# The Eclipse and Mozilla Defect Tracking Dataset: A Genuine Dataset for Mining Bug Information

Ahmed Lamkanfi, Javier Pérez, Serge Demeyer
University of Antwerp, Belgium
(Ahmed.Lamkanfi, Javier.Perez, Serge.Demeyer)@ua.ac.be

***Abstract*—The analysis of bug reports is an important subfield within the mining software repositories community. It explores the rich data available in defect tracking systems to uncover interesting and actionable information about the bug triaging process. While bug data is readily accessible from systems like Bugzilla and JIRA, a common database schema and a curated dataset could significantly enhance future research because it allows for easier replication. Consequently, in this paper we propose the Eclipse and Mozilla Defect Tracking Dataset, a representative database of bug data, filtered to contain only genuine defects (*i.e.,* no feature requests) and designed to cover the whole bug-triage life cycle (*i.e.,* store all intermediate actions). We have used this dataset ourselves for predicting bug severity, for studying bug-fixing time and for identifying erroneously assigned components. Sharing these data with the rest of the community will allow for reproducibility, validation and comparison of the results obtained in bug-report analyses and experiments.**

***Index Terms*—Bug reports, Defect Tracking, Dataset**

## I. Introduction

*Replication is not supported, industrial cases are rare, [. . . ]. In order to help the discipline mature, we think that more systematic empirical evaluation is needed.* [1]

The mining software repositories community has a long-standing desire for sharing robust and genuine datasets, mainly because the field explicitly aims to process these large volumes of information automatically [2]. As illustrated by the parody "Failure is a Four-Letter Word"' (*i.e.,* the key-strokes I-R-O-P are likely to induce defects), if one throws sufficient statistics at a given dataset, one will always find a correlation [3]. Creating shared datasets will not eradicate such deception, but it will at least make them less likely and easier to expose.

The analysis of bug reports in particular is a sub-field ripe for creating such a shared dataset. Indeed, over the past decade there have been numerous attempts at mining bug reports to uncover interesting and actionable information about the bug triaging process [4, 5, 6, 7].

In this data paper, we focus on the updates or changes that a reported bug experiences throughout its lifetime. Indeed, the information provided in a bug report is fairly technical in nature and reporters may provide incorrect information in the report [8]. Instead of only considering a single snapshot of a report, we believe more interesting findings can be mined from bug report data when the whole lifetime is investigated. The main contribution of the *Eclipse and Mozilla Defect Tracking Dataset* is the provided fine-grained update information of the bug reports originating from interesting open-source projects, namely Eclipse and Mozilla.

This paper is structured as follows. In section II, we describe the structure of the dataset and how it is formatted. Section III presents some preliminary insights on the provided dataset by performing a qualitative data analysis. Next, we briefly describe in Section IV the methodology used to obtain, preprocess and analyze the data. Subsequently, we discuss the relevance of this dataset in Section V. In Section VI, we present the challenges and limitations as introduced by our dataset. To conclude, Section VII contains a brief summary of this paper.

## II. Description

The *Eclipse and Mozilla Defect Tracking Dataset* presented in this paper contains a set of bug reports of four popular products of both Eclipse and Mozilla. Moreover, it also covers the complete lifetime of each bug report: all the updates of each considered bug report attribute are also included. The dataset can be obtained from the following link: `https://github.com/ansymo/msr2013-bug_dataset`.

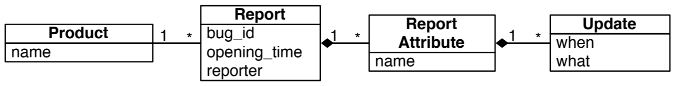
In the case of Eclipse, the following four products are considered: Eclipse Platform, JDT, CDT and PDE. Similarly, we provide the reported bugs for the following products of Mozilla: Core, Firefox, Thunderbird and Bugzilla. For each product, the dataset provides the resolved reported bugs together with the corresponding report attributes as shows in Figure 1. The attributes (instances of *Report Attribute* in the model) that are considered in this dataset are the following:

1) ***priority***: The priority denotes how soon the bug should be fixed. This attribute typically varies between *P1* to *P5* where *P1* denotes the highest priority.
2) ***severity***: the impact of the bug on the software system. This attribute varies between *trivial*, *minor*, *normal*, *major*, *critical* and *blocker*.
3) ***product***: the particular software application the bug is related to.
4) ***component***: the relevant subsystem of the product for the reported bug.
5) ***bug_status***: The attribute indicates the current state of a bug. The value of this attribute varies between *unconfirmed*, *new*, *assigned*, *reopened*, *ready*, *resolved*, *verified*.

6) ***resolution***: This attribute indicates what happened to this bug. The value of this attribute varies between *fixed*, *invalid*, *wontfix*, *duplicate*, *worksforme*, *incomplete*.
7) ***assigned_to***: the identifier of the developer who got assigned the bug.
8) ***cc***: Users who are interested in the progress of this bug.
9) ***short_desc***: a one-line summary describing the bug.
10) ***version***: the version of the product the bug was found in.
11) ***op_sys***: the operating system against which the bug is reported.

For each attribute, we compiled a list of updates denoting the different "changes" that have been performed during the entire lifetime of the respective report. For each such change, the timestamp is included of when the attribute was updated (*when*) next to the new value of the attribute of the bug report (*what*). Furthermore, there are two report attributes which are unchangeable, *i.e.,* the *reporter* and *opening_time* of the reported bug. These attributes can be found for each bug report in the instances of *Report*.

Fig. 1: Model of the dataset



In regards of its format, the *Eclipse and Mozilla Defect Tracking Dataset* has been packaged as a bundle of XML files (see Figure 2). Primarily, the two projects Eclipse and Mozilla are organized in separate directories. Next, for each product of the respective projects, a separate subdirectory stores all information relevant for that particular product. Each of these subdirectories consists of a set of XML files providing the respective updates of the considered bug report attributes. This way, the full history of each considered bug attribute is stored in each separate XML file.

Furthermore, each product directory includes an additional "main" XML file (*i.e.,* reports.xml) which stores the attributes of the bugs that are unchangeable throughout the lifetime of the bug, *i.e.,* the id of the reported bug, the recorded time of when the bug was reported (*opening_time*) and the particular reporter who filed the bug (*reporter*). For denoting a moment in time (*i.e.,* the opening time and time of each update), we use the UNIX time notation[1]. In Listing 1 and 2, we show fragments of the XML files content and structure.

Listing 1: An fragment extracted from reports.xml.

```
<reports>
  ...
  <report id="47730">
    <opening_time>1070022921</opening_time>
    <reporter>4898</reporter>
  </report>
  ...
</reports>
```

[1]Unix time, or POSIX time, is defined as the number of seconds that have elapsed since midnight Coordinated Universal Time (UTC), *i.e.,* 1 January 1970.

Fig. 2: Structure of the provided dataset

```
eclipse / mozilla
 |__product_1
 |   |__reports.xml
 |   |__attribute_1.xml
 |   |__attribute_2.xml
 |   |__...
 |   |__attribute_n.xml
 |__product_2
 |__...
 |__product_n
```

Listing 2: An fragment extracted from short_desc.xml

```
<short_desc>
  ...
  <report id="280464">
    <update>
    <when>1245144375</when>
    <what>First build performance must be
     improved
        </what>
    </update>
    <update>
      <when>1245222394</when>
      <what>API tooling performance must be
         improved</what>
    <update>
  </report>
  ...
</short_desc>
```

## III. Bug Data Analysis

In this section, we provide some insights of the provided dataset. First, we select a number of products for each project, and we quantify the size of the respective products in terms of number of components and number of reported bugs. Next, we provide the numbers of how frequently each of the considered attributes are updated during the lifetime of the bug reports. Table I lists the selected software products together with the total number of reported bugs along with the number of components for each product.

TABLE I: The different products with the corresponding number of bugs and components.

| Product | No. of Bugs | No. of Components |
|---|---|---|
| Eclipse Platform | 24775 | 22 |
| Eclipse JDT | 10814 | 6 |
| Eclipse CDT | 5640 | 20 |
| Eclipse PDE | 5655 | 5 |
| Mozilla Core | 74292 | 137 |
| Mozilla Firefox | 69879 | 47 |
| Mozilla Thunderbird | 19237 | 23 |
| Mozilla Bugzilla | 4616 | 21 |

From Table I, we distinguish products with a couple of components and products with a relatively large number of components. Furthermore, we can also distinguish products according to their total number of reported bugs for the respective products. These observations are a consequence of the modularization and the popularity of the respective products.

Bug triagers analyze the incoming bug reports in order to decide which bugs will be fixed, who should do it and when it will be fixed, etc. Here, bug triagers have the possibility to update incorrect or missing information. In particular, when a triager receives a bug report with an erroneous "component" attribute, updates will occur for that attribute. Bugzilla keeps track of these updates, hence we can extract the values for each of the considered attributes of a bug report using the update information stored in the bug database. Table II reveals the total number of such reassignments that occurred for all the products, followed by the respective percentages.

TABLE II: The different attributes selected from the reports and the respective number of reassignments.

|  | % of Reassignments |  |
|---|---|---|
| **Bug Attribute** | **Eclipse** | **Mozilla** |
| *bug_status* | 100% | 100% |
| *resolution* | 100% | 100% |
| *assigned_to* | 66% | 29% |
| *cc* | 65% | 90% |
| *short_desc* | 27% | 15% |
| *component* | 18% | 17% |
| *product* | 5% | 7% |
| *severity* | 6% | 6% |
| *version* | 6% | 21% |
| *priority* | 3% | 6% |
| *op_sys* | 5% | 10% |

In Table II, we show the different attributes and their respective update frequency sorted in descending order. As we would expect, we note that *bug_status*, *resolution* are the most frequently updated attribute of a reported bug. These attributes tend to be updated frequently since these attributes keep track of which bugs are dealt with and which are still under consideration. From Table II, we can also observe that attributes like *op_sys*, *priority* and *version* are updated much less frequently.

## IV. Methodology

The original dataset we used to prepare our curated data bundle consisted of a replica of the Bugzilla bug-tracking system of Eclipse –*i.e.,* the Bugzilla database dump. In this database, all bug reports and the corresponding updates are stored for all products within the Eclipse project. With this data dump, we set up a local copy of the Bugzilla database. We are then able to perform queries on the bug reports of Eclipse and Mozilla, allowing us to easily select and extract the relevant bug reports from the database. The main tables to extract the relevant information from include the bugs, bugs_activity, components, products tables. Using the creation_ts timestamp in the bugs table, we are able to extract the bugs that were reported from January 2006 and onwards. Next, we rely on the bugs_activity table to reconstruct the lifetime of each reported bug. This table consists of a list of atomic updates performed by the bug triagers which we then glue together to form the complete lifetime of the respective bug.

Subsequently we performed some analyses on the extracted data. These analyses allowed us to obtain a better understanding of the underlying characteristics of the reported bugs. In order to focus on defects, we filtered out feature requests from the dataset. Reports with severity *enhancement* are thus not included in the dataset. The attributes selected in this dataset are chosen according to two factors: the frequency of updates for these attributes and relevancy of the attributes to other existing studies. Here, we selected the top changing attributes to be included in the dataset. According to these analyses, we are able to formulate preprocessing steps to clean and transform the data in a research friendly format.

## V. Relevance of the Dataset

The focus of the dataset is on the updates or changes that a reported bug experience, *i.e.,* the lifetime of a reported bug. Instead of only considering a single snapshot of a report, we believe more interesting findings can be mined from the data when the whole lifetime is investigated. Moreover, this dataset allows researchers to improve their studies: now researchers can make sure that the information used in their studies is restricted to the information originally provided in the reports. In some Mining Software Repositories studies on bug reports, it is not clear whether the initial values of the report attributes are used [5, 6, 9]. Indeed, when we make predictions about newly filed bugs, the information used for evaluating a predictive model should be limited to the initial information available in the reported bugs. Furthermore, by introducing the detailed updates of the attributes of a bug report, the *Eclipse Defect Tracking Dataset* allows researchers to perform more fine-grained analyses on bug reports.

The Eclipse project has been relevant for many MSR studies [5, 6, 9, 7]. The topics of the different studies including developer assignment [7], bug tossing [10], fix-time prediction [4] and many more. This system is particularly interesting because of the success of the Eclipse project, the long lifetime of the project (earliest version dates from 2001) and its open nature. Furthermore, the Eclipse project was used as the MSR Mining Challenge in 2008 demonstrating the relevancy of the dataset.

We also provide the dataset in a convenient way. We provide a set of elementary XML files which are easy to parse, import, transform, etc. Moreover, researchers can easily include or exclude bug report attributes for their respective studies since all attribute information is modularized in a single XML file for each product. Furthermore, we have preprocessed the data so that only the relevant reports remain in the dataset. This makes it particularly easy for new-coming researchers to start their investigations on the Eclipse case.

## VI. Challenges and limitations

We encountered several challenges and difficulties when processing the reported bugs into the current format. While working with the data we have also identified some limitations of this dataset. We will briefly describe them here.

### A. Challenges

To begin with, the evolution log of a reported bug is not explicitly stored in the bug database. In order to gather the whole history of the reported bugs, it is necessary to reconstruct the different updates for each considered attribute from information spread across different tables in the Bugzilla database.

Since the "product" attribute of a bug report is also subject to updates, we had to make a decision on how to group the bug reports. Indeed, the bug reports are organized according to the "product" attribute (as shown in Figure 2). In the provided dataset, the bug reports are organized according to the most up-to-date product assignment since this product could be regarded as the *correct* product for the particular reported bug.

### B. Limitations

The dataset contains the reported bugs that have been reported from 2006 and onwards. The dataset includes the bugs that have been reported until March 2011 and December 2013 for Eclipse and Mozilla respectively. Note that Bugzilla allows users to request new features, which technically do not represent real bugs. Therefore, we do not consider reports where the severity attribute is set to *enhancement* as this category is reserved for feature requests. Furthermore, Bugzilla keeps track of which reports are resolved or not. We only include reports which have been resolved; because these represent the bugs which went through the complete life-cycle.

Instead of including bug reports from all products of the Eclipse project, we rather concentrate on a subset of the four most "active" products (see Table I). However, these are relevant products that have also been investigated in several studies [4, 5, 9].

As another restriction, our dataset does not include all the bug attributes that can be found in a Bugzilla database. We have decided to focus on a subset of the attributes available in the bug reports (see Table II). Although we think we have selected all the most important and interesting attributes and the selection is wide enough, other researchers should check in advance whether the attributes they want to analyze are included in our dataset or not. Furthermore, the obtained Bugzilla database dump at our disposal does not include open security vulnerabilities. However, this would not impact the reports provided in the dataset since we only consider resolved reported bugs.

## VII. Summary

The mining software repositories community has a long-standing desire for sharing robust and genuine datasets, mainly because the field explicitly aims to process these large volumes of information automatically [2]. The analysis of reported bugs in particular is a sub-field ripe for creating such a shared dataset. Indeed, over the last decade there have been numerous attempts at mining bug reports to uncover interesting and actionable information about the bug triaging process.

In this paper, we present the *Eclipse and Mozilla Defect Tracking Data*, a dataset that provides a comprehensive information bundle on the historical evolution of the most relevant attributes from the Eclipse and Mozilla project bug reports. The focus of our dataset is, instead of providing just a latest snapshot of each reported bug, the inclusion of the complete lifetime of each attribute of the reported bugs.

## Acknowledgments

This work has been carried out in the context of a Ph.D grant and the project entitled "Change-centric Quality Assurance (CHAQ)" (project number 120028) of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

## References

[1] P. Tonella, M. Torchiano, B. Du Bois, and T. Systä, "Empirical studies in reverse engineering: state of the art and future trends," *Empirical Software Engineering*, 2007.

[2] A. E. Hassan, "The road ahead for mining software repositories," in *Proceedings of the Future of Software Maintenance (FoSM) at the 24th IEEE International Conference on Software Maintenance (ICSM)*, 2008.

[3] A. Zeller, T. Zimmermann, and C. Bird, "Failure is a four-letter word: a parody in empirical research," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011.

[4] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, 2010.

[5] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, 2011.

[6] 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*, 2008.

[7] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering*, 2004.

[8] 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, ETX 2007*, 2007.

[9] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?" in *Proceedings of the 8th International Working Conference on Mining Software Repositories*, 2011.

[10] 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*, 2009.