2016 IEEE/ACM 13th Working Conference on Mining Software Repositories

Multi-extract and Multi-level Dataset of Mozilla Issue Tracking History

Jiaxin Zhu, Minghui Zhou*, Hong Mei
School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education, China
zhujiaxin, zhmh, meih@pku.edu.cn

ABSTRACT

Many studies analyze issue tracking repositories to understand and support software development. To facilitate the analyses, we share a Mozilla issue tracking dataset covering a 15-year history. The dataset includes three extracts and multiple levels for each extract. The three extracts were retrieved through two channels, a front-end (web user interface (UI)), and a back-end (official database dump) of Mozilla Bugzilla at three different times. The variations (dynamics) among extracts provide space for researchers to reproduce and validate their studies, while revealing potential opportunities for studies that otherwise could not be conducted. We provide different data levels for each extract ranging from raw data to standardized data as well as to the calculated data level for targeting specific research questions. Data retrieving and processing scripts related to each data level are offered too. By employing the multi-level structure, analysts can more efficiently start an inquiry from the standardized level and easily trace the data chain when necessary (e.g., to verify if a phenomenon reflected by the data is an actual event). We applied this dataset to several published studies and intend to expand the multi-level and multi-extract feature to other software engineering datasets.

1. INTRODUCTION

Researchers and practitioners investigate various software engineering topics by analyzing issue tracking repositories, such as predicting [6], locating [3], triaging [10], and resolving defects [1]. Shared robust datasets have been pursued to reduce duplicated efforts on collecting and preprocessing data and build benchmarks to support reproductions and comparisons of previous studies [4]. In this paper, we provide a dataset of a 15-year Mozilla issue tracking history, retrieved from Mozilla Bugzilla[1].

Several attempts have been previously made in sharing data extracted from issue tracking systems, e.g., the Eclipse and Mozilla defect tracking dataset [7], the Firefox temporal defect dataset [2], the Eclipse and Mozilla Bugzilla dataset of MSR Mining Challenge 2007[2]. Targeting predefined questions, these published datasets have been carefully curated with filtering, cleaning, transforming, and so on to simplify the work for prospective data users. However, information that is important to users may get discarded in the process. For example, issue activities (see Section 2.1) are not included in the MSR 2007 dataset[2], therefore users who would like to investigate the issue resolution history will not be satisfied. Moreover, these datasets are often released without raw and intermediate data, which makes it difficult to inspect data issues in further analyses. For instance, if the number of newly filed defects per month is counted, such as in [7], and outliers are observed, it may be difficult to determine if something unusual actually happened, or if the defect filtering is questionable.

In our dataset, therefore, we constructed multiple data levels ranging from raw data and standardized data to calculated data together with all data retrieving and processing scripts. Raw and standardized levels maintain all issue tracking information for the convenience of wider applications, while lower levels and scripts enable inspection of exceptions through backtracking.

Issue tracking data always change with further development. Furthermore, published significant relationships tend to become less significant or disappear once more data are collected [5]. To address this problem, we incorporated multiple pieces of data (namely data extracts) collected through different channels at different times. In particular, two extracts (Extract 2011 and Extract 2012) were retrieved via the front-end interface of Bugzilla in March 2011 and April 2012. In addition, one extract (Extract 2013) was retrieved through the back-end database in January 2013 (offered by the Mozilla development team[3]). The newer extract includes recently reported issues and updates of previous issues. The issues that were not open in the previous extract on account of privacy or security concerns may become available in a later extract. It is worth noting that the Bugzilla settings may differ across different extracts (e.g., modification of a summary of Bug 216714 at 2006-11-14 10:12:08 is split into five records in Extract 2013; however, it is recorded as one

*Corresponding author.
[1]https://bugzilla.mozilla.org/

MSR'16, May 14-15, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2901739.2903502

[2]http://msr.uwaterloo.ca/msr2007/challenge/
[3]The process is recorded at https://bugzilla.mozilla.org/show_bug.cgi?id=838947. At the time of this publication, researchers are required to sign up a nondisclosure agreement to obtain the Mozilla held data, and may not disclose the data to anyone without Mozilla's express written permission.

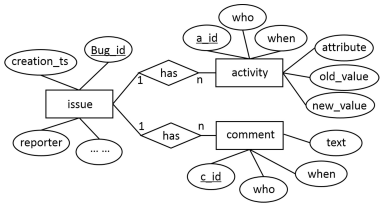


Figure 1: Dataset model.

in Extract 2011). Thus, they require unification for comparison. Data users can utilize the different extracts to validate the robustness of their analytic constructs and results. The source and impact of differences among extracts can also be independently investigated. All the data and scripts are hosted on GitHub: `https://github.com/jxshin/mzdata`. Please follow the notes in *README.md* to download the whole dataset or part of it, e.g., the Mozilla official dump.

The remainder of this paper is structured as follows. In Section 2, we describe how the dataset is retrieved, processed, and formatted. In Section 3, we discuss how the dataset can be used. We outline the limitations in Section 4 and summarize in Section 5.

2. DATA OVERVIEW

The Mozilla issue tracking dataset contains more than 774,000 issues reported from April 1998 to January 2013, including defects, feature requests, enhancements, etc. In the following subsections, we elaborate on the tracking information included in the dataset, and we describe the dataset structure and schema.

2.1 Issue Tracking Information

The Mozilla issue reports are tracked in the way outlined in Figure 1. Several attributes are defined for each issue. These include the report time (*creation_ts*); user who reported the issue (*reporter*); importance of the issue (*bug_severity*); component, product and version to which it belonged (*component*, *product*, *version*); developer to whom it was assigned for solving (*assigned_to*); status when the data was collected (*status*), processing result (*resolution*), etc. The range of the value is set for each attribute. For example, *bug_severity* can be assigned as *trivial*, *minor*, *normal*, *major*, *critical* or *blocker*. The value of an issue's attributes may change over time, in accordance with the workflow defined by the project [11]. In the meantime, users may comment on the issue. All the activities, including comments and activities that change the attribute value, illustrate how the issue is processed. Our dataset includes all the tracking information described above.

2.2 Dataset Structure and Schema

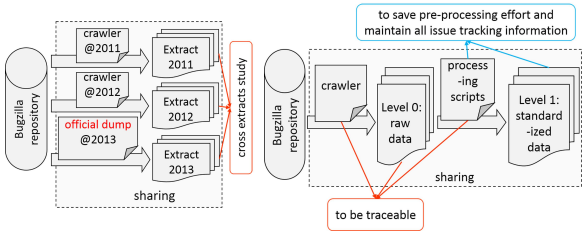
We retrieved the raw data from Mozilla Bugzilla through different channels at different times. We processed and organized the data with a structure of multiple extracts and multiple levels.

2.2.1 *Multiple Data Extracts*

Retrieving data is a challenging task with many practical complexities [4]. The conventional method of retrieving issue tracking data is to download the web pages (HTML/XML pages) with crawlers through the front-end interface of Bugzilla. Owing to the restricted access set by Bugzilla administrators and network limitations, among other constrains, such a method is often time-consuming, resource-

Table 1: Census of participation

Extract	2011	2012	2013
time span	1998.4-2011.3	1998.4-2012.4	1998.4-2013.1
issues	620448	709335	774809
reporters	143314	152776	158635
comments	5224049	6030698	6652306
commenters	178963	191680	199662
changes	6837155	8045189	9507565
operators	84377	94631	101730
participants	185662	200098	208723



a. multiple extracts b. multiple levels

Figure 2: Dataset struture.

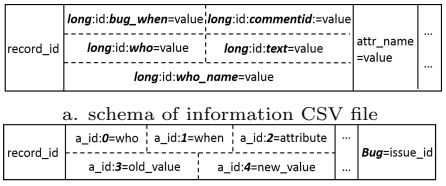
intensive and unstable. The mining software repository community may have a long-standing desire for official database dumps [4]. However, substantial efforts in sanitizing the data for legal, privacy, and security needs is often required to release the data. Furthermore, FLOSS projects simply lack the resources to handle such requests [4]. For this dataset, we successfully obtained an official dump from the Mozilla community in addition to two snapshots crawled from the web UI of Mozilla Bugzilla. The approach that we used to obtain and organize the data is depicted in Figure 2 a.

We retrieved two extracts from the web in March 2011 and April 2012, respectively. We filed an issue report to request data from Mozilla and obtained the official database dump in January 2013. This latter dump comprised the third extract. This kind of official database dump was promising in terms of revealing the differences between the front-end view and back-end database. Moreover, it enabled us to verify the quality of the data that we obtained from the front-end, etc. It should be noted that data relating to legal, privacy, and security concerns were excluded from this dump.

Table 1 presents a census of the issue tracking participation on these three extracts. The high number of issues, reporters, comments, commenters, attribute value changes, and operators demonstrates a relatively active community. In approximately 15 years, more than 774,000 issues were reported by more than 158,000 reporters. During the processing of these issues, approximately 6.7 million comments from approximately 200,000 users were posted. The difference between the three extracts show that the number of new issues and participants continued to increase. This long and active issue tracking history provided a basis for longitudinal research [8] that requires a long time span, such as studying long term contributors [11]. It also benefits the training and testing of prediction models, whereby the amount of data determines the model performance [9].

2.2.2 *Multiple Data Levels*

[4]https://bugzilla.gnome.org/show_bug.cgi?id=693295



a. schema of information CSV file

b. schema of activity CSV file

* Bold italics denote constant strings in the records. The fields in each records are unordered; the indicator before the equal sign should be used to find the target data.

Figure 3: Schema of the information and activity in CSV files.

For the convenience of traceability, the data were processed and organized with multiple levels within each extract. Figure 2 b shows two basic data levels: Level 0 holds the retrieved raw data, and Level 1 stores the standardized data for further analysis. The Level 0 data of Extract 2011 and 2013 were original Bugzilla web pages, including the main page, which contains the attributes and comments of the issues, as well as the history page, which records the modifications of issue attribute values. Level 0 of Extract 2013 is the original dump of the Bugzilla database, which stores all the data presented in the main and history page. To ease the data parsing, we standardized the raw data in Level 1 using the CSV format. This presentation simply uses separators and line breaks to store the fields and records, making it easy to be parsed and read by simple scripts. We used HTML/XML parsers and SQL queries to extract the fields from the web pages and database dump in Level 0, and we unified and stored them in the CSV files in Level 1.

Corresponding to the main page and history page of the issues, we built two kinds of CSV files, information files and activity files. The schema of them is shown in Figure 3. Based on these standardized data, higher data levels could be constructed for specific purposes. For example, non-defect issues could be filtered to obtain Level 2, and measures on defects could be calculated to obtain Level 3.

3. DATASET RELEVANCE

3.1 Target Problems

Being one of the most important software repositories, issue tracking systems have been widely mined to improve issue tracking practices [10].

"Who can fix this bug?" is an important question in bug triage needed to "accurately" assign developers to bug reports. Detecting duplicate reports is another common problem, with studies attempting various ways to compare the similarity of issue reports. Bug fixing is an important topic having a substantial amount of literature. Common questions include: which bugs get fixed? where the bug is? how to fix it? how long the fix takes? Other related topics include how the code review (through issue tracking systems) is conducted, how to measure individual (or team) effort and productivity, etc. Understanding the participation in issue workflow is a unconventional topic involving social factors like motivation and organization. Questions like what motivate the participants, what kind of tasks they do, how well they do, were explored to guide effective participation [10].

Published issue tracking datasets typically have different focuses. For example, the dataset provided in [7] was constructed for studying the updates that a reported defect experiences. Such datasets are carefully curated for the convenience of solving those questions. However, the convenience is accompanied by information loss for other users. For instance, the comments are filtered in the dataset of [7]; i.e., the participation information is incomplete, and the activeness of a participant can not be appropriately measured. Maintaining all the issue tracking information retrieved from Bugzilla, our dataset has a better extensibility for any of the summarized issue tracking problems.

Moreover, with the characters of multi-extract and multi-level, the dataset we provide could help to answer future questions on data quality and data dynamics as discussed in Section 3.2.

3.2 Benefits of Multiple Extracts and Levels

The benefit of having multiple extracts is significant. First, it provides a better understanding about the quality of the data that we retrieved from the Internet (compared to the official dump), which is not well-known knowledge in the community. For example, we found that the attributes of some (the same) issues differed across the two extracts retrieved from the Internet (Extract 2011 & 2012) [11]. A further investigation revealed that the inconsistencies were caused by the unstable downloading. For example, retrieving a snapshot of Mozilla Bugzilla required two weeks, and the network could not be consistently stable. The database dump, on the other hand, should not have these problems.

Second, it offers an opportunity for evaluating the dynamics that exist in the different extracts. Through a preliminary comparison across the three extracts, we observed several dynamics. 1) The newer extract includes recently reported issues and updated activities on issues that existed in the earlier extracts. For example, users may have added comments on an issue, or assigned an issue to another user (therefore, its status is changed from *new* to *assigned*). 2) The issue attributes and ranges of their values may have changed on account of the evolution of the product and community. For example, new attributes of *Tracking Flags* could have been added for tracking the resolution process in multiple versions of the product, and the *product* could have had new values when the new products were developed, 3) Users may have changed their Bugzilla logins. The login is the registered email of a Bugzilla user for identification. It is not fixed, i.e., the email can be changed by the user. Whether a measurement is sensitive to these dynamics [5] can be explored through comparing across different extracts.

Third, multiple extracts make the reproducibility feasible. Users can easily validate their results on a different extract, and different studies can be compared on the same dataset (multiple times). An example presented in Section 3.3 shows how we leverage this benefit in the studies.

The benefit of multi-level organization is that it enables users to trace the data level by level during analysis. Through tracing it back to the crawled web pages, we can locate missing records or questionable data. For example, in the analysis of [11], when locating participants we found 17 emails in Extract 2011 that were all mentioned in Bug 452498 but were missing from Extract 2012. A further investigation revealed that the inconsistencies were caused by retrieving the issue report. In Extract 2012, all the activities related to Bug 452498 had only a login instead of the full email address (e.g., instead of *shaver@mozilla.org*, only *shaver* was returned). This suggests that the retrieval script

had not successfully logged into Bugzilla before retrieving the issue (Bugzilla reports for unauthenticated users do not include a full email address).

In addition, our dataset may provide opportunities for users to address questions that have rarely been investigated. For example, the majority of republished issues in newer extracts are security sensitive issues that should be made private before they are resolved. It is interesting to characterize them to support early identification when triaging newly reported issues.

3.3 Research Example

We have applied this dataset to several published studies, e.g., [10, 11]. In [11] we measured and predicted how the newcomers' involvement and environment in her first month in the issue tracking system affect their odds of becoming a long term contributor (LTC). We constructed nine metrics (predictors) based on newcomers' issue tracking activities, e.g., whether or not she could get at least one issue that she reported in her first month to be fixed (*gotFixed*). From the standardized data of Level 1, we extracted all the issue tracking activities and formed a dataset of Level 2 in which each record is an activity (who does what at when on which issue). Based on that, we calculated the metrics for each newcomer. For example, we looked for the issues reported by a newcomer from the Level 2 data, checked whether the issues were *resolved*, and assigned *True* or *False* to her *gotFixed*. We then had a dataset of Level 3, where each observation is a newcomer with her calculated metrics. Using this Level 3 data as input, we fitted a logistic regression model to explain the chance of a newcomer becoming an LTC in R[5]. We used Extract 2011 to fit the model, and used Extract 2012 to test whether it involved overfitting. The analysis was furthermore reproduced on Extract 2013. We inspected the model coefficients and p-values to determine if the significant relationships were maintained. Please read our paper and visit our website[6] for the details.

4. USER CHALLENGES

Despite the above benefits, users face challenges in employing this dataset for their purposes. The main challenge is addressing differences among the extracts. The format of Level 0 data retrieved from the web (Extract 2011/2012) is totally different from that of the official dump (Extract 2013). Some attribute names are different too. However, in Level 1, all the data are presented with unified schema and names. It is easier to start from Level 1; nevertheless, it is important to be able to trace the data back to Level 0 to locate the problematic data when needed.

Bugzilla users can change their logins at any time; therefore, another challenge is to cope with the inconsistency of user logins when locating the same participant among multiple data extracts. The logins of the same user can be grouped with the heuristic method that collects the logins involved in the same activity among the three extracts; i.e., the change of value of the same attribute of the same issue at the given time.

5. SUMMARY

The sharing of data reduces efforts required for data collection, while promoting the replication and comparison of study findings. In this paper, we proposed a dataset with a multi-extract and multi-level structure. It provides more efficient and effective support on data dynamics and data traceability. It thereby helps users to better understand the data they are using, while improving the validation of their analytic constructs and results. In future work, we intend to build a multi-level and multi-extract data sharing framework and apply it to other software engineering data.

[5]www.r-project.org/

[6]https://passion-lab.org/projects/developerfluency.html

6. ACKNOWLEDGMENTS

This work is supported by the National Basic Research Program of China Grant 2015CB352203 and the National Natural Science Foundation of China Grants 61432001 and 61421091.

7. REFERENCES

[1] O. Baysal, R. Holmes, and M. W. Godfrey. Revisiting bug triage and resolution practices. In *User Evaluation for Software Engineering Researchers (USER), 2012*, pages 29–30. IEEE, 2012.

[2] M. Habayeb, A. Miranskyy, S. S. Murtaza, L. Buchanan, and A. Bener. The firefox temporal defect dataset. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 498–501. IEEE Press, 2015.

[3] E. Hill, S. Rao, and A. Kak. On the use of stemming for concern location and bug localization in java. In *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on*, pages 184–193. IEEE, 2012.

[4] Howison, J. Conklin, M. Crowston, and Kevin. Flossmole: A collaborative repository for floss research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3):17–26, 2008.

[5] J. P. Ioannidis. Why most published research findings are false. *Chance*, 18(4):40–47, 2005.

[6] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan. Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering*, pages 1–35, 2015.

[7] A. Lamkanfi, J. Pérez, and S. Demeyer. The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 203–206. IEEE Press, 2013.

[8] A. M. Pettigrew. Longitudinal field research on change: Theory and practice. *Organization Science*, 1(3):267–292, 1990.

[9] V. Popovici, W. Chen, B. G. Gallas, C. Hatzis, W. Shi, F. W. Samuelson, Y. Nikolsky, M. Tsyganova, A. Ishkin, and T. Nikolskaya. Effect of training-sample size and classification difficulty on the accuracy of genomic predictors. *Breast Cancer Research Bcr*, 12(1):R5, 2010.

[10] J. Xie, M. Zhou, and A. Mockus. Impact of triage: a study of mozilla and gnome. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pages 247–250. IEEE, 2013.

[11] M. Zhou and A. Mockus. Who will stay in the floss community? modeling participant's initial behavior. *Software Engineering, IEEE Transactions on*, 41(1):82–99, 2015.