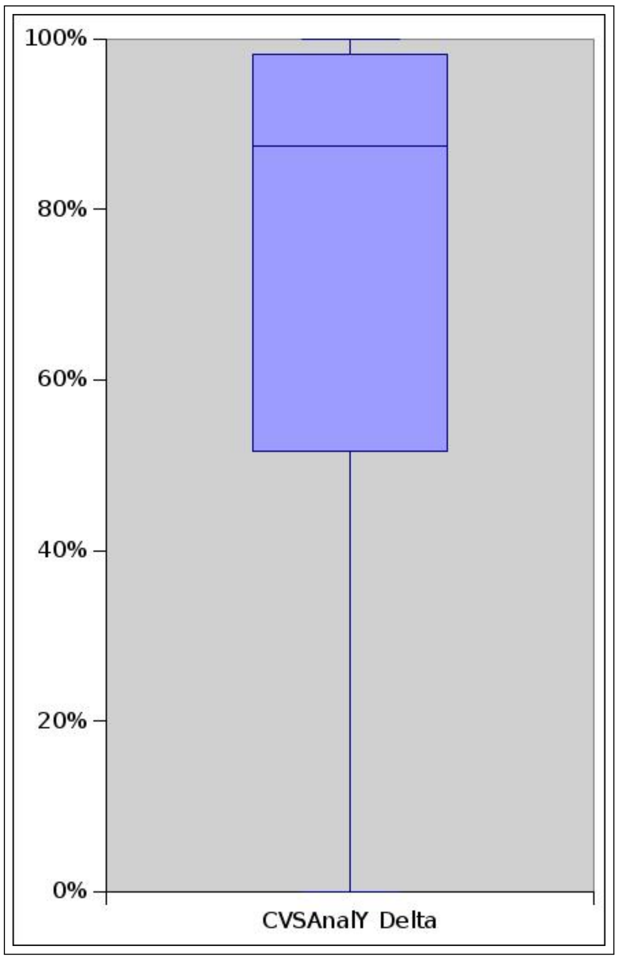


Figure 6.14: CVSAnalY delta

Furthermore, the bar chart displayed in Figure 6.15 summarise graphically the percentage of BT data and VC log been recovered and synchronised automatically in Bicho and CVSAnalY respective databases using our approach for all the 344 OSS projects we sampled in this research.

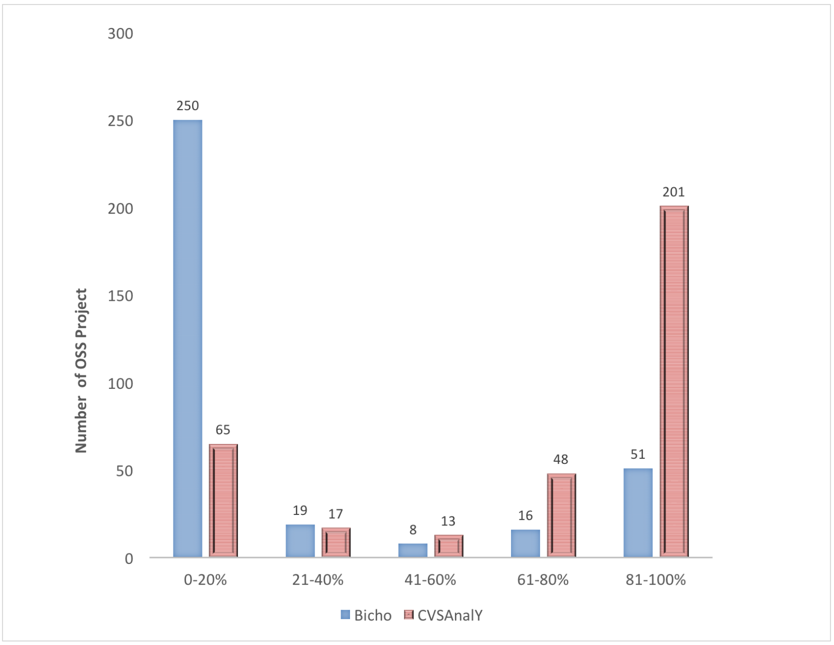
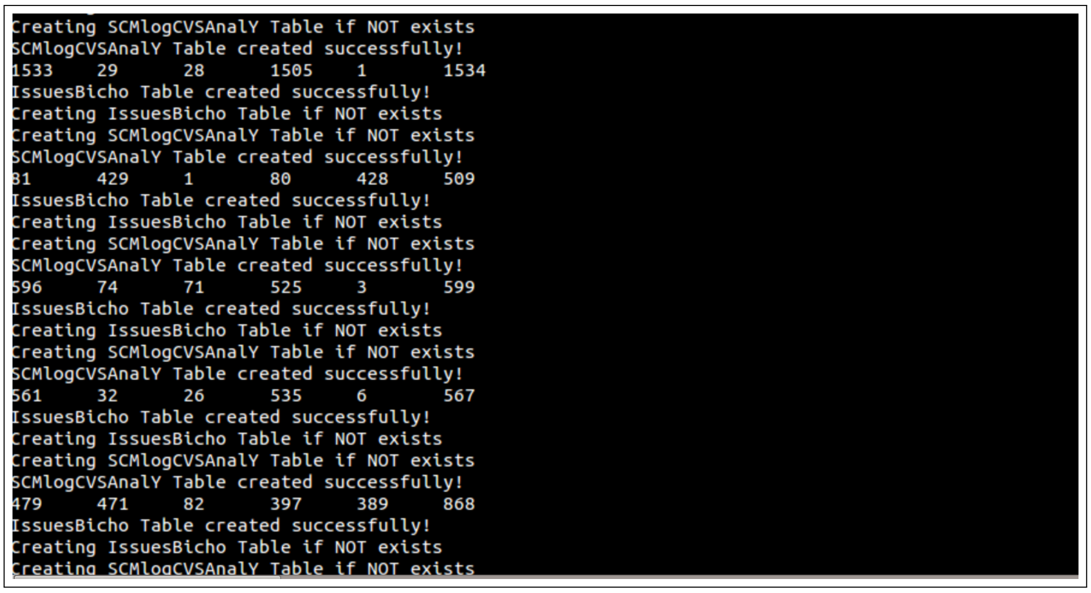
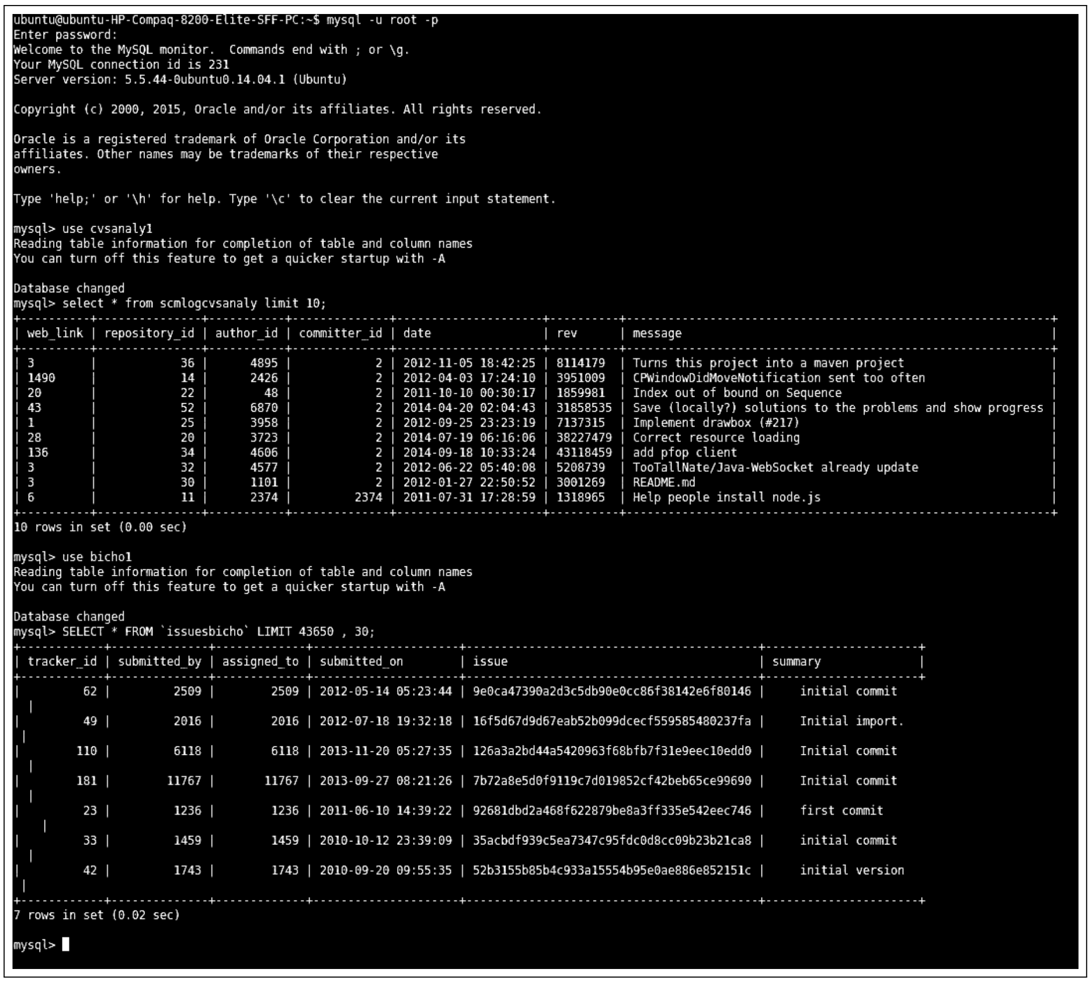


Figure 6.15: Bicho and CVSAnalY Delta



```
Creating SCMlogCVSAnalY Table if NOT exists
SCMlogCVSAnalY Table created successfully!
1533    29      28      1505    1       1534
IssuesBicho Table created successfully!
Creating IssuesBicho Table if NOT exists
Creating SCMlogCVSAnalY Table if NOT exists
SCMlogCVSAnalY Table created successfully!
81      429     1       80      428     509
IssuesBicho Table created successfully!
Creating IssuesBicho Table if NOT exists
Creating SCMlogCVSAnalY Table if NOT exists
SCMlogCVSAnalY Table created successfully!
596     74      71      525     3       599
IssuesBicho Table created successfully!
Creating IssuesBicho Table if NOT exists
Creating SCMlogCVSAnalY Table if NOT exists
SCMlogCVSAnalY Table created successfully!
561     32      26      535     6       567
IssuesBicho Table created successfully!
Creating IssuesBicho Table if NOT exists
Creating SCMlogCVSAnalY Table if NOT exists
SCMlogCVSAnalY Table created successfully!
479     471     82      397     389     868
IssuesBicho Table created successfully!
Creating IssuesBicho Table if NOT exists
Creating SCMlogCVSAnalY Table if NOT exists
```

Figure 6.16: T2



```
ubuntu@ubuntu-HP-Compaq-8200-Elite-SFF-PC:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 231
Server version: 5.5.44-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use cvsanaly1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from scmlogcvsanaly limit 10;
+----------+---------------+-----------+--------------+---------------------+----------+-----------------------------------------------------------+
| web_link | repository_id | author_id | committer_id | date                | rev      | message                                                   |
+----------+---------------+-----------+--------------+---------------------+----------+-----------------------------------------------------------+
| 3        |            36 |      4895 |            2 | 2012-11-05 18:42:25 | 8114179  | Turns this project into a maven project                   |
| 1490     |            14 |      2426 |            2 | 2012-04-03 17:24:10 | 3951009  | CPWindowDidMoveNotification sent too often                |
| 20       |            22 |        48 |            2 | 2011-10-10 00:30:17 | 1859981  | Index out of bound on Sequence                            |
| 43       |            52 |      6870 |            2 | 2014-04-20 02:04:43 | 31858535 | Save (locally?) solutions to the problems and show progress |
| 1        |            25 |      3958 |            2 | 2012-09-25 23:23:19 | 7137315  | Implement drawbox (#217)                                  |
| 28       |            20 |      3723 |            2 | 2014-07-19 06:16:06 | 38227479 | Correct resource loading                                  |
| 136      |            34 |      4606 |            2 | 2014-09-18 10:33:24 | 43118459 | add pfop client                                           |
| 3        |            32 |      4577 |            2 | 2012-06-22 05:40:08 | 5208739  | TooTallNate/Java-WebSocket already update                 |
| 3        |            30 |      1101 |            2 | 2012-01-27 22:50:52 | 3001269  | README.md                                                 |
| 6        |            11 |      2374 |         2374 | 2011-07-31 17:28:59 | 1318965  | Help people install node.js                               |
+----------+---------------+-----------+--------------+---------------------+----------+-----------------------------------------------------------+
10 rows in set (0.00 sec)

mysql> use bicho1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM `issuesbicho` LIMIT 43650 , 30;
+------------+--------------+-------------+---------------------+------------------------------------------+---------------------+
| tracker_id | submitted_by | assigned_to | submitted_on        | issue                                    | summary             |
+------------+--------------+-------------+---------------------+------------------------------------------+---------------------+
|         62 |         2509 |        2509 | 2012-05-14 05:23:44 | 9e0ca47390a2d3c5db90e0cc86f38142e6f80146 |     initial commit
  |
|         49 |         2016 |        2016 | 2012-07-18 19:32:18 | 16f5d67d9d67eab52b099dcecf559585480237fa |     Initial import.
 |
|        110 |         6118 |        6118 | 2013-11-20 05:27:35 | 126a3a2bd44a5420963f68bfb7f31e9eec10edd0 |     Initial commit
  |
|        181 |        11767 |       11767 | 2013-09-27 08:21:26 | 7b72a8e5d0f9119c7d019852cf42beb65ce99690 |     Initial commit
  |
|         23 |         1236 |        1236 | 2011-06-10 14:39:22 | 92681dbd2a468f622879be8a3ff335e542eec746 |     first commit
    |
|         33 |         1459 |        1459 | 2010-10-12 23:39:09 | 35acbdf939c5ea7347c95fdc0d8cc09b23b21ca8 |     initial commit
  |
|         42 |         1743 |        1743 | 2010-09-20 09:55:35 | 52b3155b85b4c933a15554b95e0ae886e852151c |     initial version
 |
+------------+--------------+-------------+---------------------+------------------------------------------+---------------------+
7 rows in set (0.02 sec)

mysql> █
```

Figure 6.17: Synchronised BT Data and VC logs into newer SCMlogcvsanaly and Issuesbicho

Figure 6.17 shows (i.e a snapshot of the tables in both tools databases) the result of the syncing process automatically in Bicho and CVSAnalY database for the OSS projects we sampled in this research.

6.9 Summary of the chapter

The synchronisation of Bicho and CVSAnalY tool sets is suitable for analysing open-source software projects and commercial projects, provided they are hosted or use the repository supported by Bicho and CVSAnalY respectively. However, the solution of automation and synchronisation of VC logs and BT data improve the quality of bug data in software repository reduced the impediment of incomplete, inconsistent and skewed sets of data that researchers used for empirical studies.

Furthermore, we also presented a framework for detecting and automatically synchronising bug-related data missing from these sources. The framework is easy to implement by following the steps and processes that are involved in mining VC logs and BT data. However, it can be a daunting and very challenging task, since the origin of such data sets is also not mirrored. In this way, the synchronisation will enable flexible cross-analyses of evolutionary aspects of OSS projects, since, using the tool chain, one can mine VC logs and BT data from VC systems and BT systems. Also, the synchronisation provides a simple query-result mechanism and supports complex data queries for analysis.

Chapter 7

Conclusion and Threats to Validity

7.1 Introduction

This thesis proposed a large empirical study that mines the VC logs and BT data of 344 OSS projects hosted on GitHub[1]. In addition, this thesis proposed a framework not only for "extracting", but also for automatically "syncing" VC logs and BT data supporting multiple BT systems and VC systems. The novelty of the framework, apart from supporting various OSS software repositories, is the ability to synchronise missing VC logs (concerning bugs) with data extracted from the BT system, and vice versa. The framework will assist in mining the complete set of software evolutionary facts throughout the entire life cycle of software projects; to provide complete data to bug-detection techniques; to assist the developers during the corrective software maintenance; and to provide an unbiased data set for empirical software engineering research on bugs.

In this chapter, Sections 7.2 and 7.3 summarise and highlight the contributions, beneficiaries and impacts of the thesis. In Section 7.4 of this chapter, we evaluate the contributions of the thesis. In addition, we will discuss the threats to validity that could call into question some of the findings of this thesis. These include the threats to internal, external and construct validity of our study in Section 7.5. We discuss our future work and detail possible extensions of the study reported in this thesis in Section 7.6. Finally, we discuss the conclusions based

[1]`https://github.com/`

on the problems we reported and investigated in each chapter of this thesis in Section 7.7.

7.2 Contributions of this thesis

Following the list of objectives and the problems observed as outlined in Chapter 1, the contributions of this thesis are as follows:

C1 – Tools. In this thesis, tools that trace **VC logs** and **BT data** for software projects were identified. After selecting Bicho and CVSAnalY to be used in this research, this thesis described VC logs and BT data structures of the tools selected and identified the fields that linked BT data and VC logs for synchronisation into their respective databases. Thus, researchers in software engineering can trace, mine VC logs and BT data using the identified tools collectively without mining and tracking VC logs and BT data independently.

C2 – Bugs in VC log. The thesis presented an in-depth analysis of VC logs using the SZZ algorithm, which has been used extensively by researchers to identify bugs in VC logs and BT data of software systems. In this thesis, the SZZ algorithm was partitioned in its three core components – the "bug" and "fix" keywords and "# +digit" – with a manual check-up. In addition, this thesis evaluated the precision and recall of the various parts of the SZZ algorithm and presented the precision and recall of each element in detecting bug identifiers in the development logs (VC logs). This thesis suggested using "# + digit" and the bug ID, which largely outperformed the other proxies in finding bugs in VC logs and BT data.

C3 – Discrepancies between BT data and VC logs. This thesis presented the results in a Venn diagram, which suggested that around 1/3 of the total number of VC logs and BT data were mirrored when cross-analysed and linked with BT data. Also, another 1/3 were only present in BT data retrieved by Bicho, while the rest were found in VC log data (CVSAnalY), but never summarised into BT data retrieved by a BT system tool (Bicho). This thesis present and conducted a large empirical study that mined the VC logs and BT data of 344 OSS projects, hosted on GitHub[2]. Thus, this thesis provided a large and significant statistical conclusion with reasonable evidence in the issue of traceability links recovery and syncing of VC log and BT data from open-source software repositories.

C4 – Synchronisation. The thesis presented a tool chain that synchronised VC logs and BT data, ensuring that data sets held by these tools (Bicho and CVSAnalY) are always complete and enriched effectively. Most importantly: (i) the tool chain avoids the impediment of using incomplete data sets for analysis in empirical software engineering; (ii) VC log and BT data can be identified and retrieved with higher precision; and (iii) consistent and unskewed data sets can be obtained, since the missing information in both tools is tracked and synchronised.

C5 – Tool chain. This thesis proposed and implemented a complete tool chain not only for extracting, but also for automatically syncing VC logs (development logs) and bugs of issue data (BT data) – that is, supporting multiple BT system and VC system.

The novelty of the tool chain, apart from the fact that it supports various OSS repositories, is its ability to synchronise missing VC logs (concerning bugs) with data extracted from the BT system, and vice versa. Finally, this tool chain was made available.

7.3 Beneficiaries and impact of this thesis

Following the contributions as outlined in Chapter 1 and recapitulated in Section 7.2, the beneficiaries and impact of this thesis are as follows:

1. **Open-source software (OSS) community** This thesis benefits OSS community and that aim to design and develop tools for retrieving VC logs and BT data collectively that (i) support various BT system and VC system sources; (ii) allow cross-analysis of BT data and VC logs; and (iii) track and synchronise missing BT data and VC logs of

[2]https://github.com/

software projects, ensuring that complete and consistent data sets are always stored in the database for posterior analysis.

2. **Researchers in software corrective maintenance:** Researchers in software maintenance and evolution benefit from this thesis, since the source of data most commonly used by researchers in software corrective maintenance is, by far, VC logs and BT data. Using the tool chain to extract data from various sources will help researcher by improving the quality of the data sets they used. Similarly, researchers can extract complete VC logs and BT data from various sources, and also understand the inner mechanisms of producing software artefacts that are required for research and analysis in software engineering.

3. **Researchers in empirical software engineering**: The novelty of the tool chain, apart from the fact that it supports various OSS software repositories, is its ability to synchronise missing development logs (concerning bugs) with data extracted from the BT system, and vice versa. As a result, researchers of bugs in empirical software engineering benefit from this thesis by using the tool chain in mining complete sets of evolutionary facts to provide an unbiased data set.

   In general, both large OSS and commercial projects can be analysed in order to extract and establish missing links and sync BT data with VC logs (and vice versa) for posterior analysis.

7.4 Evaluation of the thesis contribution

The research presented in this thesis aims to be an overarching discussion about how data on bugs are being extracted and used to inform studies on bug prediction, bug triaging and identification. The findings of this thesis do not confirm the hypotheses: bug IDs are not mirrored from bug trackers into VC logs and vice versa. Also, using the set of all bugs from bug-tracking systems is not always definitive in describing the overall set of bugs in a software system. Therefore, the traceability of bugs in open-source projects could benefit from the integration of two sources of information: one based on the VC logs, and one based on the BT data.

The four scenarios described in Chapter 5 Section 5.4 show an overarching problem in the traceability of bugs, which can be described as the “expressiveness” of an information source. VC logs should be expressive enough to follow the opening, fixing and closing of a bug closely and afterwards update the bug-tracking system as proof of what was achieved during the development itself. What we found from our sample of projects is that there is never a perfect match in what is recorded by developers in the different databases: what is more worrying is that the information source that is intended primarily to track defects and their resolution is often missing some pieces of information that are instead recorded in the development logs.

To be truly effective, our (and others’) approach of tracing bugs into VC logs should be integrated into a framework that not only detects and stores the discrepancies in the traceability of bugs into the VC logs, but also provides a means to synchronise (fill) the missing data in one data source if that data was to be found in the other data source. In (chapter 6), we presented our framework, which aims to integrate different types of repositories, and various approaches to bug notations. Similarly, Bicho and CVSAnalY (**before**) were run independently and producing independent results in which cross-analysing BT data and VC logs to track the missing link required significant amount of manual effort. In this thesis, we implement our framework using our approach and integrate Bicho (i.e., a BT tool) and CVSAnalY (i.e., a VC tool) functionalities. In addition, we evaluate the framework in which 98% of the missing BT data and VC logs were circumvented in their respected databases automatically as we reported in Chapter 6 of the this thesis.

In addition, **before** the implementation of the framework we used the approach presented in this thesis to track missing data of 344 OSS project. The result of the analysis presented in Chapter 5 Section 5.3 ()Figure 7.1,) are mirrored the box-plot indicated that in 75% of 344 OSS projects sampled in this research, no more than 20% of the overall number of detected bug IDs (i.e., not mirrored in Bicho and CVSAnalY database).

However, In most of the 344 OSS projects we sampled in this research, the majority of missing BT data and VC log were circumvented automatically in both Bicho and CVSAnalY

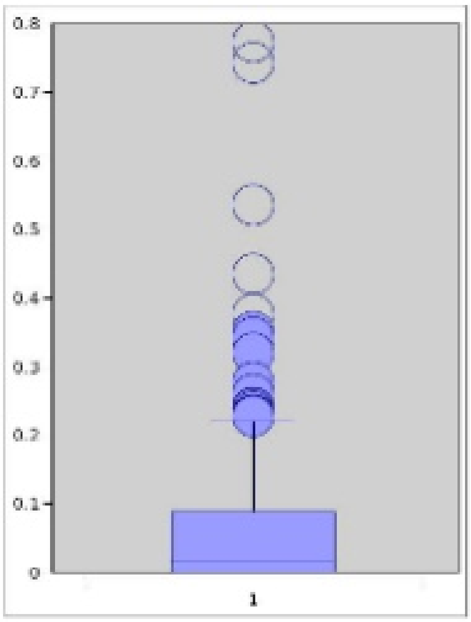


Figure 7.1: Before: Ratio of bug IDs mentioned in both development logs and bug trackers,per project

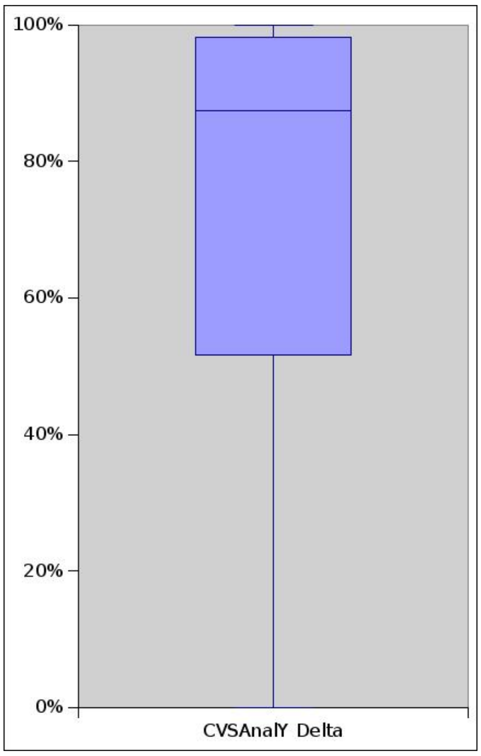


Figure 7.2: After: Ratio of bug IDs mentioned in development logs per project (in 344 OSS projects)

respective databases. **After** the implementation of our framework and approach, the box plot in Figure 7.2, shows the set of bug IDs detected that were inserted into newer *SCMlogcvsanaly* table of CVSAnalY database created in the database. This resulted in around 75% of projects bug IDs found in Bicho but not in CVSAnalY **before** was synchronised. Thus, the shared bug coverage of bug IDs was high and it shows no outlier in the box plot presented in Figure 7.2.

7.5 Threats to validity

In this section, we discuss the threats to the validity of this research. This includes internal, external, construct and conclusion validity. These threats are defined as follows.

- Internal validity is defined as the accuracy of the conclusion about the study in this research [90].
- External validity is defined as the generalised validity of the conclusions of the research in this thesis [90].
- Construct validity refers to the degree to which a conclusion can be made following the theoretical constructs on which the approach was based [90].
- Conclusion validity in this thesis is defined as a factor that can influence and lead the findings in this thesis to an incorrect conclusion.

7.5.1 Threats to validity (Chapter 3)

In this section, we will discuss the threats to validity that are specific to **our approach and finding in** Chapter 3 of this thesis.

7.5.1.1 Internal validity

The selection of OSS projects and extraction of VC logs and BT data was very time-consuming and tedious. However, the mining process was particularly slow due to the sleep-time we imposed between each OSS project extraction step in both Bicho and CVSAnalY. For instance, a sleep-time of 15 seconds was regularly imposed after VC logs data of one OSS project had been extracted and before moving on to Bicho to extract BT data of the same OSS project in order to avoid an unfriendly stress on the BT system and VC system server (refer to Appendix B in A.1 for a working copy of the tool chain in lines 23–24 for the sleep-time fragment of code).

The reliability of VC logs and BT data is also a potential issue: certainly one can never be sure that the repositories hold correct and complete VC logs and BT data. Thus we set some criteria and requirement as mentioned in Chapter 3 Section 3.2 of this thesis. For instance requirements include: every OSS project sampled in this thesis the projects must be maintained and remain under active development. This is to ensures that the analysed VC logs and BT data were not obsolete. Because extracting incomplete or inconsistent VC logs and BT data of OSS projects can lead to a biased and untrustable result and incorrect analysis [112]. Since BT systems and VC systems are not in sync, in this case, it is hard to ensure that all related data are collected. For instance, some OSS projects might have another source code or bug repositories that are not made publicly available. In this case, we might obtain empty VC logs or BT data.

There is also a threat to SZZ algorithm validity. We implemented the SZZ algorithm on the basis that SZZ is currently the best available algorithm for automatically identifying VC logs on BT data [128]. We cannot guarantee that SZZ is still the best algorithm. Although we improved the approach (SZZ algorithms) that we applied in this research, the approach may also be subject to implementation errors. We tried to minimise this threat by piloting on one OSS project (i.e., the Bracket project), as detailed in the working example in Section 3.6, before applying the approach to all the 344 OSS projects. In addition, we further validated our technique through extensive manual check-up during our analysis and implementation as detailed and reported in Chapter 4 Section 4.3.2 in this thesis. We performed a manual analysis of a random sample of 100 VC logs of 10 OSS projects we sampled in order to determine whether “Fix" or “Bug" or the # identifier are referring to a bug.

7.5.1.2 External validity

In Chapter 3, we check in Bicho and CVSAnalY databases if any over-lagging exist in their respective databases using the SZZ algorithm and traces of VC logs and BT data. . In addition, we cannot guarantee that the obtained results are generalisable on the OSS projects sampled from GitHub. Kalliamvakou et al [71] asserted that “One of the biggest threats to validity to any study that uses GitHub data indiscriminately is the bias”, because most of the repositories that are developed on GitHub are personal and inactive repositories. However, the data we obtained in this research and the selected OSS projects were active projects. We mitigated this threat by imposing some criteria and requirements which excluded non-active projects from this research. Thus, to the best of our knowledge, this increased our confidence on the approach and the results we obtained using the selected OSS projects. Moreover, the tools we selected might not have been the right tools for some projects. For instance, some developers might not mention a bug report ID in the *message* field of the *SCMlog* table in CVSAnalY, while the ID exists in the *summary* field of the *Issues* table of Bicho or in different patches that are handled via a mailing list, rather than through the BT system.

7.5.2 Threats to validity (Chapter 4)

In this section, we will discuss the threats to validity that are specific to our **finding** in Chapter 4 of this thesis.

7.5.2.1 Internal validity

With respect to internal validity, the evaluation was between VC logs retrieved using CVS-AnalY and BT data retrieved using Bicho. Both tools are executed independently and produce independent results. Similarly, the VC logs and BT data are stored in different localised databases created by both tools automatically. The extraction process – that is to say, mining VC logs and BT data of each project’s data set – was carried out simultaneously to avoid any discrepancies or over-lagging using the tool chain. This allowed us to evaluate and dissect each individual 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.

7.5.2.2 Construct validity

With respect to construct validity, which deals with the relation between the 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.

In this thesis, we have evaluated the precision and recall of the individual 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 as well as showing their p-value).

To mitigate such a threat, we began with a pilot study, in which we studied 10 OSS projects and manually analysed each VC log to determine if "Fix" or "Bug" or the # identifier were referring to a bug. After successful completion of the pilot study, we extended the study to a large number (344) of OSS projects sampled from GitHub. But in this case, the results varied significantly, considering that analysis of the 10 OSS projects was carried out manually. Also, in some OSS projects only the top 100 subsets of VC logs were considered when evaluating each component of the SZZ algorithm, while the rest of the 10 OSS projects had fewer than 100 VC logs. However, where the proportion of the three main component of the SZZ algorithm (i.e., # symbol, fixed and bug) were zeros from **Table 4** in Appendix 7.7. Section A.7 none of the logs retrieved in that project referred to the TP and FP as mentioned in the previous Section 4.5 and defined in Section 4.2 of this Chapter. In some of the 10 OSS projects analysed manually, only the top 100 subsets of VC logs were considered in evaluating each component, while the rest of the 10 OSS projects had fewer than 100 VC logs.

7.5.2.3 External validity

We welcome researchers in empirical software engineering to build on the results in this thesis and replicate our study with different and large OSS projects 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.

The results from this study are only generalisable to Bicho and CVSAnalY tool sets and the 344 OSS 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.

7.5.2.4 Conclusion validity

With respect to conclusion validity, due to the large number of OSS projects we sampled in this study, as well as non-normality of VC logs and BT data sets, we used the Mann-Whitney test to prove the significance of each individual SZZ algorithm component [59].

7.5.3 Threats to validity (Chapter 5)

In this section, we will discuss the threats to validity that are specific to our **finding** in Chapter 5 of this thesis.

7.5.3.1 Internal validity

With respect to internal validity, we conducted an in-depth analysis between VC logs retrieved using CVSAnalY and BT data retrieved using Bicho. The extraction process – that is to say, mining VC logs and BT data of each project's data set – was carried out simultaneously to avoid any discrepancies or over-lagging. This allowed us to quantify and identify each of the OSS project data sets (VC logs and BT data) we sampled in this study with careful considerations to the best of our knowledge. This was to minimise any other external factors that might have had an effect on the results in our empirical study.

7.5.3.2 Construct validity

With respect to construct validity, our aim was to quantify and identify the discrepancies of large OSS projects to provide significant evidence that VC logs and BT data are not mirrored in OSS projects. We evaluated the union and intersection of the sets for each project. Given a set of bug IDs mentioned in the VC logs, and the list of bug IDs stored from the BT system 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.

To mitigate such a threat, we began with a pilot study in which we studied 10 OSS projects (Brackets) and manually mirrored all the VC logs and BT data that exist in Bicho and CVSAnalY.

7.5.3.3 External validity

We welcome researchers in empirical software engineering to build on the results in this thesis and replicate our study with large OSS projects using our approach to advance this body of knowledge.

Our results from this study are only generalisable to Bicho and CVSAnalY tool sets and the OSS 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.

7.5.3.4 Conclusion validity

With respect to conclusion validity, due to the large number of OSS projects we sampled in this study, we mitigated this threat by conducting an in-depth analysis on the four scenarios of bug coverage reported in Chapter 5 Section 5.4. Thus, we randomly selected 37 OSS projects out of 344. This represented 10 OSS projects for three of the scenarios and seven OSS projects for one scenario that we observed based on the metric we formulated called shared bug coverage. The worked examples presented for each scenario confirmed that discrepancies exist between the data held in Bicho and CVSAnalY and suggested that BT data and VC logs of OSS projects are not mirrored.

7.5.4 Threats to validity (Chapter 6)

In this section, we will discuss the threats to validity that are specific to our **finding** in Chapter 6 of this thesis.

7.5.4.1 Internal

Internal validity is defined as the accuracy of the conclusion in this research [90]. The threats to internal validity in Chapter 6 were the synchronised BT data and VC logs of two entities – that is to say, the SCMlog table and the Issues table of CVSAnalY and Bicho respectively. The synchronisation was in two forms: in Test 1 (T1) we synced the missing BT data in the **Issues** table of Bicho into the **SCMlog** table of CVSAnalY, and also synced missing VC logs in the **SCMlog** table of CVSAnalY into the **Issues** table of Bicho. Unfortunately, as we envisaged, the occurrence of duplicate entries of BT data was not mirrored in the CVSAnalY database, as we mentioned in Section 6.6.

To mitigate such a threat, we did not intervene in the existing databases (Bicho and CVSAnalY) and tables (**Issues** and **SCMlog**). We synchronised the missing BT data and VC logs in a different integrated table within the respective CVSAnalY and Bicho databases. This was achieved using the set operation function visible in Code 6.5 in Section 6.5 in the penultimate chapter of this thesis. Thus, we realigned the Bicho and CVSAnalY databases by adding two extra tables in their respective databases – namely the **SCMlogCVSAnalY** table in CVSAnalY and the **IssuesBicho** table in Bicho.

7.5.4.2 Construct validity

The construct validity in Chapter 6 was the SZZ algorithm [116] that we applied to track and sync bugs and logs of the 344 OSS projects sampled and obtained from GitHub. In our formulation, we only looked for bugs described by the "#" sign and various numeric values (e.g., #1234) which were linked to the ID of a bug. In its original formulation, the SZZ algorithm also searches for keywords like “Bug”, “Fixed” and others. We mitigated such a threat by conducting an analysis and evaluating 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) to check their precision and recall when isolating the bug IDs in the VC logs.

7.6 Future work

In this section, we present our future work based on threats to validity we reported for each chapter.

7.6.1 Empirical studies

The possible extension of this study with different and large OSS projects to be sampled from different repositories is among our future work. As we mentioned in the previous section of this chapter, we want to conduct a blind analysis [115] and replicate our study with commercial projects using our approach in order to advance this body of knowledge.

Since the results from this study are only generalisable to Bicho and CVSAnalY tool sets and the 344 OSS projects we sampled from GitHub via FlossMole, we also plan to apply our approach to other MSR tools we mentioned in this study in Chapter 2.6, in order to validate the generalisation of our findings related to discrepancies in OSS projects we reported in Chapter 5.3.

7.6.2 Tool sets

As mentioned above, the results from this study are only generalisable to Bicho and CVSAnalY tool sets and the 344 OSS projects we sampled from GitHub via FlossMole. As stated in the previous section of this chapter, we do claim our approach might be applicable to the rest of the tools we mentioned in Chapter 2.6. Thus, we plan to implement our framework using the same approach to merge and synchronise the missing BT data and VC logs using the tools recovered in their respective databases. The implementation is made easier by the flexibility of our framework following the steps we highlighted in Chapter 6.3. Nevertheless, this will require a significant amount of effort to be achieve.

7.6.3 Tool-chain

As stated in Chapter 6.7, the ultimate goal of this research is to automate and synchronise BT data and VC logs from different sources. This is a big challenge that requires a complex method [106]. In this thesis, we reported that the size, and the number of developers has an effect in traceability of bugs in VC logs. Obtaining complete sets of data is very crucial in empirical software engineering research that deals with: prediction of software faults, software reliability and traceability, software quality, effort and cost estimation, bug prediction, and bug fixing. Thus, it is crucial to provide them with a framework and tool chain that aims to support the integration, tracking and syncing of BT data and VC logs of multiple sources. For instance, Bicho supports Bugzilla (> 4), Sourceforge.net (abandoned), Jira (unstable), Launchpad, Allura (unstable). Moreover, the implementation of the framework was carried out by developing a tool chain – that is to say, secondary software that is considered to be cost effective [92], because it does not require a significant amount of resources and time in order to be developed, given the size, time and effort needed to develop a tool with a user interface. Thus, we plan to present the tool chain with a graphical user interface that supports the integration, tracking and syncing of BT data and VC logs of multiple sources in a single platform. As we mentioned in Chapter 6.4, the tool chain will be capable of executing Bicho and CVSAnalY by querying specific entities in their respective databases. In addition, it will be able to recover and synchronise the missing data in their respective databases and vice versa.

7.7 Thesis conclusion

In this section, we discuss the conclusions based on the problem statement that were articulated into various chapters of this thesis. In addition, we will discuss and reflect the aims and objectives that motivate this research in which we outlines in Chapter 1 of this thesis.

In chapter 2 we reported the related work, techniques and tools that aim to retrieve VC logs and BT data from BT system and VC system. The contribution of the presented research is the framework to synchronise the missing VC logs and BT data, supporting various repositories and bug-tracing algorithms and approaches [108]. In addition, we report on the evaluation of the existing techniques and approaches to solving the traceability issues in linking of VC logs and BT data of software projects by [88], who suggests that the use of regular expressions might work well. They compared the effectiveness of regular expressions with that of other well-known bug-linking techniques and tools, such as ReLink by [130] and BuCo Reporter by [83]. Their results suggest the technique and tools are equally as effective as other proposed techniques in solving the issue of traceability links.

In conclusion, all the BT tools and VC tools reported in Chapter 2 retrieved VC logs and BT data independently and required a large amount of interaction. Others – such as BuCo Reporter, Bug-code Analyser, Linkster and ReLink – recover missing logs and bugs/issues accurately. Unfortunately, they are unable to synchronised BT data and VC logs in their respective databases. Therefore, our approach, and the proposed framework the tool chain, goes one step further and completed these tools by synchronising the missing VC logs and BT data in either database in an automatic way. Thus we achieved our objective (Obj1) and discovered **what researchers use in mining VC logs and BT data.**

In chapter 3 of the thesis, we presented a procedure and our approach to extract, compare and synchronise the gaps discovered in either the VC logs or the BT data of OSS projects. We showed that such an approach has been partially automated when partially implementing a well-known algorithm to isolate the bug-fixing commits (i.e., the SZZ algorithm [116]).

This chapter outlined an approach to building a complete set of bug IDs that were documented in the evolution of a software system. This comprises the analysis and parsing of both the VC logs and the BT data: this is required because we found that commonly OSS projects hold different sets of bug IDs when interrogating the BT system and the development logs.

In addition, we conducted an in-depth analysis of the SZZ algorithm, which has been used extensively by researchers to track the bug-fixing commits of software systems. We partitioned the algorithm into 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 in detecting bug-fixing commits.

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 synchronised. The possible way to do this would be to generate an automatic commit in the development logs that details the bug-fixing activity, as obtained by the BT system. In the same way, when the BT system is not aligned with the VC logs, an entry could be automatically generated to insert the bug-development activity, as detailed in the VC logs, into the BT system.

In Chapter 4 of thesis, we demonstrated that the process of collecting data related to bugs, when using open-source projects, is far from established or repeatable. 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.

The results in this chapter 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 should reinforce the message for practitioners and researchers when identifying bugs in VC logs of OSS projects to use the standard # notation for bug IDs. Thus we achieved our objective (Obj2) and **identify bugs (BT data) into VC logs** in this chapter.

In Chapter 5 we presented the results of an extended quantitative analysis on a sample of 344 OSS projects, and how the bug-related data is stored in the VC logs and the BT data. The set of bug IDs from the VC logs was compared to the set of bug IDs found in the BT systems. The objective of the research in this chapter was to ascertain how much discrepancy is visible when considering these two sources of information, and whether either could be considered as a complete and credible set of data regarding bug issues.

We found that over half of the 344 OSS projects we analysed have a portion of bug IDs mentioned in one source (either the development logs or the bug-tracking logs) but not in the other. We also found that the intersection of "common" or shared bug IDs is very low (around 20% for some 75% of the projects in the sample), while in some extreme cases projects hold a distinct set of IDs in one database that is not shared in the other database. Furthermore, we also presented a framework for detecting and automatically synchronising missing bug-related data from these sources. Thus we achieved our objective (Obj3) and detected the **discrepancies between VC logs and BT data.**

In Chapter 6 The integration and combining the functionality of Bicho and CVSAnalY tool sets is suitable for analysing open-source software projects and commercial projects provided they are hosted or used the repository supported by Bicho and CVSAnalY respectively as mentioned in Chapter 2. The solution of automation and synchronisation of VC logs and BT data reduces the impediment of incomplete, inconsistent and skewed data sets that researchers use for empirical studies. Thus we achieved our objective (Obj4) and **synchronised** the missing data from one data source by using the traces found in the other source (i.e, either in Bicho or CVSAnaly vice versa).

In addition, we implement our proposed a framework (Published in the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE) 2015). that is easy to implement following the steps and process that we outlined in the structure of the framework [108]. However, it is a daunting task and very challenging, because the origin of such data sets is also not in synchronised. In this way, the synchronisation will enable flexible cross-analyses of evolutionary aspects of OSS projects, since Bicho and CVSAnalY are capable of mining VC logs and BT data from VC systems and BT systems. Also, after the implementation and synchronisation. The tool chain provides a simple query-result mechanism and supports complex data queries for analysis. Thus we achieved our objective (Obj5) and developed a **tool chain** that automatically detects, synchronises and re-engineers missing data and discrepancies in VC logs and BT data of 344 OSS projects.

Finally, we summarise evaluate the framework and the tool chain by graphically presented the percentage of BT data and VC log been recovered and synchronised automatically in Bicho and CVSAnalY respective databases using our approach for all the 344 OSS projects we sampled in this research. In general 80-95% of the missing BT data and VC logs of 344 OSS projects we sampled in this research has been recovered in Bicho and CVSAnalY respective database.

Bibliography

[1] Ieee standard for software maintenance. *IEEE Std 1219-1998*, pages i–, 1998.

[2] Ieee standard classification for software anomalies. *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, pages 1–23, Jan 2010.

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

[4] G. Antoniol, B. Caprile, A. Potrich, and P. Tonella. Design-code traceability for object-oriented systems. *Annals of Software Engineering*, 9(1-4):35–58, Jan. 2000.

[5] G. Antoniol, M. Di Penta, H. Gall, and M. Pinzger. Towards the integration of versioning systems, bug reports and source code meta-models. *Electronic Notes in Theoretical Computer Science*, 127(3):87–99, 2005.

[6] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA, 2006. ACM.

[7] J. Anvik and G. C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology*, 20(3):10:1–10:35, Aug. 2011.

[8] L. J. Arthur. *Software Evolution: The Software Maintenance Challenge.* Wiley-Interscience, New York, NY, USA, 1988.

[9] D. Atkins, T. Ball, T. Graves, and A. Mockus. Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering*, 28:324–333, 2002.

[10] 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.

[11] A. Bachmann and A. Bernstein. Software process data quality and characteristics: A historical view on open and closed source projects. In *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, IWPSE-Evol '09, pages 119–128, New York, NY, USA, 2009. ACM.

[12] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: Bugs and bug-fix commits. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 97–106, New York, NY, USA, 2010. ACM.

[13] K. H. Bennett and V. T. Rajlich. Software maintenance and evolution: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 73–87, New York, NY, USA, 2000. ACM.

[14] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann. Quality of bug reports in eclipse. In *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, eclipse '07, pages 21–25, New York, NY, USA, 2007. ACM.

[15] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pages 308–318, New York, NY, USA, 2008. ACM.

[16] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful...; really? In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 337–345, Sept 2008.

[17] 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.

[18] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143. ACM, 2006.

[19] T. F. Bissyandé, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Reveillere. Empirical evaluation of bug linking. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 89–98. IEEE, 2013.

[20] B. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, May 1988.

[21] P. Bourque and R. Dupuis. Guide to the software engineering body of knowledge 2004 version. *Guide to the Software Engineering Body of Knowledge, 2004. SWEBOK*, pages – , 2004.

[22] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, pages 301–310, New York, NY, USA, 2010. ACM.

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

[24] G. Canfora and L. Cerulo. Fine grained indexing of software repositories to support

impact analysis. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 105–111, New York, NY, USA, 2006. ACM.

[25] A. Capiluppi, C. Boldyreff, and K.-J. Stol. Successful reuse of software components: A report from the open source perspective. In S. Hissam, B. Russo, M. de Mendonça Neto, and F. Kon, editors, *Open Source Systems: Grounding Research*, volume 365 of *IFIP Advances in Information and Communication Technology*, pages 159–176. Springer Berlin Heidelberg, 2011.

[26] 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.

[27] J. M. Chambers. *Graphical methods for data analysis.* 1983.

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

[29] C. S. Corley, N. A. Kraft, L. H. Etzkorn, and S. K. Lukins. Recovering traceability links between source code and fixed bugs via patch analysis. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '11, pages 31–37, New York, NY, USA, 2011. ACM.

[30] 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.

[31] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *Software Engineering, IEEE Transactions on*, 31(6):446–465, 2005.

[32] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286, New York, NY, USA, 2012. ACM.

[33] M. D'Ambros, M. Lanza, and M. Pinzger. "a bug's life" visualizing a bug database. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 113–120, June 2007.

[34] S. Davies and M. Roper. Bug localisation through diverse sources of information. In *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on*, pages 126–131, Nov 2013.

[35] S. Davies and M. Roper. What's in a bug report? In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 26:1–26:10, New York, NY, USA, 2014. ACM.

[36] S. Davies, M. Roper, and M. Wood. Using bug report similarity to enhance bug localisation. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 125–134, Oct 2012.

[37] B. de Alwis and J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, CHASE '09, pages 36–39, Washington, DC, USA, 2009. IEEE Computer Society.

[38] D. Draheim and L. Pekacki. Process-centric analytical processing of version control data. In *Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of*, pages 131–136, Sept 2003.

[39] N. Fenton, S. L. Pfleeger, and R. L. Glass. Science and substance: A challenge to software engineers. *IEEE Software*, 11(4):86–95, 1994.

[40] J. Fernandez-Ramil, A. Lozano, M. Wermelinger, and A. Capiluppi. Empirical studies of open source evolution. In *Software evolution*, pages 263–288. Springer Berlin Heidelberg, 2008.

[41] M. Fischer, M. Pinzger, and H. Gall. Analyzing and relating bug report data for feature

tracking. In *Proceedings of the 10th Working Conference on Reverse Engineering*, WCRE '03, pages 90–, Washington, DC, USA, 2003. IEEE Computer Society.

[42] 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.

[43] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 190–, Washington, DC, USA, 1998. IEEE Computer Society.

[44] H. Gall, M. Jazayeri, and J. Krajewski. Cvs release history data for detecting logical couplings. In *Proceedings of the 6th International Workshop on Principles of Software Evolution*, IWPSE '03, pages 13–, Washington, DC, USA, 2003. IEEE Computer Society.

[45] M. Gegick, P. Rotella, and T. Xie. Identifying security bug reports via text mining: An industrial case study. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 11–20, May 2010.

[46] D. M. German. Mining cvs repositories, the softchange experience. *Evolution*, 245(5,402):92–688, 2004.

[47] T. Gilb. Evolutionary development. *SIGSOFT Software Engineering Notes*, 6(2):17–17, Apr. 1981.

[48] M. W. Godfrey, A. E. Hassan, J. Herbsleb, G. C. Murphy, M. Robillard, P. Devanbu, A. Mockus, D. E. Perry, and D. Notkin. Future of mining software archives: A roundtable. *IEEE Transactions on Software Engineering*, 26(1):67–70, Jan. 2009.

[49] J. M. Gonzalez-Barahona, D. Izquierdo-Cortazar, G. Robles, and A. del Castillo. Analyzing gerrit code review parameters with bicho. *Electronic Communications of the EASST*, 2014.

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

[51] 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.

[52] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. "not my bug!" and other reasons for software bug report reassignments. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, CSCW '11, pages 395–404, New York, NY, USA, 2011. ACM.

[53] 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.

[54] T. Hall, D. Bowes, S. Counsell, L. Moonen, and A. Yamashita. Software fault characteristics: A synthesis of the literature. 2015.

[55] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.

[56] A. E. Hassan and T. Xie. Software intelligence: The future of mining software engineering data. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 161–166, New York, NY, USA, 2010. ACM.

[57] 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.

[58] H. Hemmati, S. Nadi, O. Baysal, O. Kononenko, W. Wang, R. Holmes, and M. Godfrey. The msr cookbook: Mining a decade of research. *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013.

[59] J. J. Higgins. *Introduction to modern nonparametric statistics.* Cengage Learning, 2003.

[60] K. Hinsen, K. Läufer, and G. K. Thiruvathukal. Essential tools: Version control systems. *Computing in Science Engineering*, 11(6):84–91, Nov 2009.

[61] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 34–43, New York, NY, USA, 2007. ACM.

[62] J. Howison, M. Conklin, and K. Crowston. Flossmole: A collaborative repository for floss research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3):17–26, 2006.

[63] International Standards Organisation (ISO). *Standard 14764 on Software Engineering - Software Maintenance*. ISO/IEC, 1999.

[64] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 52–61, June 2008.

[65] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 111–120, New York, NY, USA, 2009. ACM.

[66] S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 82–85, Sept 2008.

[67] H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, 2007.

[68] H. Kagdi, J. Maletic, and B. Sharif. Mining software repositories for traceability links. In *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, pages 145–154, June 2007.

[69] H. Kagdi, J. Maletic, B. Sharif, et al. Mining software repositories for traceability links. In *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*, pages 145–154. IEEE, 2007.

[70] H. H. Kagdi. *Mining software repositories to support software evolution.* PhD thesis, Kent State University, 2008.

[71] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github (extended version). 2015.

[72] J. Kealey and G. Mussbacher. Statsvn: Statistics for svn repositories based on the open source project statcvs.

[73] M. Kim. Developer's toolbox: A guide to version control for magento using git and beanstalk. 2014. Retrieved February 15, 2015 from http://gotgroove.com/ecommerce-blog/guide-to-version-control-for-magento-using-git-and-beanstalk/.

[74] S. Kim, T. Zimmermann, M. Kim, A. Hassan, A. Mockus, T. Girba, M. Pinzger, E. J. Whitehead, Jr., and A. Zeller. Ta-re: An exchange language for mining software repositories. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 22–25, New York, NY, USA, 2006. ACM.

[75] 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.

[76] A. J. Ko and P. K. Chilana. How power users help and hinder open bug reporting. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1665–1674, New York, NY, USA, 2010. ACM.

[77] G. Kuk. Strategic interaction and knowledge sharing in the kde developer mailing list. *Management Science*, 52(7):1031–1042, 2006.

[78] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. Predicting the severity of a reported bug. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 1–10, May 2010.

[79] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

[80] 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.

[81] M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, Sept 1980.

[82] M. M. Lehman and L. A. Belady, editors. *Program Evolution: Processes of Software Change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[83] 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.

[84] B. Livshits and T. Zimmermann. Dynamine: Finding common error patterns by mining software revision histories. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 296–305, New York, NY, USA, 2005. ACM.

[85] 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.

[86] J. Matsuda, S. Hayashi, and M. Saeki. Hierarchical categorization of edit operations for separately committing large refactoring results. 2015. Retrieved August 30, 2015 from http://www.se.cs.titech.ac.jp/ hayashi/pub/jmatsu-iwpse2015.pdf.

[87] G. Mausa, T. G. Grbac, and B. D. Basic. Software defect prediction with bug-code analyzer-a data collection tool demo. In *Software, Telecommunications and Computer Networks (SoftCOM), 2014 22nd International Conference on*, pages 425–426. IEEE, 2014.

[88] G. Mausa, P. Perkovic, T. G. Grbac, and I. Stajduhar. Techniques for bug-code linking. In *SQAMIA'14*, pages 47–55, 2014.

[89] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 346–355, Sept 2008.

[90] M. Mitchell and J. Jolley. *Research design explained*. Cengage Learning, 2012.

[91] 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.

[92] L. Morgan and P. Finnegan. Benefits and drawbacks of open source software: an exploratory study of secondary software firms. In *Open Source Development, Adoption and Innovation*, pages 307–312. Springer, 2007.

[93] G. C. Murphy and D. Cubranic. Automatic bug triage using text categorization. In *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer, 2004.

[94] G. C. Murphy, D. Notkin, and K. J. Sullivan. Software reflexion models: Bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, 27(4):364–380, Apr. 2001.

[95] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

[96] P. Oman and T. Lewis. *Milestones in software evolution*. IEEE Computer Society Press reprint collection. IEEE Computer Society Press, 1990.

[97] K. Pan, S. Kim, and E. J. Whitehead, Jr. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14(3):286–315, June 2009.

[98] L. D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 29–, Washington, DC, USA, 2007. IEEE Computer Society.

[99] C. R. Reis and R. P. de Mattos Fortes. An overview of the software engineering process and tools in the mozilla project, 2002.

[100] P. C. Rigby and A. E. Hassan. What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 23. IEEE Computer Society, 2007.

[101] P. C. Rigby and A. E. Hassan. What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 23–, Washington, DC, USA, 2007. IEEE Computer Society.

[102] P. Rob and C. Coronel. *Database Systems: Design, Implementation, and Management*. Course Technology Press, Boston, MA, United States, 8th edition, 2007.

[103] G. Robles. Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 171–180, May 2010.

[104] G. Robles and J. M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. *SIGSOFT Software Engineering Notes*, 30(4):1–5, May 2005.

[105] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. Herraiz. Tools and datasets for mining libre software repositories. *Multi-Disciplinary Advancement in Open Source Software and Processes*, page 24, 2011.

[106] 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.

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

[108] 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.

[109] 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*, OpenSym '14, pages 8:1–8:4, New York, NY, USA, 2014. ACM.

[110] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[111] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 499–510, Washington, DC, USA, 2007. IEEE Computer Society.

[112] M. Shepperd. How do i know whether to trust a research result? *Software, IEEE*, 32(1):106–109, 2015.

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

[114] J. S. Shirabad, T. C. Lethbridge, and S. Matwin. Mining the maintenance history of a legacy software system. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 95–104. IEEE, 2003.

[115] B. Sigweni and M. Shepperd. Using blind analysis for software engineering experiments. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, pages 32:1–32:6, New York, NY, USA, 2015. ACM.

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

[117] G. M. Sullivan and R. Feinn. Using effect size-or why the p value is not enough. *Journal of graduate medical education*, 4(3):279–282, 2012.

[118] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 253–262, Nov 2011.

[119] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 45–54, New York, NY, USA, 2010. ACM.

[120] A. Sureka and P. Jalote. Detecting duplicate bug report using character n-gram-based features. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 366–374. IEEE, 2010.

[121] 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.

[122] Y. Tian, C. Sun, and D. Lo. Improved duplicate bug report identification. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 385–390, March 2012.

[123] W. F. Tichy. Rcs—a system for version control. *Software: Practice and Experience*, 15(7):637–654, 1985.

[124] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 461–470, New York, NY, USA, 2008. ACM.

[125] J. Weidl and H. Gall. Binding object models to source code: an approach to object-oriented re-architecting. In *Computer Software and Applications Conference, 1998. COMPSAC '98. Proceedings. The Twenty-Second Annual International*, pages 26–31, Aug 1998.

[126] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on*, pages 1–1, May 2007.

[127] P. Wessa. Free statistics software, office for research development and education. 2016.

[128] C. Williams and J. Spacco. Szz revisited: Verifying when changes induce fixes. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, DEFECTS '08, pages 32–36, New York, NY, USA, 2008. ACM.

[129] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[130] 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.

[131] T. Xie. Bibliography on mining software engineering data. 2014. Retrieved February 15, 2014 from https://sites.google.com/site/asergrp/dmse.

[132] T. Xie and D. Notkin. Mutually enhancing test generation and specification inference. In A. Petrenko and A. Ulrich, editors, *Formal Approaches to Software Testing*, volume 2931 of *Lecture Notes in Computer Science*, pages 60–69. Springer Berlin Heidelberg, 2004.

[133] T. Xie, S. Thummalapenta, D. Lo, and C. Liu. Data mining for software engineering. *Computer*, 42(8):55–62, 2009.

[134] 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.

[135] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, Sept. 2004.

[136] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 14–24, June 2012.

[137] Y. Zhou and J. Davis. Open source software reliability model: an empirical approach. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–6. ACM, 2005.

[138] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu. Improving bug tracking systems. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 247–250, May 2009.

[139] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, pages 9–9. IEEE, 2007.

[140] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society.

[141] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429–445, 2005.

Appendix A

A.1 Tool-chain: Mining VC logs and BTS Data of 344 OSS Projects

```
#!/usr/bin/perl -w

@all = `cat GH-344-OSS-Projects.txt`;
foreach $line(@all){
    chomp $line;
    $svn_line = $line;
    $project = $line;

    $project =~ s/.*\s//;
        $project =~ s/.*\///;
        $project =~ s/\.git//;
        $num= 5;
    $project_bicho = $line;
    $project_bicho =~ s/.*github.com\///;
    $project_bicho =~ s/\.git//;


     print "\t\tExecuting␣cvsanaly...\n";
    `exec $svn_line`;
    `cvsanaly2 −u root −p password −d cvsanaly1 $project >> LOGs/log−cvsanaly−
      $project.txt`;

while ($num−−){
sleep(15);
}

        print "\t\tExecuting␣Bicho...\n";
        `bicho −−db−user−out=user −−db−password−out=password −−db−database−out
      =bicho1 −b github −u \"https://api.github.com/repos/$project_bicho/issues\"␣−−
      backend−user=user␣−−backend−password=password␣−−debug`;

␣␣␣␣`rm␣−rf␣$project`;
}
```

A.2 Tool-chain: Quantification of VC logs and BT Data of 1 OSS Projects

```

#!/usr/bin/perl −w

# DBI is the standard database interface for Perl
# DBD is the Perl module that we use to connect to the <a href="http://mysql.com/" />
      MySQL</a> database
use DBI;
use DBD::mysql;
use Set::Scalar;

use warnings;

@output_cvs = ();
@output_bicho = ();

# ------------------------------------------------------------
# open the accessDB file to retrieve the database name, host name, user name and password

# || die "Billy Can’t access your login credentials";

# -----FROM Bicho ------------------------------------------------------------
$database =’bicho’;
 $host = ’127.0.0.1’;
 $userid =’user’;
 $passwd =’pw’;


# invoke the ConnectToMySQL sub-routine to make the database connection
$connection = ConnectToMySql($database);

# set the value of your SQL query
$query = "select␣RIGHT(web_link,␣locate(’/’,reverse(web_link))-1)␣from␣issues_ext_github,␣issues␣where␣issues.id␣=␣issues_ext_github.id␣and␣issues.
```

```
        tracker_id␣=␣1";

# prepare your statement for connecting to the database
$statement = $connection->prepare($query);

# execute your SQL statement
$statement->execute();

# retrieve the values returned from executing your SQL statement
while (@data = $statement->fetchrow_array()) {

$data[0] =~ s/\s+//;

push (@output_bicho, $data[0]);



}

$database ='cvsanaly';
 $host = '127.0.0.1';
 $userid ='user';
 $passwd ='pw';



# invoke the ConnectToMySQL sub-routine to make the database connection
$connection = ConnectToMySql($database);

# set the value of your SQL query
```

```perl
$query = "select␣␣message␣from␣scmlog␣where␣repository_id=␣1␣and␣message␣NOT␣like␣'%Merge␣pull␣request%'␣and␣message␣like␣'%#%'␣";

# prepare your statement for connecting to the database
$statement = $connection->prepare($query);

# execute your SQL statement
$statement->execute();

# retrieve the values returned from executing your SQL statement
while (@data = $statement->fetchrow_array()) {

@tokens = split(/\s/, $data[0]);
        for($j=0; $j<=$#tokens; $j++){
                if ($tokens[$j] =~ /#\d+/){
                        $tokens[$j] =~ s/(\.|\,|\;|\:)//;
                        $tokens[$j] =~ s/.*#//;
                        push (@output_cvs, $tokens[$j]);
                }
        }

}
# CREATE SETS, USE SETS
$s1 = Set::Scalar->new (@output_bicho);
$s2 = Set::Scalar->new (@output_cvs);

# OPERATIONS ON SETS

$only_in_bicho  = $s1 - $s2;      # only in bicho
```

```
$in_cvs  = $s2;                    #  in cvsanaly
$only_in_cvs  = $s2−$s1;           # only in cvsanaly
$in_bicho = $s1;                   # in bicho
$common  = $s1 * $s2;              # common
$total  = $s1 + $s2;               # union




print $in_bicho−>size."\t".$in_cvs−>size."\t".$common−>size."\t".$only_in_bicho−>size
    ."\t".$only_in_cvs−>size."\t".$total−>size."\t"."\n";



# exit the script
exit;

#−−− start sub−routine
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
sub ConnectToMySql {
#
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−


my ($db) = @_;

# assign the values to your connection variable
my $connectionInfo="dbi:mysql:$db;$host";

# make connection to database
my $l_connection = DBI->connect($connectionInfo,$userid,$passwd);

# the value of this connection is returned by the sub-routine
return $l_connection;

}

#--- end sub-routine
    ----------------------------------------------------
```

A.3 Tool-chain: Quantification of VC logs and BT Data of 344 OSS Projects

```
#!/usr/bin/perl -w

# DBI is the standard database interface for Perl
# DBD is the Perl module that we use to connect to the <a href="http://mysql.com/" />
    MySQL</a> database
use DBI;
use DBD::mysql;
use Set::Scalar;

use warnings;

for ($i=1; $i<=344; $i++){
@output_cvs = ();
```

```
@output_bicho = ();

#
    -----------------------------------------------------------------

# open the accessDB file to retrieve the database name, host name, user name and
    password

# || die "Billy Can't access your login credentials";

# -----FROM Bicho
    -----------------------------------------------------------------

$database ='bicho';
 $host = '127.0.0.1';
 $userid ='userName';
 $passwd ='password';


# invoke the ConnectToMySQL sub-routine to make the database connection
$connection = ConnectToMySql($database);

# set the value of your SQL query
$query = "select RIGHT(web_link, locate('/',reverse(web_link))-1) from
    issues_ext_github, issues where issues.id = issues_ext_github.id and issues.
    tracker_id = ?";

# prepare your statement for connecting to the database
$statement = $connection->prepare($query);
```

```

# execute your SQL statement
$statement->execute($i);

# retrieve the values returned from executing your SQL statement
while (@data = $statement->fetchrow_array()) {

$data[0] =~ s/\s+//;

push (@output_bicho, $data[0]);


}

$database ='cvsanaly';
 $host = '127.0.0.1';
 $userid ='userName';
 $passwd ='passWord';


# invoke the ConnectToMySQL sub-routine to make the database connection
$connection = ConnectToMySql($database);

# set the value of your SQL query
$query = "select  message from scmlog where repository_id= ? and message NOT like 
      '%Merge pull request%' and message like '%#%' ";

# prepare your statement for connecting to the database
$statement = $connection->prepare($query);
```

```

# execute your SQL statement
$statement->execute($i);

# retrieve the values returned from executing your SQL statement
while (@data = $statement->fetchrow_array()) {

@tokens = split(/\s/, $data[0]);
        for($j=0; $j<=$#tokens; $j++){
                if ($tokens[$j] =~ /#\d+/){
                        $tokens[$j] =~ s/(\.|\,|\;|\:)//;
                        $tokens[$j] =~ s/.*#//;
                        push (@output_cvs, $tokens[$j]);
                }
        }

}
# CREATE SETS, USE SETS
$s1 = Set::Scalar->new (@output_bicho);
$s2 = Set::Scalar->new (@output_cvs);

# OPERATIONS ON SETS

$only_in_bicho  = $s1 - $s2;     # only in bicho
$in_cvs  = $s2;                   #  in cvsanaly
$only_in_cvs  = $s2-$s1;          # only in cvsanaly
$in_bicho = $s1;                  # in bicho
$common  = $s1 * $s2;             # common
$total  = $s1 + $s2;              # union
```

```
print $in_bicho->size."\t".$in_cvs->size."\t".$common->size."\t".$only_in_bicho->size
    ."\t".$only_in_cvs->size."\t".$total->size."\t"."\n";

}

# exit the script
exit;

#--- start sub-routine
    ---------------------------------------------
sub ConnectToMySql {
#
    ----------------------------------------------------


my ($db) = @_;

# assign the values to your connection variable
my $connectionInfo="dbi:mysql:$db;$host";

# make connection to database
my $l_connection = DBI->connect($connectionInfo,$userid,$passwd);

# the value of this connection is returned by the sub-routine
return $l_connection;
```

```

}

#--- end sub-routine
    ----------------------------------------------------------
```

A.4 Tool-chain: Re-engineering CVSAnalY and Bicho and integrate extra tables (SCMlogcvsanaly table and Issuesbicho table in their respective databases) and Synchronisation of VC logs and BT data of 344 OSS Projects

```
#!/usr/bin/perl -w
#use strict;
use v5.10; # for say() function
# DBI is the standard database interface for Perl
# DBD is the Perl module that we use to connect to the <a href="http://mysql.com/" />
    MySQL</a> database
use DBI;
use DBD::mysql;
use Set::Scalar;

use warnings;

for ($i=1; $i<=344; $i++){
@output_cvs = ();
@output_bicho = ();

```

```
# --------------------------------------------------------------------------------
# open the accessDB file to retrieve the database name, host name, user name and password

# || die "Billy Can't access your login credentials";

# -----FROM Bicho -------------------------------------------------------------

$database ='bicho2';
 $host = '127.0.0.1';
 $userid ='UserName';
 $passwd ='Password';


#say "IssuesBicho Table created successfully!";
# invoke the ConnectToMySQL sub-routine to make the database connection
$connection = ConnectToMySql($database);
#say "Creating IssuesBicho Table if NOT exists ";


$query= "CREATE TABLE IF NOT EXISTS issuesbicho (
  tracker_id int(11) DEFAULT NULL,
  submitted_by int(11) DEFAULT NULL,
  assigned_to int(11) DEFAULT NULL,
  submitted_on datetime DEFAULT NULL,
  issue mediumtext,
␣␣summary␣longtext)";
$statement = $connection−>prepare($query);
$statement−>execute();

# set the value of your SQL query
$query = "select␣RIGHT(web_link,␣locate('/',reverse(web_link))−1),tracker_id,␣
      submitted_by,␣assigned_to,␣submitted_on,issue,␣summary␣from␣issues_ext_github,␣
      issues␣where␣issues.id␣=␣issues_ext_github.id␣and␣issues.tracker_id␣=?";

# prepare your statement for connecting to the database
$statement = $connection−>prepare($query);

# execute your SQL statement
$statement−>execute($i);

# retrieve the values returned from executing your SQL statement
while (@data = $statement−>fetchrow_array()) {

$data[0] =~ s/\s+//;

push (@output_bicho, $data[0]);


}

$database ='cvsanaly2';
 $host = '127.0.0.1';
 $userid ='UserName';
 $passwd ='Pssowrd';
```

```
# invoke the ConnectToMySQL sub-routine to make the database connection
$connection = ConnectToMySql($database);

$query = "CREATE TABLE IF NOT EXISTS scmlogcvsanaly (
  weblink_bugid varchar(255),
  repository_id int(11),
  author_id int(10),
  committer_id int(10),
  date datetime,
  rev varchar(255),
  message longtext)";

$statement = $connection->prepare($query);

$statement->execute();
say "Creating SCMlogCVSAnalY Table if NOT exists ";

say "SCMlogCVSAnalY Table created successfully!";

# set the value of your SQL query
$query = " select  message from scmlog where repository_id= ? and message NOT like
     '%Merge pull request%' and message like '%#%'  ";

# prepare your statement for connecting to the database
$statement = $connection->prepare($query);

# execute your SQL statement
```

```
$statement->execute($i);

# retrieve the values returned from executing your SQL statement
while (@data = $statement->fetchrow_array()) {

@tokens = split(/\s/, $data[0]);
  for($j=0; $j<=$#tokens; $j++){
    if ($tokens[$j] =~ /#\d+/){
      $tokens[$j] =~ s/(\.|\,|\;|\:)//;
      $tokens[$j] =~ s/.*#//;
      push (@output_cvs, $tokens[$j]);
    }
  }

}
# CREATE SETS, USE SETS
$s1 = Set::Scalar->new (@output_bicho);
$s2 = Set::Scalar->new (@output_cvs);

foreach my $e ($s1->elements){
}

foreach my $e ($s2->elements){
}

# OPERATIONS ON SETS

$only_in_bicho  = $s1 - $s2;      # only in bicho
$in_cvs  = $s2;         #  in cvsanaly
```

```
$only_in_cvs  = $s2-$s1;          # only in cvsanaly
$in_bicho = $s1;      # in bicho
$common  = $s1 * $s2;       # common
$total  = $s1 + $s2;        # union


$conn1 = ConnectToMySql('bicho2');

foreach my $e ($only_in_cvs->elements){
}

foreach my $mida ($only_in_cvs->elements){


my $sql = "SELECT  distinct repository_id,author_id,committer_id,date,rev, message  
      from scmlog where repository_id= ? and message NOT like '%Merge pull request%'
      and message like ?";

  # prepare your statement for connecting to the database
$stat = $connection->prepare($sql);

# execute your SQL statement
$stat->execute($i, '%#'.$mida.'%');

# retrieve the values returned from executing your SQL statement
while (@data = $stat->fetchrow_array()) {

  my ($repid,$autid,$comid,$date,$revid,$messagelog) = @data;


  $query = "INSERT␣␣INTO␣␣issuesbicho␣(tracker_id,␣submitted_by,␣assigned_to,␣submitted_on,␣issue,␣␣summary)
␣␣␣␣␣␣values␣(?,␣?,␣?,␣?,?,?)␣";

  $statement = $conn1−>prepare($query);

  $statement−>execute($i, $autid, $comid, $date,$revid,$messagelog);

  }

}

foreach my $midb ($only_in_bicho−>elements){


  my $sql = "select␣␣RIGHT(web_link,␣locate('/',reverse(web_link))−1),tracker_id,␣submitted_by,␣assigned_to,␣submitted_on,issue,␣summary␣from␣issues_ext_github,␣issues␣where␣issues.id␣=␣issues_ext_github.id␣and␣issues.tracker_id␣=?␣and␣RIGHT(web_link,␣locate('/',reverse(web_link))−1)␣=␣?";

  $statement = $conn1−>prepare($sql);

  $statement−>execute($i, $midb);

  while (@data = $statement−>fetchrow_array()) {


    my ($web_link, $trackida, $submittedbya, $assignedtoa, $submittedona,$issuea,
     $summarya) = @data;

    $query = "INSERT␣␣INTO␣scmlogcvsanaly␣(weblink_bugid,␣repository_id,␣author_id,
     ␣committer_id,␣date,␣rev,␣message)
␣␣␣␣␣␣␣␣values␣(?,␣?,␣?,␣?,␣?,?,?)␣";
    $stat = $connection->prepare($query);

    $stat->execute($web_link, $trackida, $submittedbya, $assignedtoa, $submittedona,
     $issuea,$summarya);


  }
}

print $in_bicho->size."\t".$in_cvs->size."\t".$common->size."\t".$only_in_bicho->size
     ."\t".$only_in_cvs->size."\t".$total->size."\n";#"\t" .$Bicho->size."\t".$CVSAnalY
     ->size."\n";
}

# exit the script
exit;

#--- start sub-routine
     ---------------------------------------------------
sub ConnectToMySql {
#
     -----------------------------------------------------------
```

```

my ($db) = @_;

# assign the values to your connection variable
my $connectionInfo="dbi:mysql:$db;$host";

# make connection to database
my $l_connection = DBI->connect($connectionInfo,$userid,$passwd);

# the value of this connection is returned by the sub-routine
return $l_connection;

}

#--- end sub-routine
    ----------------------------------------------------
```

A.5 Tool-chain: Evaluating Bicho and CVSAnalY delta

```

#!/usr/bin/perl -w

# DBI is the standard database interface for Perl
# DBD is the Perl module that we use to connect to the <a href="http://mysql.com/" />
    MySQL</a> database
use DBI;
use DBD::mysql;
use Set::Scalar;

use warnings;

for ($i=1; $i<=344; $i++){
@output_cvs = ();
@output_bicho = ();

# −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

# open the accessDB file to retrieve the database name, host name, user name and
    password

# || die "Billy Can’t access your login credentials";

# −−−−−FROM Bicho
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

$database =’bicho2’;
 $host = ’127.0.0.1’;
 $userid =’Username’;
 $passwd =’Password’;


# invoke the ConnectToMySQL sub−routine to make the database connection
$connection = ConnectToMySql($database);

# set the value of your SQL query
```

```
$query1 = "select␣count(*)␣from␣issuesbicho␣␣where␣tracker_id=?";

# prepare your statement for connecting to the database
$statement = $connection->prepare($query1);

# execute your SQL statement
$statement->execute($i);

while (@data = $statement->fetchrow_array()) {


push (@output_bicho, $data[0]);
# retrieve the values returned from executing your SQL statement
}




$database ='cvsanaly2';
 $host = '127.0.0.1';
 $userid ='UserName';
 $passwd ='Password';


# invoke the ConnectToMySQL sub-routine to make the database connection
$connection = ConnectToMySql($database);

# set the value of your SQL query
$query2 = "select␣count(*)␣␣from␣scmlogcvsanaly␣␣where␣repository_id=?";
```

```

# prepare your statement for connecting to the database
$statement = $connection->prepare($query2);

# execute your SQL statement
$statement->execute($i);
while (@data = $statement->fetchrow_array()) {


push (@output_cvs, $data[0]);

$s1 = Set::Scalar->new (@output_bicho);
$s2 = Set::Scalar->new (@output_cvs);

$in_cvs  = $s2;
$in_bicho = $s1;                    # in bicho


print $in_bicho."\t".$in_cvs."\t"."\n";
}

}

# exit the script
exit;

#--- start sub-routine
    --------------------------------------------------
sub ConnectToMySql {
```

```
#
    ------------------------------------------------------------------

my ($db) = @_;

# assign the values to your connection variable
my $connectionInfo="dbi:mysql:$db;$host";

# make connection to database
my $l_connection = DBI->connect($connectionInfo,$userid,$passwd);

# the value of this connection is returned by the sub-routine
return $l_connection;

}

#--- end sub-routine
    ------------------------------------------------------------
```

A.6 Finding discrepancies: Results - 344 OSS Projects

Table 1: SZZ Algorithm: # Symbol - 344 OSS Projects

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
1	57	6	6	51	0	57	0.10526315789474
2	449	19	19	430	0	449	0.04231625835189
3	790	30	30	760	0	790	0.0379746835443
4	213	15	14	199	1	214	0.06542056074766
5	6	0	0	6	0	6	0
6	101	21	20	81	1	102	0.19607843137255
7	18	0	0	18	0	18	0
8	1459	218	202	1257	16	1475	0.13694915254237
9	34	2	2	32	0	34	0.05882352941176
10	2	1	0	2	1	3	0
11	18	3	2	16	1	19	0.10526315789474
12	29	1	1	28	0	29	0.03448275862069
13	14	541	1	13	540	554	0.00180505415162
14	2257	554	544	1713	10	2267	0.2399647110719
15	195	22	19	176	3	198	0.0959595959596
16	494	56	54	440	2	496	0.10887096774194
17	0	13	0	0	13	13	0
18	1	0	0	1	0	1	0
19	792	31	5	787	26	818	0.00611246943765
20	33	1	1	32	0	33	0.03030303030303
21	321	6	6	315	0	321	0.01869158878505
22	40	1	1	39	0	40	0.025
23	1	0	0	1	0	1	0
24	6	0	0	6	0	6	0
25	11	3	0	11	3	14	0
26	166	107	3	163	104	270	0.01111111111111
27	121	17	17	104	0	121	0.1404958677686
28	325	68	68	257	0	325	0.20923076923077
29	3	0	0	3	0	3	0
30	7	6	0	7	6	13	0
31	232	192	185	47	7	239	0.77405857740586
32	8	0	0	8	0	8	0
33	21	0	0	21	0	21	0
34	139	2	0	139	2	141	0
35	364	28	28	336	0	364	0.07692307692308
36	27	1	1	26	0	27	0.03703703703704
37	47	6	6	41	0	47	0.12765957446809
38	0	0	0	0	0	0	0
39	945	122	105	840	17	962	0.10914760914761
40	27	2	2	25	0	27	0.07407407407407
41	236	35	4	232	31	267	0.01498127340824
42	52	202	0	52	202	254	0
43	421	94	93	328	1	422	0.22037914691943
44	24	3	3	21	0	24	0.125
45	0	0	0	0	0	0	0
46	107	1	1	106	0	107	0.00934579439252
47	26	37	0	26	37	63	0
48	68	5	5	63	0	68	0.07352941176471
49	55	9	8	47	1	56	0.14285714285714
50	51	0	0	51	0	51	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
51	582	134	123	459	11	593	0.20741989881956
52	141	30	30	111	0	141	0.21276595744681
53	171	2	2	169	0	171	0.01169590643275
54	0	1	0	0	1	1	0
55	116	34	28	88	6	122	0.22950819672131
56	0	2	0	0	2	2	0
57	0	0	0	0	0	0	0
58	237	12	11	226	1	238	0.04621848739496
59	7	1048	1	6	1047	1054	0.00094876660342
60	2	0	0	2	0	2	0
61	110	14	11	99	3	113	0.09734513274336
62	7	571	1	6	570	577	0.00173310225303
63	134	21	10	124	11	145	0.06896551724138
64	82	61	61	21	0	82	0.74390243902439
65	0	196	0	0	196	196	0
66	8	787	2	6	785	793	0.00252206809584
67	0	348	0	0	348	348	0
68	716	101	95	621	6	722	0.13157894736842
69	12	4	1	11	3	15	0.06666666666667
70	1387	204	195	1192	9	1396	0.13968481375358
71	2	0	0	2	0	2	0
72	1	4	0	1	4	5	0
73	2	37	0	2	37	39	0
74	590	10	9	581	1	591	0.01522842639594
75	1	67	0	1	67	68	0
76	78	563	2	76	561	639	0.00312989045383
77	0	46	0	0	46	46	0
78	667	282	260	407	22	689	0.37735849056604
79	0	11	0	0	11	11	0
80	112	0	0	112	0	112	0
81	0	2	0	0	2	2	0
82	48	163	7	41	156	204	0.0343137254902
83	826	0	0	826	0	826	0
84	0	312	0	0	312	312	0
85	6	0	0	6	0	6	0
86	29	63	13	16	50	79	0.16455696202532
87	188	73	0	188	73	261	0
88	17	11	3	14	8	25	0.12
89	0	0	0	0	0	0	0
90	0	156	0	0	156	156	0
91	5	15	0	5	15	20	0
92	867	0	0	867	0	867	0
93	0	554	0	0	554	554	0
94	3610	0	0	3610	0	3610	0
95	1	2077	1	0	2076	2077	0.00048146364949
96	40	2	0	40	2	42	0
97	9	2	0	9	2	11	0
98	1	16	1	0	15	16	0.0625
99	0	67	0	0	67	67	0
100	49	14	10	39	4	53	0.18867924528302

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
101	3	0	0	3	0	3	0
102	83	2	1	82	1	84	0.01190476190476
103	0	0	0	0	0	0	0
104	32	1	0	32	1	33	0
105	0	3	0	0	3	3	0
106	38	0	0	38	0	38	0
107	23	222	6	17	216	239	0.02510460251046
108	5	0	0	5	0	5	0
109	7	67	0	7	67	74	0
110	0	23	0	0	23	23	0
111	56	153	4	52	149	205	0.01951219512195
112	70	0	0	70	0	70	0
113	468	102	96	372	6	474	0.20253164556962
114	7	36	5	2	31	38	0.13157894736842
115	435	27	0	435	27	462	0
116	2	17	0	2	17	19	0
117	1	0	0	1	0	1	0
118	38	25	22	16	3	41	0.53658536585366
119	9	0	0	9	0	9	0
120	43	0	0	43	0	43	0
121	0	42	0	0	42	42	0
122	131	20	11	120	9	140	0.07857142857143
123	233	7	2	231	5	238	0.00840336134454
124	105	22	0	105	22	127	0
125	0	0	0	0	0	0	0
126	541	77	2	539	75	616	0.00324675324675
127	2	0	0	2	0	2	0
128	2	19	1	1	18	20	0.05
129	43	9	1	42	8	51	0.01960784313725
130	0	0	0	0	0	0	0
131	160	227	24	136	203	363	0.06611570247934
132	0	175	0	0	175	175	0
133	6	747	2	4	745	751	0.00266311584554
134	1941	538	389	1552	149	2090	0.18612440191388
135	3323	0	0	3323	0	3323	0
136	0	17	0	0	17	17	0
137	3050	6	6	3044	0	3050	0.00196721311475
138	773	382	48	725	334	1107	0.04336043360434
139	245	70	9	236	61	306	0.02941176470588
140	481	116	45	436	71	552	0.08152173913043
141	3009	83	79	2930	4	3013	0.0262197145702
142	779	14	4	775	10	789	0.00506970849176
143	1704	37	33	1671	4	1708	0.01932084309133
144	1265	12	12	1253	0	1265	0.00948616600791
145	1850	274	90	1760	184	2034	0.04424778761062
146	1174	41	23	1151	18	1192	0.01929530201342
147	16	0	0	16	0	16	0
148	794	123	93	701	30	824	0.1128640776699
149	287	0	0	287	0	287	0
150	23	71	6	17	65	88	0.06818181818182

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
151	1159	0	0	1159	0	1159	0
152	78	38	13	65	25	103	0.12621359223301
153	1033	353	196	837	157	1190	0.16470588235294
154	10	183	0	10	183	193	0
155	583	61	60	523	1	584	0.1027397260274
156	2867	535	456	2411	79	2946	0.15478615071283
157	572	2969	258	314	2711	3283	0.07858665854401
158	483	43	33	450	10	493	0.06693711967546
159	3457	0	0	3457	0	3457	0
160	9127	13	13	9114	0	9127	0.00142434534896
161	945	95	93	852	2	947	0.09820485744456
162	14	123	2	12	121	135	0.01481481481481
163	133	91	22	111	69	202	0.10891089108911
164	345	5	1	344	4	349	0.00286532951289
165	650	9	7	643	2	652	0.01073619631902
166	168	21	20	148	1	169	0.11834319526627
167	101	40	36	65	4	105	0.34285714285714
168	111	30	2	109	28	139	0.01438848920863
169	124	9	7	117	2	126	0.05555555555556
170	470	403	39	431	364	834	0.04676258992806
171	172	54	41	131	13	185	0.22162162162162
172	0	109	0	0	109	109	0
173	69	329	19	50	310	379	0.05013192612137
174	622	4	4	618	0	622	0.0064308681672
175	277	1160	115	162	1045	1322	0.08698940998487
176	112	53	16	96	37	149	0.10738255033557
177	2534	8	2	2532	6	2540	0.0007874015748
178	276	6	6	270	0	276	0.02173913043478
179	440	82	69	371	13	453	0.1523178807947
180	81	4	4	77	0	81	0.04938271604938
181	560	14	13	547	1	561	0.02317290552585
182	34	121	1	33	120	154	0.00649350649351
183	0	12	0	0	12	12	0
184	1533	29	28	1505	1	1534	0.01825293350717
185	81	429	1	80	428	509	0.00196463654224
186	596	74	71	525	3	599	0.11853088480801
187	561	32	26	535	6	567	0.04585537918871
188	479	471	82	397	389	868	0.09447004608295
189	451	24	23	428	1	452	0.05088495575221
190	1467	1301	464	1003	837	2304	0.20138888888889
191	346	118	73	273	45	391	0.18670076726343
192	1798	5	0	1798	5	1803	0
193	608	11	9	599	2	610	0.01475409836066
194	6	560	1	5	559	565	0.00176991150442
195	203	4	2	201	2	205	0.00975609756098
196	1114	8	7	1107	1	1115	0.00627802690583
197	104	7	1	103	6	110	0.00909090909091
198	197	1646	5	192	1641	1838	0.00272034820457
199	511	78	16	495	62	573	0.02792321116928
200	1978	38	38	1940	0	1978	0.01921132457027
201	572	31	29	543	2	574	0.05052264808362
202	416	0	0	416	0	416	0
203	240	2	2	238	0	240	0.00833333333333
204	29	2035	0	29	2035	2064	0
205	71	1	0	71	1	72	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
206	3806	2095	826	2980	1269	5075	0.16275862068966
207	109	5	0	109	5	114	0
208	4164	80	13	4151	67	4231	0.00307255967856
209	230	3216	9	221	3207	3437	0.00261856270003
210	461	3	3	458	0	461	0.00650759219089
211	64	0	0	64	0	64	0
212	0	52	0	0	52	52	0
213	0	57	0	0	57	57	0
214	476	1	1	475	0	476	0.00210084033613
215	1694	382	327	1367	55	1749	0.18696397941681
216	19	15	3	16	12	31	0.09677419354839
217	2142	1	1	2141	0	2142	0.00046685340803
218	314	88	83	231	5	319	0.26018808777429
219	252	64	45	207	19	271	0.16605166051661
220	326	89	41	285	48	374	0.1096256684492
221	357	732	48	309	684	1041	0.04610951008646
222	448	4	4	444	0	448	0.00892857142857
223	12467	14	13	12454	1	12468	0.00104266923324
224	0	12	0	0	12	12	0
225	125	1	1	124	0	125	0.008
226	83	140	36	47	104	187	0.19251336898396
227	111	48	3	108	45	156	0.01923076923077
228	1270	15	13	1257	2	1272	0.01022012578616
229	330	16	16	314	0	330	0.04848484848485
230	163	42	3	160	39	202	0.01485148514851
231	71	0	0	71	0	71	0
232	452	52	52	400	0	452	0.11504424778761
233	189	20	17	172	3	192	0.08854166666667
234	208	1	1	207	0	208	0.00480769230769
235	270	1	1	269	0	270	0.0037037037037
236	51	115	2	49	113	164	0.01219512195122
237	119	0	0	119	0	119	0
238	790	35	35	755	0	790	0.04430379746835
239	1	18	0	1	18	19	0
240	510	0	0	510	0	510	0
241	1092	2	1	1091	1	1093	0.00091491308326
242	403	0	0	403	0	403	0
243	209	125	62	147	63	272	0.22794117647059
244	580	26	24	556	2	582	0.04123711340206
245	339	176	125	214	51	390	0.32051282051282
246	405	2	1	404	1	406	0.00246305418719
247	51	0	0	51	0	51	0
248	62	156	13	49	143	205	0.06341463414634
249	636	16	15	621	1	637	0.02354788069074
250	313	15	14	299	1	314	0.04458598726115
251	388	5	4	384	1	389	0.01028277634961
252	74	8	2	72	6	80	0.025
253	43	286	29	14	257	300	0.09666666666667
254	232	432	105	127	327	559	0.18783542039356
255	1752	2	1	1751	1	1753	0.00057045065602
256	38	1	1	37	0	38	0.02631578947368
257	232	13	13	219	0	232	0.05603448275862
258	215	16	16	199	0	215	0.07441860465116
259	119	10	10	109	0	119	0.08403361344538
260	102	11	5	97	6	108	0.0462962962963
261	154	248	122	32	126	280	0.43571428571429

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
262	419	111	65	354	46	465	0.13978494623656
263	0	11	0	0	11	11	0
264	10	22	1	9	21	31	0.03225806451613
265	225	42	4	221	38	263	0.01520912547529
266	41	23	3	38	20	61	0.04918032786885
267	686	0	0	686	0	686	0
268	170	1	1	169	0	170	0.00588235294118
269	88	0	0	88	0	88	0
270	5	0	0	5	0	5	0
271	0	0	0	0	0	0	0
272	15	0	0	15	0	15	0
273	12	0	0	12	0	12	0
274	0	58	0	0	58	58	0
275	749	51	51	698	0	749	0.06809078771696
276	328	0	0	328	0	328	0
277	68	16	6	62	10	78	0.07692307692308
278	2562	12	12	2550	0	2562	0.00468384074941
279	31	12	7	24	5	36	0.19444444444444
280	75	1	1	74	0	75	0.01333333333333
281	300	108	106	194	2	302	0.35099337748344
282	2458	304	39	2419	265	2723	0.01432243848696
283	1587	79	46	1541	33	1620	0.0283950617284
284	417	18	14	403	4	421	0.0332541567696
285	166	6	6	160	0	166	0.03614457831325
286	381	14	11	370	3	384	0.02864583333333
287	434	103	41	393	62	496	0.08266129032258
288	678	13	3	675	10	688	0.00436046511628
289	197	37	34	163	3	200	0.17
290	83	3	3	80	0	83	0.03614457831325
291	1446	163	156	1290	7	1453	0.10736407432897
292	655	27	23	632	4	659	0.03490136570561
293	47	2	2	45	0	47	0.04255319148936
294	1433	9908	176	1257	9732	11165	0.01576354679803
295	1590	277	269	1321	8	1598	0.16833541927409
296	1756	88	86	1670	2	1758	0.04891922639363
297	675	1375	39	636	1336	2011	0.01939333664843
298	198	14	12	186	2	200	0.06
299	154	14	14	140	0	154	0.09090909090909
300	864	93	90	774	3	867	0.1038062283737
301	1661	234	196	1465	38	1699	0.1153619776339
302	1211	168	164	1047	4	1215	0.13497942386831
303	549	55	53	496	2	551	0.0961887477314
304	439	71	63	376	8	447	0.14093959731544
305	64	21	5	59	16	80	0.0625
306	387	11	6	381	5	392	0.01530612244898
307	106	12	2	104	10	116	0.01724137931034
308	191	12	10	181	2	193	0.05181347150259
309	20	0	0	20	0	20	0
310	317	99	2	315	97	414	0.0048309178744
311	19	0	0	19	0	19	0
312	273	28	10	263	18	291	0.03436426116838
313	47	73	0	47	73	120	0
314	267	78	75	192	3	270	0.27777777777778

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
315	234	56	55	179	1	235	0.23404255319149
316	430	63	56	374	7	437	0.12814645308924
317	2338	321	284	2054	37	2375	0.11957894736842
318	296	4	4	292	0	296	0.01351351351351
319	4	0	0	4	0	4	0
320	114	0	0	114	0	114	0
321	152	15	15	137	0	152	0.09868421052632
322	175	43	42	133	1	176	0.23863636363636
323	321	5	5	316	0	321	0.01557632398754
324	47	8	2	45	6	53	0.0377358490566
325	463	79	75	388	4	467	0.16059957173448
326	144	8	8	136	0	144	0.05555555555556
327	324	20	19	305	1	325	0.05846153846154
328	12	49	8	4	41	53	0.15094339622642
329	142	2	1	141	1	143	0.00699300699301
330	25	12	1	24	11	36	0.02777777777778
331	164	3	2	162	1	165	0.01212121212121
332	266	59	57	209	2	268	0.21268656716418
333	4634	604	267	4367	337	4971	0.05371152685576
334	3374	3117	1252	2122	1865	5239	0.23897690398931
335	20	0	0	20	0	20	0
336	48	0	0	48	0	48	0
337	83	0	0	83	0	83	0
338	104	39	19	85	20	124	0.15322580645161
339	162	43	21	141	22	184	0.11413043478261
340	668	40	38	630	2	670	0.05671641791045
341	163	9	6	157	3	166	0.03614457831325
342	168	53	40	128	13	181	0.22099447513812
343	620	63	62	558	1	621	0.09983896940419
344	141	18	15	126	3	144	0.10416666666667

Table 2: SZZ Algorithm: Fixed - 344 OSS Projects

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
1	57	0	0	57	0	57	0
2	449	0	0	449	0	449	0
3	790	4	4	786	0	790	0.0050632911
4	213	0	0	213	0	213	0
5	6	0	0	6	0	6	0
6	101	1	1	100	0	101	0.0099009901
7	18	0	0	18	0	18	0
8	1459	37	34	1425	3	1462	0.023255814
9	34	1	1	33	0	34	0.0294117647
10	2	1	0	2	1	3	0
11	18	1	1	17	0	18	0.0555555556
12	29	0	0	29	0	29	0
13	14	119	0	14	119	133	0
14	2257	169	168	2089	1	2258	0.0744021258
15	195	5	4	191	1	196	0.0204081633
16	494	0	0	494	0	494	0
17	0	3	0	0	3	3	0
18	1	0	0	1	0	1	0
19	792	3	1	791	2	794	0.0012594458
20	33	0	0	33	0	33	0
21	321	0	0	321	0	321	0
22	40	0	0	40	0	40	0
23	1	0	0	1	0	1	0
24	6	0	0	6	0	6	0
25	11	0	0	11	0	11	0
26	166	20	0	166	20	186	0
27	121	0	0	121	0	121	0
28	325	5	5	320	0	325	0.0153846154
29	3	0	0	3	0	3	0
30	7	2	0	7	2	9	0
31	232	0	0	232	0	232	0
32	8	0	0	8	0	8	0
33	21	0	0	21	0	21	0
34	139	0	0	139	0	139	0
35	364	2	2	362	0	364	0.0054945055
36	27	0	0	27	0	27	0
37	47	0	0	47	0	47	0
38	0	0	0	0	0	0	0
39	945	3	3	942	0	945	0.0031746032
40	27	0	0	27	0	27	0
41	236	1	0	236	1	237	0
42	52	3	0	52	3	55	0
43	421	0	0	421	0	421	0
44	24	0	0	24	0	24	0
45	0	0	0	0	0	0	0
46	107	0	0	107	0	107	0
47	26	2	0	26	2	28	0
48	68	2	2	66	0	68	0.0294117647
49	55	1	1	54	0	55	0.0181818182
50	51	0	0	51	0	51	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
51	582	0	0	582	0	582	0
52	141	1	1	140	0	141	0.0070921986
53	171	0	0	171	0	171	0
54	0	0	0	0	0	0	0
55	116	23	21	95	2	118	0.1779661017
56	0	0	0	0	0	0	0
57	0	0	0	0	0	0	0
58	237	0	0	237	0	237	0
59	7	33	0	7	33	40	0
60	2	0	0	2	0	2	0
61	110	2	1	109	1	111	0.009009009
62	7	43	0	7	43	50	0
63	134	3	1	133	2	136	0.0073529412
64	82	0	0	82	0	82	0
65	0	18	0	0	18	18	0
66	8	52	0	8	52	60	0
67	0	8	0	0	8	8	0
68	716	1	1	715	0	716	0.001396648
69	12	0	0	12	0	12	0
70	1387	4	4	1383	0	1387	0.0028839221
71	2	0	0	2	0	2	0
72	1	0	0	1	0	1	0
73	2	1	0	2	1	3	0
74	590	0	0	590	0	590	0
75	1	2	0	1	2	3	0
76	78	155	0	78	155	233	0
77	0	6	0	0	6	6	0
78	667	3	3	664	0	667	0.0044977511
79	0	1	0	0	1	1	0
80	112	0	0	112	0	112	0
81	0	1	0	0	1	1	0
82	48	2	0	48	2	50	0
83	826	0	0	826	0	826	0
84	0	21	0	0	21	21	0
85	6	0	0	6	0	6	0
86	29	47	8	21	39	68	0.1176470588
87	188	0	0	188	0	188	0
88	17	0	0	17	0	17	0
89	0	0	0	0	0	0	0
90	0	12	0	0	12	12	0
91	5	0	0	5	0	5	0
92	867	0	0	867	0	867	0
93	0	84	0	0	84	84	0
94	3610	0	0	3610	0	3610	0
95	1	129	0	1	129	130	0
96	40	0	0	40	0	40	0
97	9	1	0	9	1	10	0
98	1	0	0	1	0	1	0
99	0	6	0	0	6	6	0
100	49	1	1	48	0	49	0.0204081633
101	3	0	0	3	0	3	0
102	83	0	0	83	0	83	0
103	0	0	0	0	0	0	0
104	32	0	0	32	0	32	0
105	0	0	0	0	0	0	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
106	38	0	0	38	0	38	0
107	23	9	0	23	9	32	0
108	5	0	0	5	0	5	0
109	7	8	0	7	8	15	0
110	0	0	0	0	0	0	0
111	56	13	1	55	12	68	0.0147058824
112	70	0	0	70	0	70	0
113	468	9	9	459	0	468	0.0192307692
114	7	1	0	7	1	8	0
115	435	0	0	435	0	435	0
116	2	0	0	2	0	2	0
117	1	0	0	1	0	1	0
118	38	18	15	23	3	41	0.3658536585
119	9	0	0	9	0	9	0
120	43	0	0	43	0	43	0
121	0	9	0	0	9	9	0
122	131	0	0	131	0	131	0
123	233	0	0	233	0	233	0
124	105	3	0	105	3	108	0
125	0	0	0	0	0	0	0
126	541	7	0	541	7	548	0
127	2	0	0	2	0	2	0
128	2	0	0	2	0	2	0
129	43	0	0	43	0	43	0
130	0	0	0	0	0	0	0
131	160	2	0	160	2	162	0
132	0	2	0	0	2	2	0
133	6	107	0	6	107	113	0
134	1941	60	56	1885	4	1945	0.0287917738
135	3323	0	0	3323	0	3323	0
136	0	1	0	0	1	1	0
137	3050	0	0	3050	0	3050	0
138	773	6	1	772	5	778	0.001285347
139	245	0	0	245	0	245	0
140	481	3	2	479	1	482	0.0041493776
141	3009	0	0	3009	0	3009	0
142	779	3	1	778	2	781	0.0012804097
143	1704	2	2	1702	0	1704	0.0011737089
144	1265	2	2	1263	0	1265	0.0015810277
145	1850	1	1	1849	0	1850	0.0005405405
146	1174	1	1	1173	0	1174	0.0008517888
147	16	0	0	16	0	16	0
148	794	3	3	791	0	794	0.0037783375
149	287	0	0	287	0	287	0
150	23	19	3	20	16	39	0.0769230769
151	1159	0	0	1159	0	1159	0
152	78	7	2	76	5	83	0.0240963855
153	1033	9	5	1028	4	1037	0.0048216008
154	10	1	0	10	1	11	0
155	583	16	16	567	0	583	0.0274442539
156	2867	16	11	2856	5	2872	0.0038300836
157	572	53	11	561	42	614	0.0179153094
158	483	0	0	483	0	483	0
159	3457	0	0	3457	0	3457	0
160	9127	0	0	9127	0	9127	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
161	945	29	29	916	0	945	0.0306878307
162	14	3	0	14	3	17	0
163	133	1	0	133	1	134	0
164	345	0	0	345	0	345	0
165	650	0	0	650	0	650	0
166	168	0	0	168	0	168	0
167	101	0	0	101	0	101	0
168	111	1	0	111	1	112	0
169	124	0	0	124	0	124	0
170	470	53	1	469	52	522	0.0019157088
171	172	0	0	172	0	172	0
172	0	0	0	0	0	0	0
173	69	75	16	53	59	128	0.125
174	622	0	0	622	0	622	0
175	277	35	3	274	32	309	0.0097087379
176	112	3	2	110	1	113	0.017699115
177	2534	1	0	2534	1	2535	0
178	276	0	0	276	0	276	0
179	440	5	4	436	1	441	0.0090702948
180	81	0	0	81	0	81	0
181	560	0	0	560	0	560	0
182	34	2	0	34	2	36	0
183	0	0	0	0	0	0	0
184	1533	2	2	1531	0	1533	0.0013046314
185	81	5	0	81	5	86	0
186	596	63	61	535	2	598	0.102006689
187	561	2	1	560	1	562	0.0017793594
188	479	8	1	478	7	486	0.0020576132
189	451	1	1	450	0	451	0.0022172949
190	1467	10	4	1463	6	1473	0.0027155465
191	346	0	0	346	0	346	0
192	1798	1	0	1798	1	1799	0
193	608	0	0	608	0	608	0
194	6	8	0	6	8	14	0
195	203	0	0	203	0	203	0
196	1114	1	1	1113	0	1114	0.0008976661
197	104	0	0	104	0	104	0
198	197	247	0	197	247	444	0
199	511	2	1	510	1	512	0.001953125
200	1978	1	1	1977	0	1978	0.0005055612
201	572	3	3	569	0	572	0.0052447552
202	416	0	0	416	0	416	0
203	240	0	0	240	0	240	0
204	29	103	0	29	103	132	0
205	71	1	0	71	1	72	0
206	3806	1057	449	3357	608	4414	0.1017217943
207	109	0	0	109	0	109	0
208	4164	16	1	4163	15	4179	0.0002392917
209	230	880	3	227	877	1107	0.0027100271
210	461	0	0	461	0	461	0
211	64	0	0	64	0	64	0
212	0	1	0	0	1	1	0
213	0	2	0	0	2	2	0
214	476	0	0	476	0	476	0
215	1694	8	8	1686	0	1694	0.0047225502

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
216	19	1	1	18	0	19	0.0526315789
217	2142	1	1	2141	0	2142	0.0004668534
218	314	2	1	313	1	315	0.0031746032
219	252	2	0	252	2	254	0
220	326	1	0	326	1	327	0
221	357	58	1	356	57	414	0.0024154589
222	448	0	0	448	0	448	0
223	12467	3	2	12465	1	12468	0.0001604107
224	0	0	0	0	0	0	0
225	125	0	0	125	0	125	0
226	83	5	1	82	4	87	0.0114942529
227	111	0	0	111	0	111	0
228	1270	0	0	1270	0	1270	0
229	330	0	0	330	0	330	0
230	163	22	2	161	20	183	0.0109289617
231	71	0	0	71	0	71	0
232	452	0	0	452	0	452	0
233	189	4	4	185	0	189	0.0211640212
234	208	0	0	208	0	208	0
235	270	0	0	270	0	270	0
236	51	2	0	51	2	53	0
237	119	0	0	119	0	119	0
238	790	0	0	790	0	790	0
239	1	1	0	1	1	2	0
240	510	0	0	510	0	510	0
241	1092	0	0	1092	0	1092	0
242	403	0	0	403	0	403	0
243	209	26	21	188	5	214	0.0981308411
244	580	4	4	576	0	580	0.0068965517
245	339	9	5	334	4	343	0.0145772595
246	405	0	0	405	0	405	0
247	51	0	0	51	0	51	0
248	62	9	1	61	8	70	0.0142857143
249	636	0	0	636	0	636	0
250	313	0	0	313	0	313	0
251	388	1	1	387	0	388	0.0025773196
252	74	0	0	74	0	74	0
253	43	32	1	42	31	74	0.0135135135
254	232	24	4	228	20	252	0.0158730159
255	1752	0	0	1752	0	1752	0
256	38	0	0	38	0	38	0
257	232	0	0	232	0	232	0
258	215	0	0	215	0	215	0
259	119	1	1	118	0	119	0.0084033613
260	102	0	0	102	0	102	0
261	154	5	4	150	1	155	0.0258064516
262	419	32	31	388	1	420	0.0738095238
263	0	0	0	0	0	0	0
264	10	1	0	10	1	11	0
265	225	4	1	224	3	228	0.0043859649
266	41	2	0	41	2	43	0
267	686	0	0	686	0	686	0
268	170	0	0	170	0	170	0
269	88	0	0	88	0	88	0
270	5	0	0	5	0	5	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
271	0	0	0	0	0	0	0
272	15	0	0	15	0	15	0
273	12	0	0	12	0	12	0
274	0	0	0	0	0	0	0
275	749	6	6	743	0	749	0.0080106809
276	328	0	0	328	0	328	0
277	68	1	0	68	1	69	0
278	2562	1	1	2561	0	2562	0.0003903201
279	31	0	0	31	0	31	0
280	75	0	0	75	0	75	0
281	300	0	0	300	0	300	0
282	2458	3	2	2456	1	2459	0.0008133388
283	1587	0	0	1587	0	1587	0
284	417	0	0	417	0	417	0
285	166	0	0	166	0	166	0
286	381	1	1	380	0	381	0.0026246719
287	434	4	3	431	1	435	0.0068965517
288	678	1	0	678	1	679	0
289	197	1	1	196	0	197	0.0050761421
290	83	1	1	82	0	83	0.0120481928
291	1446	2	2	1444	0	1446	0.0013831259
292	655	1	0	655	1	656	0
293	47	0	0	47	0	47	0
294	1433	1408	13	1420	1395	2828	0.0045968883
295	1590	2	1	1589	1	1591	0.0006285355
296	1756	17	17	1739	0	1756	0.0096810934
297	675	118	2	673	116	791	0.002528445
298	198	1	1	197	0	198	0.0050505051
299	154	2	2	152	0	154	0.012987013
300	864	15	15	849	0	864	0.0173611111
301	1661	4	4	1657	0	1661	0.0024081878
302	1211	23	23	1188	0	1211	0.0189925681
303	549	9	9	540	0	549	0.0163934426
304	439	8	8	431	0	439	0.0182232346
305	64	2	1	63	1	65	0.0153846154
306	387	0	0	387	0	387	0
307	106	11	2	104	9	115	0.0173913043
308	191	0	0	191	0	191	0
309	20	0	0	20	0	20	0
310	317	3	0	317	3	320	0
311	19	0	0	19	0	19	0
312	273	15	1	272	14	287	0.0034843206
313	47	5	0	47	5	52	0
314	267	2	2	265	0	267	0.0074906367
315	234	0	0	234	0	234	0
316	430	1	1	429	0	430	0.0023255814
317	2338	46	38	2300	8	2346	0.0161977835
318	296	0	0	296	0	296	0
319	4	0	0	4	0	4	0
320	114	0	0	114	0	114	0
321	152	8	8	144	0	152	0.0526315789
322	175	3	3	172	0	175	0.0171428571
323	321	1	1	320	0	321	0.0031152648
324	47	3	0	47	3	50	0
325	463	1	0	463	1	464	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
326	144	0	0	144	0	144	0
327	324	1	1	323	0	324	0.0030864198
328	12	13	3	9	10	22	0.1363636364
329	142	0	0	142	0	142	0
330	25	0	0	25	0	25	0
331	164	0	0	164	0	164	0
332	266	2	1	265	1	267	0.0037453184
333	4634	63	31	4603	32	4666	0.0066438063
334	3374	457	220	3154	237	3611	0.0609249515
335	20	0	0	20	0	20	0
336	48	0	0	48	0	48	0
337	83	0	0	83	0	83	0
338	104	6	3	101	3	107	0.0280373832
339	162	4	1	161	3	165	0.0060606061
340	668	1	1	667	0	668	0.001497006
341	163	1	1	162	0	163	0.0061349693
342	168	2	2	166	0	168	0.0119047619
343	620	12	12	608	0	620	0.0193548387
344	141	5	4	137	1	142	0.0281690141

Table 3: SZZ Algorithm: Bug - 344 OSS Projects

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
1	57	0	0	57	0	57	0
2	449	0	0	449	0	449	0
3	790	1	1	789	0	790	0.0012658228
4	213	0	0	213	0	213	0
5	6	0	0	6	0	6	0
6	101	1	0	101	1	102	0
7	18	0	0	18	0	18	0
8	1459	23	20	1439	3	1462	0.0136798906
9	34	0	0	34	0	34	0
10	2	1	0	2	1	3	0
11	18	1	1	17	0	18	0.0555555556
12	29	0	0	29	0	29	0
13	14	24	0	14	24	38	0
14	2257	48	47	2210	1	2258	0.0208148804
15	195	2	2	193	0	195	0.0102564103
16	494	0	0	494	0	494	0
17	0	2	0	0	2	2	0
18	1	0	0	1	0	1	0
19	792	2	0	792	2	794	0
20	33	0	0	33	0	33	0
21	321	2	2	319	0	321	0.0062305296
22	40	0	0	40	0	40	0
23	1	0	0	1	0	1	0
24	6	0	0	6	0	6	0
25	11	0	0	11	0	11	0
26	166	4	0	166	4	170	0
27	121	0	0	121	0	121	0
28	325	43	43	282	0	325	0.1323076923
29	3	0	0	3	0	3	0
30	7	2	0	7	2	9	0
31	232	6	6	226	0	232	0.025862069
32	8	0	0	8	0	8	0
33	21	0	0	21	0	21	0
34	139	0	0	139	0	139	0
35	364	0	0	364	0	364	0
36	27	0	0	27	0	27	0
37	47	0	0	47	0	47	0
38	0	0	0	0	0	0	0
39	945	4	4	941	0	945	0.0042328042
40	27	0	0	27	0	27	0
41	236	2	1	235	1	237	0.0042194093
42	52	8	0	52	8	60	0
43	421	4	4	417	0	421	0.0095011876
44	24	0	0	24	0	24	0
45	0	0	0	0	0	0	0
46	107	0	0	107	0	107	0
47	26	0	0	26	0	26	0
48	68	0	0	68	0	68	0
49	55	1	1	54	0	55	0.0181818182
50	51	0	0	51	0	51	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
51	582	7	5	577	2	584	0.0085616438
52	141	1	1	140	0	141	0.0070921986
53	171	0	0	171	0	171	0
54	0	0	0	0	0	0	0
55	116	1	0	116	1	117	0
56	0	0	0	0	0	0	0
57	0	0	0	0	0	0	0
58	237	2	2	235	0	237	0.0084388186
59	7	915	0	7	915	922	0
60	2	0	0	2	0	2	0
61	110	1	1	109	0	110	0.0090909091
62	7	562	0	7	562	569	0
63	134	0	0	134	0	134	0
64	82	0	0	82	0	82	0
65	0	68	0	0	68	68	0
66	8	780	2	6	778	786	0.0025445293
67	0	347	0	0	347	347	0
68	716	0	0	716	0	716	0
69	12	0	0	12	0	12	0
70	1387	8	8	1379	0	1387	0.0057678443
71	2	0	0	2	0	2	0
72	1	1	0	1	1	2	0
73	2	3	0	2	3	5	0
74	590	1	1	589	0	590	0.0016949153
75	1	67	0	1	67	68	0
76	78	7	0	78	7	85	0
77	0	46	0	0	46	46	0
78	667	5	4	663	1	668	0.005988024
79	0	2	0	0	2	2	0
80	112	0	0	112	0	112	0
81	0	0	0	0	0	0	0
82	48	3	0	48	3	51	0
83	826	0	0	826	0	826	0
84	0	312	0	0	312	312	0
85	6	0	0	6	0	6	0
86	29	4	3	26	1	30	0.1
87	188	13	0	188	13	201	0
88	17	0	0	17	0	17	0
89	0	0	0	0	0	0	0
90	0	156	0	0	156	156	0
91	5	2	0	5	2	7	0
92	867	0	0	867	0	867	0
93	0	17	0	0	17	17	0
94	3610	0	0	3610	0	3610	0
95	1	1981	1	0	1980	1981	0.0005047956
96	40	0	0	40	0	40	0
97	9	0	0	9	0	9	0
98	1	0	0	1	0	1	0
99	0	67	0	0	67	67	0
100	49	0	0	49	0	49	0
101	3	0	0	3	0	3	0
102	83	0	0	83	0	83	0
103	0	0	0	0	0	0	0
104	32	0	0	32	0	32	0
105	0	0	0	0	0	0	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
106	38	0	0	38	0	38	0
107	23	114	0	23	114	137	0
108	5	0	0	5	0	5	0
109	7	2	0	7	2	9	0
110	0	1	0	0	1	1	0
111	56	10	0	56	10	66	0
112	70	0	0	70	0	70	0
113	468	2	2	466	0	468	0.0042735043
114	7	31	0	7	31	38	0
115	435	3	0	435	3	438	0
116	2	4	0	2	4	6	0
117	1	0	0	1	0	1	0
118	38	0	0	38	0	38	0
119	9	0	0	9	0	9	0
120	43	0	0	43	0	43	0
121	0	11	0	0	11	11	0
122	131	2	2	129	0	131	0.0152671756
123	233	4	1	232	3	236	0.0042372881
124	105	2	0	105	2	107	0
125	0	0	0	0	0	0	0
126	541	73	1	540	72	613	0.0016313214
127	2	0	0	2	0	2	0
128	2	2	0	2	2	4	0
129	43	0	0	43	0	43	0
130	0	0	0	0	0	0	0
131	160	1	0	160	1	161	0
132	0	6	0	0	6	6	0
133	6	106	0	6	106	112	0
134	1941	45	31	1910	14	1955	0.0158567775
135	3323	0	0	3323	0	3323	0
136	0	0	0	0	0	0	0
137	3050	0	0	3050	0	3050	0
138	773	21	6	767	15	788	0.0076142132
139	245	2	2	243	0	245	0.0081632653
140	481	4	1	480	3	484	0.0020661157
141	3009	1	1	3008	0	3009	0.0003323363
142	779	0	0	779	0	779	0
143	1704	15	11	1693	4	1708	0.006440281
144	1265	4	4	1261	0	1265	0.0031620553
145	1850	4	0	1850	4	1854	0
146	1174	3	1	1173	2	1176	0.0008503401
147	16	0	0	16	0	16	0
148	794	2	2	792	0	794	0.0025188917
149	287	0	0	287	0	287	0
150	23	4	0	23	4	27	0
151	1159	0	0	1159	0	1159	0
152	78	6	2	76	4	82	0.0243902439
153	1033	7	5	1028	2	1035	0.0048309179
154	10	7	0	10	7	17	0
155	583	1	1	582	0	583	0.0017152659
156	2867	15	11	2856	4	2871	0.0038314176
157	572	129	20	552	109	681	0.0293685756
158	483	0	0	483	0	483	0
159	3457	0	0	3457	0	3457	0
160	9127	1	1	9126	0	9127	0.000109565

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
161	945	9	9	936	0	945	0.0095238095
162	14	4	1	13	3	17	0.0588235294
163	133	0	0	133	0	133	0
164	345	0	0	345	0	345	0
165	650	2	2	648	0	650	0.0030769231
166	168	0	0	168	0	168	0
167	101	3	3	98	0	101	0.0297029703
168	111	4	0	111	4	115	0
169	124	0	0	124	0	124	0
170	470	282	30	440	252	722	0.0415512465
171	172	0	0	172	0	172	0
172	0	9	0	0	9	9	0
173	69	6	0	69	6	75	0
174	622	0	0	622	0	622	0
175	277	107	6	271	101	378	0.0158730159
176	112	1	0	112	1	113	0
177	2534	4	0	2534	4	2538	0
178	276	0	0	276	0	276	0
179	440	1	1	439	0	440	0.0022727273
180	81	0	0	81	0	81	0
181	560	0	0	560	0	560	0
182	34	4	0	34	4	38	0
183	0	0	0	0	0	0	0
184	1533	5	5	1528	0	1533	0.0032615786
185	81	15	0	81	15	96	0
186	596	63	61	535	2	598	0.102006689
187	561	4	4	557	0	561	0.0071301248
188	479	6	1	478	5	484	0.0020661157
189	451	0	0	451	0	451	0
190	1467	23	5	1462	18	1485	0.0033670034
191	346	5	1	345	4	350	0.0028571429
192	1798	0	0	1798	0	1798	0
193	608	1	1	607	0	608	0.0016447368
194	6	14	0	6	14	20	0
195	203	0	0	203	0	203	0
196	1114	0	0	1114	0	1114	0
197	104	0	0	104	0	104	0
198	197	139	0	197	139	336	0
199	511	5	4	507	1	512	0.0078125
200	1978	1	1	1977	0	1978	0.0005055612
201	572	1	1	571	0	572	0.0017482517
202	416	0	0	416	0	416	0
203	240	0	0	240	0	240	0
204	29	47	0	29	47	76	0
205	71	0	0	71	0	71	0
206	3806	697	180	3626	517	4323	0.0416377516
207	109	0	0	109	0	109	0
208	4164	3	2	4162	1	4165	0.0004801921
209	230	263	0	230	263	493	0
210	461	0	0	461	0	461	0
211	64	0	0	64	0	64	0
212	0	2	0	0	2	2	0
213	0	3	0	0	3	3	0
214	476	0	0	476	0	476	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
215	1694	17	16	1678	1	1695	0.009439528
216	19	2	1	18	1	20	0.05
217	2142	0	0	2142	0	2142	0
218	314	4	4	310	0	314	0.0127388535
219	252	0	0	252	0	252	0
220	326	2	1	325	1	327	0.003058104
221	357	34	2	355	32	389	0.0051413882
222	448	0	0	448	0	448	0
223	12467	0	0	12467	0	12467	0
224	0	0	0	0	0	0	0
225	125	0	0	125	0	125	0
226	83	5	1	82	4	87	0.0114942529
227	111	0	0	111	0	111	0
228	1270	1	1	1269	0	1270	0.0007874016
229	330	0	0	330	0	330	0
230	163	33	1	162	32	195	0.0051282051
231	71	0	0	71	0	71	0
232	452	0	0	452	0	452	0
233	189	0	0	189	0	189	0
234	208	0	0	208	0	208	0
235	270	0	0	270	0	270	0
236	51	4	0	51	4	55	0
237	119	0	0	119	0	119	0
238	790	4	4	786	0	790	0.0050632911
239	1	2	0	1	2	3	0
240	510	0	0	510	0	510	0
241	1092	0	0	1092	0	1092	0
242	403	0	0	403	0	403	0
243	209	3	1	208	2	211	0.0047393365
244	580	1	1	579	0	580	0.0017241379
245	339	10	4	335	6	345	0.0115942029
246	405	0	0	405	0	405	0
247	51	0	0	51	0	51	0
248	62	8	0	62	8	70	0
249	636	0	0	636	0	636	0
250	313	0	0	313	0	313	0
251	388	0	0	388	0	388	0
252	74	1	0	74	1	75	0
253	43	33	1	42	32	75	0.0133333333
254	232	22	4	228	18	250	0.016
255	1752	0	0	1752	0	1752	0
256	38	0	0	38	0	38	0
257	232	2	2	230	0	232	0.0086206897
258	215	0	0	215	0	215	0
259	119	0	0	119	0	119	0
260	102	2	1	101	1	103	0.0097087379
261	154	8	5	149	3	157	0.0318471338
262	419	5	3	416	2	421	0.0071258907
263	0	0	0	0	0	0	0
264	10	0	0	10	0	10	0
265	225	9	1	224	8	233	0.0042918455
266	41	2	0	41	2	43	0
267	686	0	0	686	0	686	0
268	170	0	0	170	0	170	0
269	88	0	0	88	0	88	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
270	5	0	0	5	0	5	0
271	0	0	0	0	0	0	0
272	15	0	0	15	0	15	0
273	12	0	0	12	0	12	0
274	0	0	0	0	0	0	0
275	749	3	3	746	0	749	0.0040053405
276	328	0	0	328	0	328	0
277	68	0	0	68	0	68	0
278	2562	1	1	2561	0	2562	0.0003903201
279	31	1	1	30	0	31	0.0322580645
280	75	0	0	75	0	75	0
281	300	31	30	270	1	301	0.0996677741
282	2458	5	1	2457	4	2462	0.0004061738
283	1587	1	1	1586	0	1587	0.0006301197
284	417	0	0	417	0	417	0
285	166	0	0	166	0	166	0
286	381	0	0	381	0	381	0
287	434	8	7	427	1	435	0.016091954
288	678	3	0	678	3	681	0
289	197	0	0	197	0	197	0
290	83	0	0	83	0	83	0
291	1446	0	0	1446	0	1446	0
292	655	0	0	655	0	655	0
293	47	0	0	47	0	47	0
294	1433	5767	54	1379	5713	7146	0.0075566751
295	1590	12	12	1578	0	1590	0.0075471698
296	1756	3	3	1753	0	1756	0.0017084282
297	675	1186	30	645	1156	1831	0.0163844894
298	198	3	3	195	0	198	0.0151515152
299	154	0	0	154	0	154	0
300	864	7	7	857	0	864	0.0081018519
301	1661	3	3	1658	0	1661	0.0018061409
302	1211	1	1	1210	0	1211	0.0008257638
303	549	1	0	549	1	550	0
304	439	0	0	439	0	439	0
305	64	14	1	63	13	77	0.012987013
306	387	0	0	387	0	387	0
307	106	0	0	106	0	106	0
308	191	0	0	191	0	191	0
309	20	0	0	20	0	20	0
310	317	0	0	317	0	317	0
311	19	0	0	19	0	19	0
312	273	0	0	273	0	273	0
313	47	44	0	47	44	91	0
314	267	0	0	267	0	267	0
315	234	2	2	232	0	234	0.0085470085
316	430	2	1	429	1	431	0.0023201856
317	2338	9	9	2329	0	2338	0.003849444
318	296	3	3	293	0	296	0.0101351351
319	4	0	0	4	0	4	0
320	114	0	0	114	0	114	0
321	152	0	0	152	0	152	0
322	175	1	1	174	0	175	0.0057142857
323	321	0	0	321	0	321	0
324	47	0	0	47	0	47	0

S/N	All in Bicho	All in CVSAnalY	Intersection	Only in Bicho	only CSVAnalY	Union	Shared bug coverage
325	463	2	2	461	0	463	0.0043196544
326	144	1	1	143	0	144	0.0069444444
327	324	1	1	323	0	324	0.0030864198
328	12	4	0	12	4	16	0
329	142	1	0	142	1	143	0
330	25	6	0	25	6	31	0
331	164	0	0	164	0	164	0
332	266	4	3	263	1	267	0.0112359551
333	4634	154	79	4555	75	4709	0.0167763856
334	3374	375	94	3280	281	3655	0.0257181943
335	20	0	0	20	0	20	0
336	48	0	0	48	0	48	0
337	83	0	0	83	0	83	0
338	104	2	1	103	1	105	0.0095238095
339	162	2	2	160	0	162	0.012345679
340	668	5	5	663	0	668	0.0074850299
341	163	0	0	163	0	163	0
342	168	1	0	168	1	169	0
343	620	5	5	615	0	620	0.0080645161
344	141	2	2	139	0	141	0.0141843972

A.7 344 OSS Projects Precision and recall of the three main components of the SZZ algorithm

The table in 4 is the result of the precision and recall of each individual component of the SZZ Algorithm which was evaluated per project. In addition, the result of each component was used and compare against each component and obtained the p-value reported in chapter 4.6 that demonstrate the significant of each component which was summarise in the matrix table in 4.7 of chapter 4.

A.8 Bicho and CVSAnalY Delta-344 OSS Projects

The table 5 present the percentage of BT data and VC logs recovered and synchronised in the auxiliary tables of Bicho and CVSAnalY databases per project. The columns in table 1 such as **Only in Bicho** and **Only in CVSAnalY** was synchronised using # Symbol of the SZZ Algorithm for all the 344 OSS Projects in Bicho and CVSAnalY respective databases.

Table 4: 344 OSS Projects Precision and recall of the three main components of the SZZ algorithm

Precision							**Recall**					
S/N	hash	fix	hash	bug	fix	bug	hash	fix	hash	bug	fix	bug
1	0.895	0.000	0.895	0.000	0.000	0.000	0.531	0.000	0.531	0.000	0.000	0.000
2	0.958	0.000	0.958	0.000	0.000	0.000	0.511	0.000	0.511	0.000	0.000	0.000
3	0.962	0.000	0.962	0.000	0.000	0.000	0.510	0.000	0.510	0.000	0.000	0.000
4	0.934	0.000	0.934	0.000	0.000	0.000	0.518	0.000	0.518	0.000	0.000	0.000
5	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
6	0.802	0.000	0.802	0.000	0.000	0.000	0.570	0.000	0.570	0.000	0.000	0.000
7	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
8	0.862	0.002	0.862	-0.002	0.002	-0.002	0.544	0.002	0.544	0.002	0.002	0.002
9	0.941	0.000	0.941	0.000	0.000	0.000	0.516	0.000	0.516	0.000	0.000	0.000
10	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
11	0.889	0.000	0.889	0.000	0.000	0.000	0.533	0.000	0.533	0.000	0.000	0.000
12	0.966	0.000	0.966	0.000	0.000	0.000	0.509	0.000	0.509	0.000	0.000	0.000
13	0.929	0.000	0.929	0.000	0.000	0.000	0.520	0.000	0.520	0.000	0.000	0.000
14	0.759	0.000	0.759	0.000	0.000	0.000	0.594	0.000	0.594	0.000	0.000	0.000
15	0.903	0.005	0.903	0.000	0.005	0.000	0.529	0.005	0.529	0.000	0.005	0.000
16	0.891	0.000	0.891	0.000	0.000	0.000	0.533	0.000	0.533	0.000	0.000	0.000
17	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
18	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
19	0.994	0.003	0.994	0.000	0.003	0.000	0.502	0.003	0.502	0.000	0.003	0.000
20	0.97	0.000	0.97	0.000	0.000	0.000	0.508	0.000	0.508	0.000	0.000	0.000
21	0.981	0.000	0.981	0.000	0.000	0.000	0.505	0.000	0.505	0.000	0.000	0.000
22	0.975	0.000	0.975	0.000	0.000	0.000	0.506	0.000	0.506	0.000	0.000	0.000
23	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
24	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
25	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
26	0.982	0.000	0.982	0.000	0.000	0.000	0.505	0.000	0.505	0.000	0.000	0.000
27	0.86	0.000	0.86	0.000	0.000	0.000	0.545	0.000	0.545	0.000	0.000	0.000
28	0.791	0.000	0.791	0.000	0.000	0.000	0.576	0.000	0.576	0.000	0.000	0.000
29	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
30	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
31	0.203	0.000	0.203	0.000	0.000	0.000	-0.516	0.000	-0.516	0.000	0.000	0.000
32	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
33	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
34	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
35	0.923	0.000	0.923	0.000	0.000	0.000	0.522	0.000	0.522	0.000	0.000	0.000
36	0.963	0.000	0.963	0.000	0.000	0.000	0.510	0.000	0.510	0.000	0.000	0.000
37	0.872	0.000	0.872	0.000	0.000	0.000	0.539	0.000	0.539	0.000	0.000	0.000
38	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
39	0.889	0.000	0.889	0.000	0.000	0.000	0.533	0.000	0.533	0.000	0.000	0.000
40	0.926	0.000	0.926	0.000	0.000	0.000	0.521	0.000	0.521	0.000	0.000	0.000
41	0.983	0.000	0.983	-0.004	0.000	-0.004	0.504	0.000	0.504	0.004	0.000	0.004
42	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
43	0.779	0.000	0.779	0.000	0.000	0.000	0.583	0.000	0.583	0.000	0.000	0.000
44	0.875	0.000	0.875	0.000	0.000	0.000	0.538	0.000	0.538	0.000	0.000	0.000
45	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
46	0.991	0.000	0.991	0.000	0.000	0.000	0.502	0.000	0.502	0.000	0.000	0.000
47	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
48	0.926	0.000	0.926	0.000	0.000	0.000	0.521	0.000	0.521	0.000	0.000	0.000
49	0.855	0.000	0.855	0.000	0.000	0.000	0.547	0.000	0.547	0.000	0.000	0.000
50	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
51	0.789	0.000	0.789	-0.003	0.000	-0.003	0.577	0.000	0.577	0.003	0.000	0.003
52	0.787	0.000	0.787	0.000	0.000	0.000	0.578	0.000	0.578	0.000	0.000	0.000

Precision										Recall							
S/N	hash	fix		hash	bug		fix	bug		hash	fix		hash	bug		fix	bug
53	0.988	0.000		0.988	0.000		0.000	0.000		0.503	0.000		0.503	0.000		0.000	0.000
54	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
55	0.759	0.017		0.759	0.000		0.017	0.000		0.595	0.018		0.595	0.000		0.018	0.000
56	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
57	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
58	0.954	0.000		0.954	0.000		0.000	0.000		0.512	0.000		0.512	0.000		0.000	0.000
59	0.857	0.000		0.857	0.000		0.000	0.000		0.545	0.000		0.545	0.000		0.000	0.000
60	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
61	0.9	0.009		0.9	0.000		0.009	0.000		0.529	0.009		0.529	0.000		0.009	0.000
62	0.857	0.000		0.857	0.000		0.000	0.000		0.545	0.000		0.545	0.000		0.000	0.000
63	0.925	0.015		0.925	0.000		0.015	0.000		0.521	0.015		0.521	0.000		0.015	0.000
64	0.256	0.000		0.256	0.000		0.000	0.000		-1.105	0.000		-1.105	0.000		0.000	0.000
65	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
66	0.75	0.000		0.75	0.503		0.000	0.503		0.600	0.000		0.600	0.990		0.000	0.990
67	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
68	0.867	0.000		0.867	0.000		0.000	0.000		0.541	0.000		0.541	0.000		0.000	0.000
69	0.917	0.000		0.917	0.000		0.000	0.000		0.524	0.000		0.524	0.000		0.000	0.000
70	0.859	0.000		0.859	0.000		0.000	0.000		0.545	0.000		0.545	0.000		0.000	0.000
71	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
72	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
73	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
74	0.985	0.000		0.985	0.000		0.000	0.000		0.504	0.000		0.504	0.000		0.000	0.000
75	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
76	0.974	0.000		0.974	0.000		0.000	0.000		0.507	0.000		0.507	0.000		0.000	0.000
77	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
78	0.61	0.000		0.61	-0.002		0.000	-0.002		0.735	0.000		0.735	0.001		0.000	0.001
79	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
80	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
81	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
82	0.854	0.000		0.854	0.000		0.000	0.000		0.547	0.000		0.547	0.000		0.000	0.000
83	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
84	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
85	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
86	0.552	1.345		0.552	-0.037		1.345	-0.037		0.842	-3.900		0.842	0.033		-3.900	0.033
87	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
88	0.824	0.000		0.824	0.000		0.000	0.000		0.560	0.000		0.560	0.000		0.000	0.000
89	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
90	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
91	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
92	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
93	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
94	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
95	0	0.000		0	0.500		0.000	0.500		0.000	0.000		0.000	0.999		0.000	0.999
96	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
97	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
98	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
99	0	0.000		0	0.000		0.000	0.000		0.000	0.000		0.000	0.000		0.000	0.000
100	0.796	0.000		0.796	0.000		0.000	0.000		0.574	0.000		0.574	0.000		0.000	0.000

		Precision						**Recall**					
S/N	hash	fix	hash	bug	fix	bug	hash	fix	hash	bug	fix	bug
101	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
102	0.988	0.000	0.988	0.000	0.000	0.000	0.503	0.000	0.503	0.000	0.000	0.000
103	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
104	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
105	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
106	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
107	0.739	0.000	0.739	0.000	0.000	0.000	0.607	0.000	0.607	0.000	0.000	0.000
108	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
109	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
110	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
111	0.929	0.000	0.929	0.000	0.000	0.000	0.520	0.273	0.520	0.000	0.273	0.000
112	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
113	0.795	0.000	0.795	0.000	0.000	0.000	0.574	0.000	0.574	0.000	0.000	0.000
114	0.286	0.000	0.286	0.000	0.000	0.000	-2.000	0.000	-2.000	0.000	0.000	0.000
115	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
116	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
117	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
118	0.421	0.079	0.421	0.000	0.079	0.000	1.600	0.086	1.600	0.000	0.086	0.000
119	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
120	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
121	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
122	0.916	0.000	0.916	0.000	0.000	0.000	0.524	0.000	0.524	0.000	0.000	0.000
123	0.991	0.000	0.991	-0.013	0.000	-0.013	0.502	0.000	0.502	0.013	0.000	0.013
124	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
125	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
126	0.996	0.000	0.996	-0.181	0.000	-0.181	0.501	0.000	0.501	0.117	0.000	0.117
127	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
128	0.5	0.000	0.5	0.000	0.000	0.000	1.000	0.000	1.000	0.000	0.000	0.000
129	0.977	0.000	0.977	0.000	0.000	0.000	0.506	0.000	0.506	0.000	0.000	0.000
130	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
131	0.85	0.000	0.85	0.000	0.000	0.000	0.548	0.000	0.548	0.000	0.000	0.000
132	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
133	0.667	0.000	0.667	0.000	0.000	0.000	0.667	0.000	0.667	0.000	0.000	0.000
134	0.8	0.002	0.8	-0.007	0.002	-0.007	0.572	0.002	0.572	0.007	0.002	0.007
135	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
136	0	0.000	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
137	0.998	0.000	0.998	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
138	0.938	0.006	0.938	-0.020	0.006	-0.020	0.517	0.007	0.517	0.019	0.007	0.019
139	0.963	0.000	0.963	0.000	0.000	0.000	0.510	0.000	0.510	0.000	0.000	0.000
140	0.906	0.002	0.906	-0.006	0.002	-0.006	0.527	0.002	0.527	0.006	0.002	0.006
141	0.974	0.000	0.974	0.000	0.000	0.000	0.507	0.000	0.507	0.000	0.000	0.000
142	0.995	0.003	0.995	0.000	0.003	0.000	0.501	0.003	0.501	0.000	0.003	0.000
143	0.981	0.000	0.981	-0.002	0.000	-0.002	0.505	0.000	0.505	0.002	0.000	0.002
144	0.991	0.000	0.991	0.000	0.000	0.000	0.502	0.000	0.502	0.000	0.000	0.000
145	0.951	0.000	0.951	0.000	0.000	0.000	0.513	0.000	0.513	0.000	0.000	0.000
146	0.98	0.000	0.98	-0.002	0.000	-0.002	0.505	0.000	0.505	0.002	0.000	0.002
147	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
148	0.883	0.000	0.883	0.000	0.000	0.000	0.536	0.000	0.536	0.000	0.000	0.000
149	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
150	0.739	0.696	0.739	0.000	0.696	0.000	0.607	2.286	0.607	0.000	2.286	0.000
151	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
152	0.833	0.064	0.833	-0.057	0.064	-0.057	0.556	0.068	0.556	0.049	0.068	0.049
153	0.81	0.004	0.81	-0.002	0.004	-0.002	0.566	0.004	0.566	0.002	0.004	0.002
154	1	0.000	1	0.000	0.000	0.000	0.500	0.000	0.500	0.000	0.000	0.000
155	0.897	0.000	0.897	0.000	0.000	0.000	0.530	0.000	0.530	0.000	0.000	0.000

	Precision							Recall					
S/N	hash	fix	hash	bug	fix	bug		hash	fix	hash	bug	fix	bug
156	0.841	0.002	0.841	-0.001	0.002	-0.001		0.552	0.002	0.552	0.001	0.002	0.001
157	0.549	0.073	0.549	-0.308	0.073	-0.308		0.849	0.079	0.849	0.160	0.079	0.160
158	0.932	0.000	0.932	0.000	0.000	0.000		0.519	0.000	0.519	0.000	0.000	0.000
159	1	0.000	1	0.000	0.000	0.000		0.500	0.000	0.500	0.000	0.000	0.000
160	0.999	0.000	0.999	0.000	0.000	0.000		0.500	0.000	0.500	0.000	0.000	0.000
161	0.902	0.000	0.902	0.000	0.000	0.000		0.529	0.000	0.529	0.000	0.000	0.000
162	0.857	0.000	0.857	-0.375	0.000	-0.375		0.545	0.000	0.545	0.176	0.000	0.176
163	0.835	0.000	0.835	0.000	0.000	0.000		0.555	0.000	0.555	0.000	0.000	0.000
164	0.997	0.000	0.997	0.000	0.000	0.000		0.501	0.000	0.501	0.000	0.000	0.000
165	0.989	0.000	0.989	0.000	0.000	0.000		0.503	0.000	0.503	0.000	0.000	0.000
166	0.881	0.000	0.881	0.000	0.000	0.000		0.536	0.000	0.536	0.000	0.000	0.000
167	0.644	0.000	0.644	0.000	0.000	0.000		0.691	0.000	0.691	0.000	0.000	0.000
168	0.982	0.000	0.982	0.000	0.000	0.000		0.505	0.000	0.505	0.000	0.000	0.000
169	0.944	0.000	0.944	0.000	0.000	0.000		0.515	0.000	0.515	0.000	0.000	0.000
170	0.917	0.111	0.917	7.412	0.111	7.412		0.524	0.124	0.524	0.349	0.124	0.349
171	0.762	0.000	0.762	0.000	0.000	0.000		0.593	0.000	0.593	0.000	0.000	0.000
172	0	0.000	0	0.000	0.000	0.000		0.000	0.000	0.000	0.000	0.000	0.000
173	0.725	0.855	0.725	0.000	0.855	0.000		0.617	5.900	0.617	0.000	5.900	0.000
174	0.994	0.000	0.994	0.000	0.000	0.000		0.502	0.000	0.502	0.000	0.000	0.000
175	0.585	0.116	0.585	-1.347	0.116	-1.347		0.775	0.131	0.775	0.267	0.131	0.267
176	0.857	0.009	0.857	0.000	0.009	0.000		0.545	0.009	0.545	0.000	0.009	0.000
177	0.999	0.000	0.999	0.000	0.000	0.000		0.500	0.000	0.500	0.000	0.000	0.000
178	0.978	0.000	0.978	0.000	0.000	0.000		0.506	0.000	0.506	0.000	0.000	0.000
179	0.843	0.002	0.843	0.000	0.002	0.000		0.551	0.002	0.551	0.000	0.002	0.000
180	0.951	0.000	0.951	0.000	0.000	0.000		0.513	0.000	0.513	0.000	0.000	0.000
181	0.977	0.000	0.977	0.000	0.000	0.000		0.506	0.000	0.506	0.000	0.000	0.000
182	0.971	0.000	0.971	0.000	0.000	0.000		0.508	0.000	0.508	0.000	0.000	0.000
183	0	0.000	0	0.000	0.000	0.000		0.000	0.000	0.000	0.000	0.000	0.000
184	0.982	0.000	0.982	0.000	0.000	0.000		0.505	0.000	0.505	0.000	0.000	0.000
185	0.988	0.000	0.988	0.000	0.000	0.000		0.503	0.000	0.503	0.000	0.000	0.000
186	0.881	0.003	0.881	-0.003	0.003	-0.003		0.536	0.003	0.536	0.003	0.003	0.003
187	0.954	0.002	0.954	0.000	0.002	0.000		0.512	0.002	0.512	0.000	0.002	0.000
188	0.829	0.015	0.829	-0.011	0.015	-0.011		0.558	0.015	0.558	0.010	0.015	0.010
189	0.949	0.000	0.949	0.000	0.000	0.000		0.514	0.000	0.514	0.000	0.000	0.000
190	0.684	0.004	0.684	-0.013	0.004	-0.013		0.650	0.004	0.650	0.012	0.004	0.012
191	0.789	0.000	0.789	-0.012	0.000	-0.012		0.577	0.000	0.577	0.011	0.000	0.011
192	1	0.000	1	0.000	0.000	0.000		0.500	0.000	0.500	0.000	0.000	0.000
193	0.985	0.000	0.985	0.000	0.000	0.000		0.504	0.000	0.504	0.000	0.000	0.000
194	0.833	0.000	0.833	0.000	0.000	0.000		0.556	0.000	0.556	0.000	0.000	0.000
195	0.99	0.000	0.99	0.000	0.000	0.000		0.503	0.000	0.503	0.000	0.000	0.000
196	0.994	0.000	0.994	0.000	0.000	0.000		0.502	0.000	0.502	0.000	0.000	0.000
197	0.99	0.000	0.99	0.000	0.000	0.000		0.502	0.000	0.502	0.000	0.000	0.000
198	0.975	0.000	0.975	0.000	0.000	0.000		0.507	0.000	0.507	0.000	0.000	0.000
199	0.969	0.002	0.969	-0.002	0.002	-0.002		0.508	0.002	0.508	0.002	0.002	0.002

Precision									**Recall**							
S/N	hash	fix		hash	bug		fix	bug	hash	fix		hash	bug		fix	bug
200	0.981	0.000		0.981	0.000		0.000	0.000	0.505	0.000		0.505	0.000		0.000	0.000
201	0.949	0.000		0.949	0.000		0.000	0.000	0.514	0.000		0.514	0.000		0.000	0.000
202	1	0.000		1			0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
203	0.992	0.000		0.992	0.000		0.000	0.000	0.502	0.000		0.502	0.000		0.000	0.000
204	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
205	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
206	0.783	0.160		0.783	-0.187		0.160	-0.187	0.580	0.190		0.580	0.120		0.190	0.120
207	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
208	0.997	0.004		0.997	0.000		0.004	0.000	0.501	0.004		0.501	0.000		0.004	0.000
209	0.961	3.813		0.961	0.000		3.813	0.000	0.510	-1.355		0.510	0.000		-1.355	0.000
210	0.993	0.000		0.993	0.000		0.000	0.000	0.502	0.000		0.502	0.000		0.000	0.000
211	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
212	0	0.000		0	0.000		0.000	0.000	0.000	0.000		0.000	0.000		0.000	0.000
213	0	0.000		0	0.000		0.000	0.000	0.000	0.000		0.000	0.000		0.000	0.000
214	0.998	0.000		0.998	0.000		0.000	0.000	0.501	0.000		0.501	0.000		0.000	0.000
215	0.807	0.000		0.807	-0.001		0.000	-0.001	0.568	0.000		0.568	0.001		0.000	0.001
216	0.842	0.000		0.842	-0.059		0.000	-0.059	0.552	0.000		0.552	0.050		0.000	0.050
217	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
218	0.736	0.003		0.736	0.000		0.003	0.000	0.609	0.003		0.609	0.000		0.003	0.000
219	0.821	0.000		0.821	0.000		0.000	0.000	0.561	0.000		0.561	0.000		0.000	0.000
220	0.874	0.000		0.874	-0.003		0.000	-0.003	0.539	0.000		0.539	0.003		0.000	0.003
221	0.866	0.160		0.866	-0.109		0.160	-0.109	0.542	0.190		0.542	0.082		0.190	0.082
222	0.991	0.000		0.991	0.000		0.000	0.000	0.502	0.000		0.502	0.000		0.000	0.000
223	0.999	0.000		0.999	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
224	0	0.000		0	0.000		0.000	0.000	0.000	0.000		0.000	0.000		0.000	0.000
225	0.992	0.000		0.992	0.000		0.000	0.000	0.502	0.000		0.502	0.000		0.000	0.000
226	0.566	0.048		0.566	-0.053		0.048	-0.053	0.810	0.051		0.810	0.046		0.051	0.046
227	0.973	0.000		0.973	0.000		0.000	0.000	0.507	0.000		0.507	0.000		0.000	0.000
228	0.99	0.000		0.99	0.000		0.000	0.000	0.503	0.000		0.503	0.000		0.000	0.000
229	0.952	0.000		0.952	0.000		0.000	0.000	0.513	0.000		0.513	0.000		0.000	0.000
230	0.982	0.123		0.982	-0.323		0.123	-0.323	0.505	0.140		0.505	0.164		0.140	0.164
231	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
232	0.885	0.000		0.885	0.000		0.000	0.000	0.535	0.000		0.535	0.000		0.000	0.000
233	0.91	0.000		0.91	0.000		0.000	0.000	0.526	0.000		0.526	0.000		0.000	0.000
234	0.995	0.000		0.995	0.000		0.000	0.000	0.501	0.000		0.501	0.000		0.000	0.000
235	0.996	0.000		0.996	0.000		0.000	0.000	0.501	0.000		0.501	0.000		0.000	0.000
236	0.961	0.000		0.961	0.000		0.000	0.000	0.510	0.000		0.510	0.000		0.000	0.000
237	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
238	0.956	0.000		0.956	0.000		0.000	0.000	0.512	0.000		0.512	0.000		0.000	0.000
239	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
240	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000

	Precision								**Recall**							
S/N	hash	fix		hash	bug		fix	bug	hash	fix		hash	bug		fix	bug
241	0.999	0.000		0.999	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
242	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
243	0.703	0.024		0.703	-0.010		0.024	-0.010	0.634	0.025		0.634	0.009		0.025	0.009
244	0.959	0.000		0.959	0.000		0.000	0.000	0.511	0.000		0.511	0.000		0.000	0.000
245	0.631	0.012		0.631	-0.018		0.012	-0.018	0.706	0.012		0.706	0.017		0.012	0.017
246	0.998	0.000		0.998	0.000		0.000	0.000	0.501	0.000		0.501	0.000		0.000	0.000
247	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
248	0.79	0.129		0.79	0.000		0.129	0.000	0.576	0.148		0.576	0.000		0.148	0.000
249	0.976	0.000		0.976	0.000		0.000	0.000	0.506	0.000		0.506	0.000		0.000	0.000
250	0.955	0.000		0.955	0.000		0.000	0.000	0.512	0.000		0.512	0.000		0.000	0.000
251	0.99	0.000		0.99	0.000		0.000	0.000	0.503	0.000		0.503	0.000		0.000	0.000
252	0.973	0.000		0.973	0.000		0.000	0.000	0.507	0.000		0.507	0.000		0.000	0.000
253	0.326	0.721		0.326	1.524		0.721	1.524	-14.000	2.583		-14.000	0.427		2.583	0.427
254	0.547	0.086		0.547	-0.092		0.086	-0.092	0.852	0.094		0.852	0.072		0.094	0.072
255	0.999	0.000		0.999	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
256	0.974	0.000		0.974	0.000		0.000	0.000	0.507	0.000		0.507	0.000		0.000	0.000
257	0.944	0.000		0.944	0.000		0.000	0.000	0.515	0.000		0.515	0.000		0.000	0.000
258	0.926	0.000		0.926	0.000		0.000	0.000	0.521	0.000		0.521	0.000		0.000	0.000
259	0.916	0.000		0.916	0.000		0.000	0.000	0.524	0.000		0.524	0.000		0.000	0.000
260	0.951	0.000		0.951	-0.010		0.000	-0.010	0.513	0.000		0.513	0.010		0.000	0.010
261	0.208	0.006		0.208	-0.020		0.006	-0.020	-0.552	0.007		-0.552	0.019		0.007	0.019
262	0.845	0.002		0.845	-0.005		0.002	-0.005	0.551	0.002		0.551	0.005		0.002	0.005
263	0	0.000		0	0.000		0.000	0.000	0.000	0.000		0.000	0.000		0.000	0.000
264	0.9	0.000		0.9	0.000		0.000	0.000	0.529	0.000		0.529	0.000		0.000	0.000
265	0.982	0.013		0.982	-0.038		0.013	-0.038	0.505	0.014		0.505	0.034		0.014	0.034
266	0.927	0.000		0.927	0.000		0.000	0.000	0.521	0.000		0.521	0.000		0.000	0.000
267	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
268	0.994	0.000		0.994	0.000		0.000	0.000	0.501	0.000		0.501	0.000		0.000	0.000
269	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
270	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
271	0	0.000		0	0.000		0.000	0.000	0.000	0.000		0.000	0.000		0.000	0.000
272	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
273	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
274	0	0.000		0	0.000		0.000	0.000	0.000	0.000		0.000	0.000		0.000	0.000
275	0.932	0.000		0.932	0.000		0.000	0.000	0.519	0.000		0.519	0.000		0.000	0.000
276	1	0.000		1	0.000		0.000	0.000	0.500	0.000		0.500	0.000		0.000	0.000
277	0.912	0.000		0.912	0.000		0.000	0.000	0.525	0.000		0.525	0.000		0.000	0.000
278	0.995	0.000		0.995	0.000		0.000	0.000	0.501	0.000		0.501	0.000		0.000	0.000
279	0.774	0.000		0.774	0.000		0.000	0.000	0.585	0.000		0.585	0.000		0.000	0.000
280	0.987	0.000		0.987	0.000		0.000	0.000	0.503	0.000		0.503	0.000		0.000	0.000
281	0.647	0.000		0.647	-0.003		0.000	-0.003	0.688	0.000		0.688	0.003		0.000	0.003
282	0.984	0.000		0.984	-0.002		0.000	-0.002	0.504	0.000		0.504	0.002		0.000	0.002
283	0.971	0.000		0.971	0.000		0.000	0.000	0.508	0.000		0.508	0.000		0.000	0.000
284	0.966	0.000		0.966	0.000		0.000	0.000	0.509	0.000		0.509	0.000		0.000	0.000
285	0.964	0.000		0.964	0.000		0.000	0.000	0.510	0.000		0.510	0.000		0.000	0.000
286	0.971	0.000		0.971	0.000		0.000	0.000	0.508	0.000		0.508	0.000		0.000	0.000
287	0.906	0.002		0.906	-0.002		0.002	-0.002	0.528	0.002		0.528	0.002		0.002	0.002
288	0.996	0.000		0.996	0.000		0.000	0.000	0.501	0.000		0.501	0.000		0.000	0.000
289	0.827	0.000		0.827	0.000		0.000	0.000	0.558	0.000		0.558	0.000		0.000	0.000
290	0.964	0.000		0.964	0.000		0.000	0.000	0.510	0.000		0.510	0.000		0.000	0.000

	Precision									Recall							
S/N	hash	fix		hash	bug		fix	bug		hash	fix		hash	bug		fix	bug
291	0.892	0.000		0.892	0.000		0.000	0.000		0.532	0.000		0.532	0.000		0.000	0.000
292	0.965	0.000		0.965	0.000		0.000	0.000		0.509	0.000		0.509	0.000		0.000	0.000
293	0.957	0.000		0.957	0.000		0.000	0.000		0.511	0.000		0.511	0.000		0.000	0.000
294	0.877	0.973		0.877	0.572		0.973	0.572		0.538	36.711		0.538	0.799		36.711	0.799
295	0.831	0.001		0.831	0.000		0.001	0.000		0.557	0.001		0.557	0.000		0.001	0.000
296	0.951	0.000		0.951	0.000		0.000	0.000		0.513	0.000		0.513	0.000		0.000	0.000
297	0.942	0.172		0.942	0.706		0.172	0.706		0.516	0.208		0.516	0.631		0.208	0.631
298	0.939	0.000		0.939	0.000		0.000	0.000		0.517	0.000		0.517	0.000		0.000	0.000
299	0.909	0.000		0.909	0.000		0.000	0.000		0.526	0.000		0.526	0.000		0.000	0.000
300	0.896	0.000		0.896	0.000		0.000	0.000		0.531	0.000		0.531	0.000		0.000	0.000
301	0.882	0.000		0.882	0.000		0.000	0.000		0.536	0.000		0.536	0.000		0.000	0.000
302	0.865	0.000		0.865	0.000		0.000	0.000		0.542	0.000		0.542	0.000		0.000	0.000
303	0.903	0.000		0.903	0.000		0.000	0.000		0.528	0.000		0.528	0.000		0.000	0.000
304	0.856	0.000		0.856	0.000		0.000	0.000		0.546	0.000		0.546	0.000		0.000	0.000
305	0.922	0.016		0.922	-0.342		0.016	-0.342		0.522	0.016		0.522	0.169		0.016	0.169
306	0.984	0.000		0.984	0.000		0.000	0.000		0.504	0.000		0.504	0.000		0.000	0.000
307	0.981	0.085		0.981	0.000		0.085	0.000		0.505	0.093		0.505	0.000		0.093	0.000
308	0.948	0.000		0.948	0.000		0.000	0.000		0.514	0.000		0.514	0.000		0.000	0.000
309	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
310	0.994	0.000		0.994	0.000		0.000	0.000		0.502	0.000		0.502	0.000		0.000	0.000
311	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
312	0.963	0.051		0.963	0.000		0.051	0.000		0.510	0.054		0.510	0.000		0.054	0.000
313	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
314	0.719	0.000		0.719	0.000		0.000	0.000		0.621	0.000		0.621	0.000		0.000	0.000
315	0.765	0.000		0.765	0.000		0.000	0.000		0.591	0.000		0.591	0.000		0.000	0.000
316	0.87	0.000		0.87	-0.002		0.000	-0.002		0.540	0.000		0.540	0.002		0.000	0.002
317	0.879	0.003		0.879	0.000		0.003	0.000		0.537	0.003		0.537	0.000		0.003	0.000
318	0.986	0.000		0.986	0.000		0.000	0.000		0.503	0.000		0.503	0.000		0.000	0.000
319	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
320	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
321	0.901	0.000		0.901	0.000		0.000	0.000		0.529	0.000		0.529	0.000		0.000	0.000
322	0.76	0.000		0.76	0.000		0.000	0.000		0.594	0.000		0.594	0.000		0.000	0.000
323	0.984	0.000		0.984	0.000		0.000	0.000		0.504	0.000		0.504	0.000		0.000	0.000
324	0.957	0.000		0.957	0.000		0.000	0.000		0.511	0.000		0.511	0.000		0.000	0.000
325	0.838	0.000		0.838	0.000		0.000	0.000		0.553	0.000		0.553	0.000		0.000	0.000
326	0.944	0.000		0.944	0.000		0.000	0.000		0.515	0.000		0.515	0.000		0.000	0.000
327	0.941	0.000		0.941	0.000		0.000	0.000		0.516	0.000		0.516	0.000		0.000	0.000
328	0.333	0.833		0.333	0.000		0.833	0.000		0.000	5.000		0.000	0.000		5.000	0.000
329	0.993	0.000		0.993	0.000		0.000	0.000		0.502	0.000		0.502	0.000		0.000	0.000
330	0.96	0.000		0.96	0.000		0.000	0.000		0.511	0.000		0.511	0.000		0.000	0.000
331	0.988	0.000		0.988	0.000		0.000	0.000		0.503	0.000		0.503	0.000		0.000	0.000
332	0.786	0.004		0.786	-0.004		0.004	-0.004		0.579	0.004		0.579	0.004		0.004	0.004
333	0.942	0.007		0.942	-0.017		0.007	-0.017		0.516	0.007		0.516	0.016		0.007	0.016
334	0.629	0.070		0.629	-0.100		0.070	-0.100		0.709	0.076		0.709	0.077		0.076	0.077
335	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
336	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
337	1	0.000		1	0.000		0.000	0.000		0.500	0.000		0.500	0.000		0.000	0.000
338	0.817	0.029		0.817	-0.010		0.029	-0.010		0.563	0.030		0.563	0.010		0.030	0.010
339	0.87	0.019		0.87	0.000		0.019	0.000		0.540	0.019		0.540	0.000		0.019	0.000
340	0.943	0.000		0.943	0.000		0.000	0.000		0.516	0.000		0.516	0.000		0.000	0.000
341	0.963	0.000		0.963	0.000		0.000	0.000		0.510	0.000		0.510	0.000		0.000	0.000
342	0.762	0.000		0.762	0.000		0.000	0.000		0.593	0.000		0.593	0.000		0.000	0.000
343	0.9	0.000		0.9	0.000		0.000	0.000		0.529	0.000		0.529	0.000		0.000	0.000
344	0.894	0.007		0.894	0.000		0.007	0.000		0.532	0.007		0.532	0.000		0.007	0.000

Table 5: Synchronisation of BT data and VC log using # Symbol of the SZZ Algorithm - 344 OSS Projects

Proj. IDs	**No intersection**	**Classic**	**Bicho contained**	**CVS. contained**	**perfect**	**Both empty**	**Shered bug coverage**	**Scen. 1**	**Scen. 2**	**Scen. 3**	**Scen. 4**	**Bicho Delta**	**CVS. Delta**
1	0	0	1	0	0	0	0.105	0	0	1	0	0.00%	89.47%
2	0	0	0	0	0	0	0.042	0	0	1	0	0.00%	95.77%
3	0	0	1	0	0	0	0.038	0	0	1	0	0.00%	96.20%
4	0	1	0	0	0	0	0.065	0	1	0	0	0.47%	92.99%
5	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
6	0	1	0	0	0	0	0.196	0	1	0	0	0.98%	79.41%
7	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
8	0	1	0	0	0	0	0.137	0	1	0	0	1.08%	85.22%
9	0	0	1	0	0	0	0.059	0	0	1	0	0.00%	94.12%
10	1	0	0	0	0	0	0.000	1	0	0	0	33.33%	66.67%
11	0	1	0	0	0	0	0.105	0	1	0	0	5.26%	84.21%
12	0	0	1	0	0	0	0.034	0	0	1	0	0.00%	96.55%
13	0	1	0	0	0	0	0.002	0	1	0	0	97.47%	2.35%
14	0	1	0	0	0	0	0.240	0	1	0	0	0.44%	75.56%
15	0	1	0	0	0	0	0.096	0	1	0	0	1.52%	79.80%
16	0	1	0	0	0	0	0.109	0	1	0	0	0.40%	88.71%
17	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
18	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
19	0	1	0	0	0	0	0.006	0	1	0	0	3.18%	96.21%
20	0	0	1	0	0	0	0.030	0	0	1	0	0.00%	96.97%
21	0	0	1	0	0	0	0.019	0	0	1	0	0.00%	98.13%
22	0	0	1	0	0	0	0.025	0	0	1	0	0.00%	97.50%
23	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
24	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
25	1	0	0	0	0	0	0.000	1	0	0	0	21.43%	78.57%
26	0	1	0	0	0	0	0.011	0	1	0	0	38.52%	60.37%
27	0	0	1	0	0	0	0.140	0	0	1	0	0.00%	85.95%
28	0	0	1	0	0	0	0.209	0	0	1	0	0.00%	79.08%
29	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
30	1	0	0	0	0	0	0.000	1	0	0	0	46.15%	53.85%
31	0	1	0	0	0	0	0.774	0	1	0	0	2.93%	19.67%
32	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
33	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
34	1	0	0	0	0	0	0.000	1	0	0	0	1.42%	98.58%
35	0	0	1	0	0	0	0.077	0	0	1	0	0.00%	92.31%
36	0	0	1	0	0	0	0.037	0	0	1	0	0.00%	96.30%
37	0	0	1	0	0	0	0.128	0	0	1	0	0.00%	87.23%
38	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
39	0	1	0	0	0	0	0.109	0	1	0	0	1.77%	87.32%
40	0	0	1	0	0	0	0.074	0	0	1	0	0.00%	92.59%
41	0	1	0	0	0	0	0.015	0	1	0	0	11.61%	86.89%
42	1	0	0	0	0	0	0.000	1	0	0	0	79.53%	20.47%
43	0	1	0	0	0	0	0.220	0	1	0	0	0.24%	77.73%
44	0	0	1	0	0	0	0.125	0	0	1	0	0.00%	87.50%
45	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
46	0	0	1	0	0	0	0.009	0	0	1	0	0.00%	99.07%
47	1	0	0	0	0	0	0.000	1	0	0	0	58.73%	41.27%
48	0	0	1	0	0	0	0.074	0	0	1	0	0.00%	92.65%
49	0	1	0	0	0	0	0.143	0	1	0	0	1.79%	83.93%
50	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
51	0	1	0	0	0	0	0.207	0	1	0	0	1.85%	77.40%
52	0	0	1	0	0	0	0.213	0	0	1	0	0.00%	78.72%

Proj. IDs	**No intersection**	**Classic**	**Bicho contained**	**CVS. contained**	**perfect**	**Both empty**	**Shered bug coverage**	**Scen. 1**	**Scen. 2**	**Scen. 3**	**Scen. 4**	**Bicho Delta**	**CVS. Delta**
53	0	0	1	0	0	0	0.012	0	0	1	0	0.00%	98.83%
54	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
55	0	1	0	0	0	0	0.230	0	1	0	0	4.92%	72.13%
56	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
57	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
58	0	1	0	0	0	0	0.046	0	1	0	0	0.42%	94.96%
59	0	1	0	0	0	0	0.001	0	1	0	0	99.34%	0.57%
60	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
61	0	1	0	0	0	0	0.097	0	1	0	0	2.65%	87.61%
62	0	1	0	0	0	0	0.002	0	1	0	0	98.79%	1.04%
63	0	1	0	0	0	0	0.069	0	1	0	0	7.59%	85.52%
64	0	0	1	0	0	0	0.744	0	0	1	0	0.00%	25.61%
65	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
66	0	1	0	0	0	0	0.003	0	1	0	0	98.99%	0.76%
67	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
68	0	1	0	0	0	0	0.132	0	1	0	0	0.83%	86.01%
69	0	1	0	0	0	0	0.067	0	1	0	0	20.00%	73.33%
70	0	1	0	0	0	0	0.140	0	1	0	0	0.64%	85.39%
71	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
72	1	0	0	0	0	0	0.000	1	0	0	0	80.00%	20.00%
73	1	0	0	0	0	0	0.000	1	0	0	0	94.87%	5.13%
74	0	1	0	0	0	0	0.015	0	1	0	0	0.17%	98.31%
75	1	0	0	0	0	0	0.000	1	0	0	0	98.53%	1.47%
76	0	1	0	0	0	0	0.003	0	1	0	0	87.79%	11.89%
77	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
78	0	1	0	0	0	0	0.377	0	1	0	0	3.19%	59.07%
79	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
80	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
81	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
82	0	1	0	0	0	0	0.034	0	1	0	0	76.47%	20.10%
83	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
84	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
85	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
86	0	1	0	0	0	0	0.165	0	1	0	0	63.29%	20.25%
87	1	0	0	0	0	0	0.000	1	0	0	0	27.97%	72.03%
88	0	1	0	0	0	0	0.120	0	1	0	0	32.00%	56.00%
89	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
90	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
91	1	0	0	0	0	0	0.000	1	0	0	0	75.00%	25.00%
92	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
93	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
94	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
95	0	0	0	1	0	0	0.000	0	0	1	0	99.95%	0.00%
96	1	0	0	0	0	0	0.000	1	0	0	0	4.76%	95.24%
97	1	0	0	0	0	0	0.000	1	0	0	0	18.18%	81.82%
98	0	0	0	1	0	0	0.063	0	0	1	0	93.75%	0.00%
99	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
100	0	1	0	0	0	0	0.189	0	1	0	0	7.55%	73.58%

Proj. IDs	**No intersection**	**Classic**	**Bicho contained**	**CVS. contained**	**perfect**	**Both empty**	**Shered bug coverage**	**Scen. 1**	**Scen. 2**	**Scen. 3**	**Scen. 4**	**Bicho Delta**	**CVS. Delta**
101	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
102	0	1	0	0	0	0	0.012	0	1	0	0	1.19%	97.62%
103	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
104	1	0	0	0	0	0	0.000	1	0	0	0	3.03%	96.97%
105	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
106	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
107	0	1	0	0	0	0	0.025	0	1	0	0	90.38%	7.11%
108	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
109	1	0	0	0	0	0	0.000	1	0	0	0	90.54%	9.46%
110	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
111	0	1	0	0	0	0	0.020	0	1	0	0	72.68%	25.37%
112	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
113	0	1	0	0	0	0	0.203	0	1	0	0	1.27%	78.48%
114	0	1	0	0	0	0	0.132	0	1	0	0	81.58%	5.26%
115	1	0	0	0	0	0	0.000	1	0	0	0	5.84%	94.16%
116	1	0	0	0	0	0	0.000	1	0	0	0	89.47%	10.53%
117	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
118	0	1	0	0	0	0	0.537	0	1	0	0	7.32%	39.02%
119	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
120	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
121	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
122	0	1	0	0	0	0	0.079	0	1	0	0	6.43%	85.71%
123	0	1	0	0	0	0	0.008	0	1	0	0	2.10%	97.06%
124	1	0	0	0	0	0	0.000	1	0	0	0	17.32%	82.68%
125	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
126	0	1	0	0	0	0	0.003	0	1	0	0	12.18%	87.50%
127	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
128	0	1	0	0	0	0	0.050	0	1	0	0	90.00%	5.00%
129	0	1	0	0	0	0	0.020	0	1	0	0	15.69%	82.35%
130	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
131	0	1	0	0	0	0	0.066	0	1	0	0	55.92%	37.47%
132	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
133	0	1	0	0	0	0	0.003	0	1	0	0	99.20%	0.53%
134	0	1	0	0	0	0	0.186	0	1	0	0	7.13%	74.26%
135	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
136	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
137	0	0	1	0	0	0	0.002	0	0	1	0	0.00%	99.80%
138	0	1	0	0	0	0	0.043	0	1	0	0	30.17%	65.49%
139	0	1	0	0	0	0	0.029	0	1	0	0	19.93%	77.12%
140	0	1	0	0	0	0	0.082	0	1	0	0	12.86%	78.99%
141	0	1	0	0	0	0	0.026	0	1	0	0	0.13%	97.25%
142	0	1	0	0	0	0	0.005	0	1	0	0	1.27%	98.23%
143	0	1	0	0	0	0	0.019	0	1	0	0	0.23%	97.83%
144	0	0	1	0	0	0	0.009	0	0	1	0	0.00%	99.05%
145	0	1	0	0	0	0	0.044	0	1	0	0	9.05%	86.53%
146	0	1	0	0	0	0	0.019	0	1	0	0	1.51%	96.56%
147	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
148	0	1	0	0	0	0	0.113	0	1	0	0	3.64%	85.07%
149	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
150	0	1	0	0	0	0	0.068	0	1	0	0	73.86%	19.32%
151	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
152	0	1	0	0	0	0	0.126	0	1	0	0	24.27%	63.11%
153	0	1	0	0	0	0	0.165	0	1	0	0	13.19%	70.34%
154	1	0	0	0	0	0	0.000	1	0	0	0	94.82%	5.18%
155	0	1	0	0	0	0	0.103	0	1	0	0	0.17%	89.55%
156	0	1	0	0	0	0	0.155	0	1	0	0	2.68%	81.84%
157	0	1	0	0	0	0	0.079	0	1	0	0	82.58%	9.56%
158	0	1	0	0	0	0	0.067	0	1	0	0	2.03%	91.28%
159	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
160	0	0	1	0	0	0	0.001	0	0	1	0	0.00%	99.86%
161	0	1	0	0	0	0	0.098	0	1	0	0	0.21%	89.97%
162	0	1	0	0	0	0	0.015	0	1	0	0	89.63%	8.89%
163	0	1	0	0	0	0	0.109	0	1	0	0	34.16%	54.95%

Proj. IDs	**No intersection**	**Classic**	**Bicho contained**	**CVS. contained**	**perfect**	**Both empty**	**Shered bug coverage**	**Scen. 1**	**Scen. 2**	**Scen. 3**	**Scen. 4**	**Bicho Delta**	**CVS. Delta**
164	0	1	0	0	0	0	0.003	0	1	0	0	1.15%	98.57%
165	0	1	0	0	0	0	0.011	0	1	0	0	0.31%	98.62%
166	0	1	0	0	0	0	0.118	0	1	0	0	0.59%	87.57%
167	0	1	0	0	0	0	0.343	0	1	0	0	3.81%	61.90%
168	0	1	0	0	0	0	0.014	0	1	0	0	20.14%	78.42%
169	0	1	0	0	0	0	0.056	0	1	0	0	1.59%	92.86%
170	0	1	0	0	0	0	0.047	0	1	0	0	43.65%	51.68%
171	0	1	0	0	0	0	0.222	0	1	0	0	7.03%	70.81%
172	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
173	0	1	0	0	0	0	0.050	0	1	0	0	81.79%	13.19%
174	0	0	1	0	0	0	0.006	0	0	1	0	0.00%	99.36%
175	0	1	0	0	0	0	0.087	0	1	0	0	79.05%	12.25%
176	0	1	0	0	0	0	0.107	0	1	0	0	24.83%	64.43%
177	0	1	0	0	0	0	0.001	0	1	0	0	0.24%	99.69%
178	0	0	1	0	0	0	0.022	0	0	1	0	0.00%	97.83%
179	0	1	0	0	0	0	0.152	0	1	0	0	2.87%	81.90%
180	0	0	1	0	0	0	0.049	0	0	1	0	0.00%	95.06%
181	0	1	0	0	0	0	0.023	0	1	0	0	0.18%	97.50%
182	0	1	0	0	0	0	0.006	0	1	0	0	77.92%	21.43%
183	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
184	0	1	0	0	0	0	0.018	0	1	0	0	0.07%	98.11%
185	0	1	0	0	0	0	0.002	0	1	0	0	84.09%	15.72%
186	0	1	0	0	0	0	0.119	0	1	0	0	0.50%	87.65%
187	0	1	0	0	0	0	0.046	0	1	0	0	1.06%	94.36%
188	0	1	0	0	0	0	0.094	0	1	0	0	44.82%	45.74%
189	0	1	0	0	0	0	0.051	0	1	0	0	0.22%	94.69%
190	0	1	0	0	0	0	0.201	0	1	0	0	36.33%	43.53%
191	0	1	0	0	0	0	0.187	0	1	0	0	11.51%	69.82%
192	1	0	0	0	0	0	0.000	1	0	0	0	0.28%	99.72%
193	0	1	0	0	0	0	0.015	0	1	0	0	0.33%	98.20%
194	0	1	0	0	0	0	0.002	0	1	0	0	98.94%	0.88%
195	0	1	0	0	0	0	0.010	0	1	0	0	0.98%	98.05%
196	0	1	0	0	0	0	0.006	0	1	0	0	0.09%	99.28%
197	0	1	0	0	0	0	0.009	0	1	0	0	5.45%	93.64%
198	0	1	0	0	0	0	0.003	0	1	0	0	89.28%	10.45%
199	0	1	0	0	0	0	0.028	0	1	0	0	10.82%	86.39%
200	0	0	1	0	0	0	0.019	0	0	1	0	0.00%	98.08%
201	0	1	0	0	0	0	0.051	0	1	0	0	0.35%	94.60%
202	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
203	0	0	1	0	0	0	0.008	0	0	1	0	0.00%	99.17%
204	1	0	0	0	0	0	0.000	1	0	0	0	98.59%	1.41%
205	1	0	0	0	0	0	0.000	1	0	0	0	1.39%	98.61%
206	0	1	0	0	0	0	0.163	0	1	0	0	25.00%	58.72%
207	1	0	0	0	0	0	0.000	1	0	0	0	4.39%	95.61%
208	0	1	0	0	0	0	0.003	0	1	0	0	1.58%	98.11%
209	0	1	0	0	0	0	0.003	0	1	0	0	93.31%	6.43%
210	0	0	1	0	0	0	0.007	0	0	1	0	0.00%	99.35%
211	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
212	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
213	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
214	0	0	1	0	0	0	0.002	0	0	1	0	0.00%	99.79%
215	0	1	0	0	0	0	0.187	0	1	0	0	3.14%	78.16%
216	0	1	0	0	0	0	0.097	0	1	0	0	38.71%	51.61%
217	0	0	1	0	0	0	0.000	0	0	1	0	0.00%	99.95%
218	0	1	0	0	0	0	0.260	0	1	0	0	1.57%	72.41%
219	0	1	0	0	0	0	0.166	0	1	0	0	7.01%	76.38%
220	0	1	0	0	0	0	0.110	0	1	0	0	12.83%	76.20%
221	0	1	0	0	0	0	0.046	0	1	0	0	65.71%	29.68%
222	0	0	1	0	0	0	0.009	0	0	1	0	0.00%	99.11%
223	0	1	0	0	0	0	0.001	0	1	0	0	0.01%	99.89%
224	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
225	0	0	1	0	0	0	0.008	0	0	1	0	0.00%	99.20%

Proj. IDs	No intersection	Classic	Bicho contained	CVS. contained	perfect	Both empty	Shered bug coverage	Scen. 1	Scen. 2	Scen. 3	Scen. 4	Bicho Delta	CVS. Delta
226	0	1	0	0	0	0	0.193	0	1	0	0	55.61%	25.13%
227	0	1	0	0	0	0	0.019	0	1	0	0	28.85%	69.23%
228	0	1	0	0	0	0	0.010	0	1	0	0	0.16%	98.82%
229	0	0	1	0	0	0	0.048	0	0	1	0	0.00%	95.15%
230	0	1	0	0	0	0	0.015	0	1	0	0	19.31%	79.21%
231	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
232	0	0	1	0	0	0	0.115	0	0	1	0	0.00%	88.50%
233	0	1	0	0	0	0	0.089	0	1	0	0	1.56%	89.58%
234	0	0	1	0	0	0	0.005	0	0	1	0	0.00%	99.52%
235	0	0	1	0	0	0	0.004	0	0	1	0	0.00%	99.63%
236	0	1	0	0	0	0	0.012	0	1	0	0	68.90%	29.88%
237	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
238	0	0	1	0	0	0	0.044	0	0	1	0	0.00%	95.57%
239	1	0	0	0	0	0	0.000	1	0	0	0	94.74%	5.26%
240	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
241	0	1	0	0	0	0	0.001	0	1	0	0	0.09%	99.82%
242	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
243	0	1	0	0	0	0	0.228	0	1	0	0	23.16%	54.04%
244	0	1	0	0	0	0	0.041	0	1	0	0	0.34%	95.53%
245	0	1	0	0	0	0	0.321	0	1	0	0	13.08%	54.87%
246	0	1	0	0	0	0	0.002	0	1	0	0	0.25%	99.51%
247	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
248	0	1	0	0	0	0	0.063	0	1	0	0	69.76%	23.90%
249	0	1	0	0	0	0	0.024	0	1	0	0	0.16%	97.49%
250	0	1	0	0	0	0	0.045	0	1	0	0	0.32%	95.22%
251	0	1	0	0	0	0	0.010	0	1	0	0	0.26%	98.71%
252	0	1	0	0	0	0	0.025	0	1	0	0	7.50%	90.00%
253	0	1	0	0	0	0	0.097	0	1	0	0	85.67%	4.67%
254	0	1	0	0	0	0	0.188	0	1	0	0	58.50%	22.72%
255	0	1	0	0	0	0	0.001	0	1	0	0	0.06%	99.89%
256	0	0	1	0	0	0	0.026	0	0	1	0	0.00%	97.37%
257	0	0	1	0	0	0	0.056	0	0	1	0	0.00%	94.40%
258	0	0	1	0	0	0	0.074	0	0	1	0	0.00%	92.56%
259	0	0	1	0	0	0	0.084	0	0	1	0	0.00%	91.60%
260	0	1	0	0	0	0	0.046	0	1	0	0	5.56%	89.81%
261	0	1	0	0	0	0	0.436	0	1	0	0	45.00%	11.43%
262	0	1	0	0	0	0	0.140	0	1	0	0	9.89%	76.13%
263	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
264	0	1	0	0	0	0	0.032	0	1	0	0	67.74%	29.03%
265	0	1	0	0	0	0	0.015	0	1	0	0	14.45%	84.03%
266	0	1	0	0	0	0	0.049	0	1	0	0	32.79%	62.30%
267	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
268	0	0	1	0	0	0	0.006	0	0	1	0	0.00%	99.41%
269	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
270	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
271	1	0	0	0	0	1	0.000	0	0	0	1	0.00%	0.00%
272	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
273	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
274	1	0	0	1	0	0	0.000	0	0	1	0	100.00%	0.00%
275	0	0	1	0	0	0	0.068	0	0	1	0	0.00%	93.19%
276	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
277	0	1	0	0	0	0	0.077	0	1	0	0	12.82%	79.49%
278	0	0	1	0	0	0	0.005	0	0	1	0	0.00%	99.53%
279	0	1	0	0	0	0	0.194	0	1	0	0	13.89%	66.67%

Proj. IDs	No intersection	Classic	Bicho contained	CVS. contained	perfect	Both empty	Shered bug coverage	Scen. 1	Scen. 2	Scen. 3	Scen. 4	Bicho Delta	CVS. Delta
280	0	0	1	0	0	0	0.013	0	0	1	0	0.00%	98.67%
281	0	1	0	0	0	0	0.351	0	1	0	0	0.66%	64.24%
282	0	1	0	0	0	0	0.014	0	1	0	0	9.73%	88.84%
283	0	1	0	0	0	0	0.028	0	1	0	0	2.04%	95.12%
284	0	1	0	0	0	0	0.033	0	1	0	0	0.95%	95.72%
285	0	0	1	0	0	0	0.036	0	0	1	0	0.00%	96.39%
286	0	1	0	0	0	0	0.029	0	1	0	0	0.78%	96.35%
287	0	1	0	0	0	0	0.083	0	1	0	0	12.50%	79.23%
288	0	1	0	0	0	0	0.004	0	1	0	0	1.45%	98.11%
289	0	1	0	0	0	0	0.170	0	1	0	0	1.50%	81.50%
290	0	0	1	0	0	0	0.036	0	0	1	0	0.00%	96.39%
291	0	1	0	0	0	0	0.107	0	1	0	0	0.48%	88.78%
292	0	1	0	0	0	0	0.035	0	1	0	0	0.61%	95.90%
293	0	0	1	0	0	0	0.043	0	0	1	0	0.00%	95.74%
294	0	1	0	0	0	0	0.016	0	1	0	0	87.17%	11.26%
295	0	1	0	0	0	0	0.168	0	1	0	0	0.50%	82.67%
296	0	1	0	0	0	0	0.049	0	1	0	0	0.11%	94.99%
297	0	1	0	0	0	0	0.019	0	1	0	0	66.43%	31.63%
298	0	1	0	0	0	0	0.060	0	1	0	0	1.00%	93.00%
299	0	0	1	0	0	0	0.091	0	0	1	0	0.00%	90.91%
300	0	1	0	0	0	0	0.104	0	1	0	0	0.35%	89.27%
301	0	1	0	0	0	0	0.115	0	1	0	0	2.24%	86.23%
302	0	1	0	0	0	0	0.135	0	1	0	0	0.33%	86.17%
303	0	1	0	0	0	0	0.096	0	1	0	0	0.36%	90.02%
304	0	1	0	0	0	0	0.141	0	1	0	0	1.79%	84.12%
305	0	1	0	0	0	0	0.063	0	1	0	0	20.00%	73.75%
306	0	1	0	0	0	0	0.015	0	1	0	0	1.28%	97.19%
307	0	1	0	0	0	0	0.017	0	1	0	0	8.62%	89.66%
308	0	1	0	0	0	0	0.052	0	1	0	0	1.04%	93.78%
309	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
310	0	1	0	0	0	0	0.005	0	1	0	0	23.43%	76.09%
311	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
312	0	1	0	0	0	0	0.034	0	1	0	0	6.19%	90.38%
313	1	0	0	0	0	0	0.000	1	0	0	0	60.83%	39.17%
314	0	1	0	0	0	0	0.278	0	1	0	0	1.11%	71.11%
315	0	1	0	0	0	0	0.234	0	1	0	0	0.43%	76.17%
316	0	1	0	0	0	0	0.128	0	1	0	0	1.60%	85.58%
317	0	1	0	0	0	0	0.120	0	1	0	0	1.56%	86.48%
318	0	0	1	0	0	0	0.014	0	0	1	0	0.00%	98.65%
319	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
320	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
321	0	0	1	0	0	0	0.099	0	0	1	0	0.00%	90.13%
322	0	1	0	0	0	0	0.239	0	1	0	0	0.57%	75.57%
323	0	0	1	0	0	0	0.016	0	0	1	0	0.00%	98.44%
324	0	1	0	0	0	0	0.038	0	1	0	0	11.32%	84.91%
325	0	1	0	0	0	0	0.161	0	1	0	0	0.86%	83.08%
326	0	0	1	0	0	0	0.056	0	0	1	0	0.00%	94.44%
327	0	1	0	0	0	0	0.058	0	1	0	0	0.31%	93.85%
328	0	1	0	0	0	0	0.151	0	1	0	0	77.36%	7.55%
329	0	1	0	0	0	0	0.007	0	1	0	0	0.70%	98.60%
330	0	1	0	0	0	0	0.028	0	1	0	0	30.56%	66.67%
331	0	1	0	0	0	0	0.012	0	1	0	0	0.61%	98.18%
332	0	1	0	0	0	0	0.213	0	1	0	0	0.75%	77.99%
333	0	1	0	0	0	0	0.054	0	1	0	0	6.78%	87.85%
334	0	1	0	0	0	0	0.239	0	1	0	0	35.60%	40.50%
335	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
336	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
337	1	0	1	0	0	0	0.000	0	0	1	0	0.00%	100.00%
338	0	1	0	0	0	0	0.153	0	1	0	0	16.13%	68.55%
339	0	1	0	0	0	0	0.114	0	1	0	0	11.96%	76.63%
340	0	1	0	0	0	0	0.057	0	1	0	0	0.30%	94.03%
341	0	1	0	0	0	0	0.036	0	1	0	0	1.81%	94.58%
342	0	1	0	0	0	0	0.221	0	1	0	0	7.18%	70.72%
343	0	1	0	0	0	0	0.100	0	1	0	0	0.16%	89.86%
344	0	1	0	0	0	0	0.104	0	1	0	0	2.08%	87.50%
	106	182	102	26	0	8		25	182	129	8		
			128					No. OSS Projects in four scenario					