BugTracking: A tool to assist in the Bug Triage Process

Gema Rodriguez[1] and Jesus M. Gonzalez-Barahona[1]

[1] gerope@libresoft.es, jgb@gsyc.es
[2] GSyC/LibreSoft, Universidad Rey Juan Carlos

1 Introduction

Many efforts on how and why bugs are introduced in the software source code are underway in the software engineering research community. Software source code is affected by many changes, many of them due to failure of the software because of emergent bugs.

While a software system is being developed, software engineers use version repositories to produce and manage their code. Developers and tester report issues, which are stored in other repositories, known as issue-tracking system, when a wrong behavior or bugs are found in the systems.

Issue-tracking systems help solving these bugs, but their problem is the difficulty of distinguish the bug reports from other that are not bugs. These systems provide an interface to manage reports of maintenance activities where developers can report issues describing bug reports, features or optimizations. During the bug triage process it is difficult to distinguish bug reports from other issues; a study describes that two of five issues are misclassified [2]. This misclassification causes bias predicting bugs wher non-bug reports are taken into account.

To classify issues as bug reports or non bug reports we can use automatic classification systems as the one described in [1], but the vocabulary used in the issues could change from project to project, as well as the policy depending on the project. Consequently, data validation is recommended in the studies [2].

Linking a bug reports in a issue-tracking system and the corresponding fix-commit may be not a trivial task. Traditionally, the methods used in link recovery [5, 6] are based on text patterns or the mining of key phrases. Unfortunately, these methods can include many false negatives causing bias in data [7, 8]. Therefore other methods, such as Mlink approach, have been developed that link bug report with fixes using features in the changed source files corresponding to commit logs in addition to the traditional textual features [4]. But all of them suppose that the issues are bug reports.

In this paper, we propose a tool to display the data necessary to the developers/testers, who could decide whether the issue is a bug report or not. The developers have the best available knowledge of their system, therefore the tool will help them choosing only bug reports to be analyzed in the bug-triage process removing any bias induced by non bug reports.

2 The tool

The tool works in the browser, displaying the main characteristics to distinguish bug report from others. Developers will be responsible to classify the tickets as bug report or not, could explain his decisions in each ticket.

2.1 Architecture

This project works with Launchpad as issue-tracking system and Gerrit as code review system. The image 1 presents the architecture used in the tool, which has been developed with JavaScript, Node, JQuery and HTML5 technologies. The server side works making queries to the API of Gerrit an Launchpad, and the client side is where the user can see the information displayed. The client side interacts with the server through events. Both sides share the information required using JSON files. Furthermore, to integrate some functionalities from GitHub, we use a third-party application between GitHub and the browser.

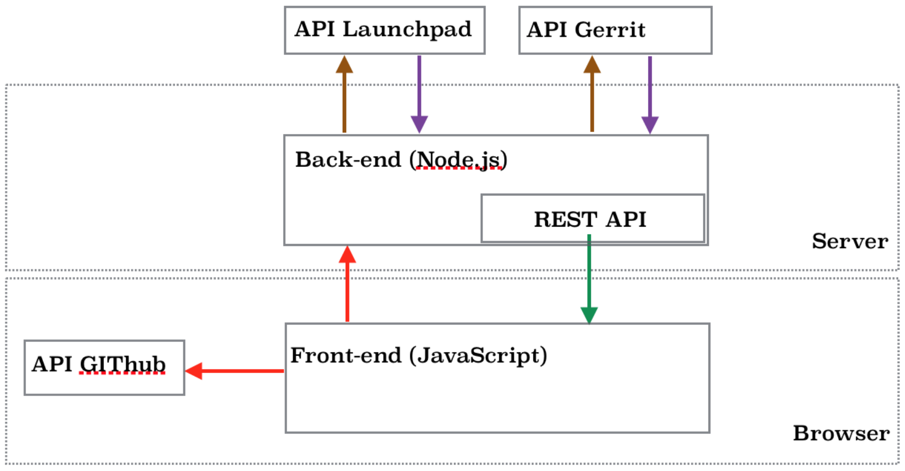


Fig. 1. Architecture of the tool.

2.2 Main Features

The tool shows the id of the tickets, which are extracted randomly from each issue-tracking repository of OpenStack, and displays the information necessary to decide whether the issue is a bug report or not. We focused in display the main parameters that help in the classification, such as the title of the report and the description as well as the description of the fix commit.

The left side in the image 2, shows the information related with the ticket in Launchpad and its correspondent fix-commit in Gerrit. Some information displayed link with the original pages in Launchpad and Gerrit, thereby the developers have at their disposal extra information such as the comments that other developers have done. Could tracking the history since the ticket opens until the commit fixed the ticket.

The right side is guided to user's opinion, after reading all the information displayed, they have to classify the ticket as *Bug report* or *Not Bug report*. Due to unsophisticated description used in the ticket, the developers could doubt in the classification, for this reason we add an extra option in the classification, *Undecided*. Furthermore, the developers have a textarea to write their opinion about their classification in each ticket, as well as other textareas to write the keywords found in the title and the description which can help us building an automatic classification system in the future.

The tool allow carry out a double bind analysis, due to the data is saved in GitHub user's account. In order that, the user can only see and modify their own data saved. GitHub is a control version system in which we have access to the whole information of each commit submitted by the developer. Thus, saving the data in GitHub, we could measure the time that each developer spend in the analysis, which tickets were more difficult to analyze and other statistics that can help us understanding the current problem of issues misclassification.

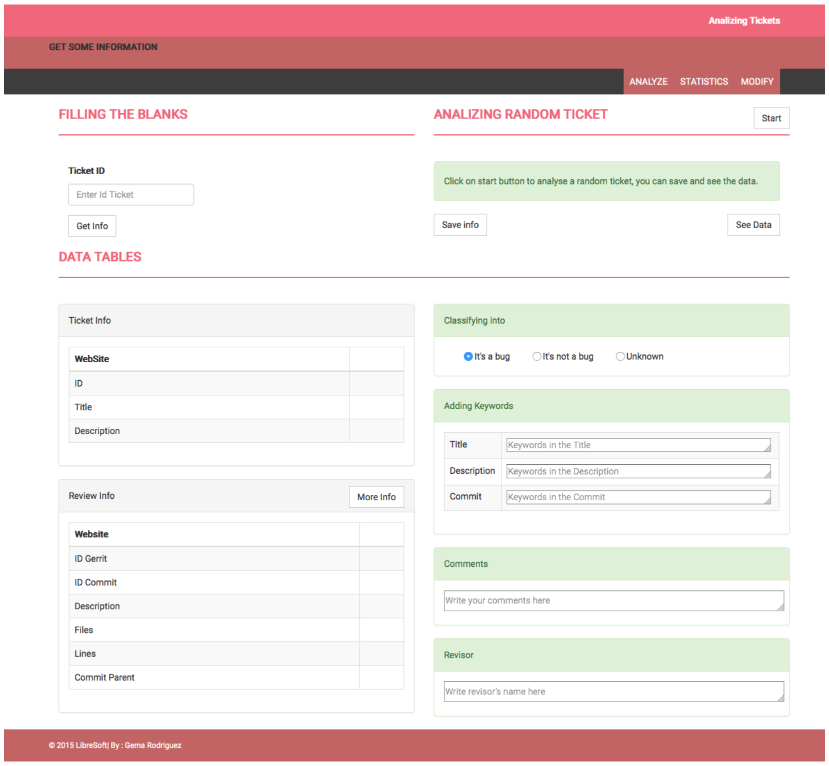
We continue developing the tool but an initial version is available[1], as well as a demonstration video[2]. It presents a licence type GPL 0 (General Public License) and you can find the code in my GitHub account[3]. Anyone can use it regardless of have GitHub account or not. But, they only can save and modify their data, in addition, one of the requirements to save and modify data is create a new repository with the same name that the repository of OpenStack you want to analyze.

3 Validation Study

OpenStack was particularly of interest because of its highest scope and heterogeneous nature with hundreds of developers contributing, furthermore due to its short life, only 5 years, all history is saved and available in a version control system. The issues are called tickets in OpenStack and available in the Launchpad, a web interface of ticket tracking system, classifying them as bug report or not.

We use the tool to analyze randomly tickets from the four principal repositories in OpenStack. This tickets could be tagged as either "Fix Commited" or "Fix Released", to be able to localize the patch implemented into de

[1] `bugtracking.libresoft.es`
[2] `https://www.youtube.com/watch?v=q0-TIvL4mqc&feature=youtu.be`
[3] `https://github.com/Gemarodri/BugTracking`



Click on start button to analyse a random ticket, you can save and see the data.

[✓ It's a bug] It's not a bug Unknown

Fig. 2. Screenshot of Index

source code in the version repository. They are generally tracked in Launchpad `Nova`,`Cinder`,`Horizon` and `Neutron`[4]

The parameters analyzed for each ticket were the title and the description of the report and the description of the fix commit. Also, the code changes if neither the descriptions and the comments clarified the underlying ticket. Each ticket was then categorized into one of three following groups.

1. The ticket describes a bug report.
2. The ticket describes a feature, an optimization code, changes in test files or other not bug reports.
3. The ticket presents a vague description and cannot be classified without doubts.

Henceforth, we will refer to Group 1 as *Bug Report*, Group 2 as *not Bug Report* and Group 3 as *Undecided*.

To validate the tool four diferent developers analyzed tickets belongs to the four main repositories. The developers analyzed more than one hundred tickets to can probe than the tool work as them expected, clasifiying the tickets

[4] `https://bugs.launchpad.net/NameOfRepository`

into one of the three groups and corroborating that the issues reports present misclassification as [2] mencioned.

4 Results

We have manually analyzed more than one hundred tickets with support of the present tool. The table 1 show the statistics of each developer after analyzed the tickets.

Table 1. Classification statistics of each developer

	Bug Report	Not Bug Report	Undecided	Total
Developer 1	53.89%	34.13%	11.98%	334
Developer 2	–%	–%	–%	125
Developer 3	–%	–%	–%	125
Developer 4	%	%	%	100

All the data is available in the github's account of the developers [5],[6][7], the repositories has the same name that the projects analyzed in OpenStack.

5 Discussion

Once we have all the tickets analyzed by diferents developers who have used a double blind, how to proceed if there are discordances between them:

1. Should they discuss after their analysis to reach a better classification?
2. Does the bug report only the same ticket classified as Bug report for all the developers?

5.1 Future Work

We would like know what grade of responsibility, none or totally, practice the previous commit in the seeding of a bug in OpenStack, considering that currently exists an implicit assumption: the line that contains the error was caused by the immediately previous commit[3]. The accuracy in our results depends on the quality of the data, thus we should focus only on bug reports discarding the other issues.

[5] `https://github.com/Gemarodri/Horizon`
[6] `https://github.com/ddalipaj/Cinder`
[7] `https://github.com/nellysek/Nova`

The next step in the tool is implement the part in where we analyzed the previous commit, displaying the code after and before the bug fixed and after and before the bug-introduction to determinate if the previous commit is responsible or not.

References

1. Antoniol, Giuliano, et al. Is it a Bug or an Enhancement? A Text-based Approach to Classify Change Requests. 2008.
2. Herzig, Kim; Just, Sascha; Zeller, Andreas. It's not a Bug, it's a Feature: How Misclassification Impacts Bug Prediction. month, 2012.
3. J. Sliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? ACM sigsoft software engineering notes, 30(4):15, 2005.
4. Nguyen, Anh Tuan, et al. 'Multi-layered approach for recovering links between bug reports and fixes." Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 2012.
5. Zimmermann, Thomas; Premraj, Rahul; Zeller, Andreas. Predicting defects for eclipse. En Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on. IEEE, 2007. p. 9-9.
6. Zimmermann, Thomas, and Peter Weigerber. "Preprocessing CVS data for fine-grained analysis." Proceedings of the International Workshop on Mining Software Repositories. 2004.
7. Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., & Devanbu, P. (2009, August). Fair and balanced?: bias in bug-fix datasets. 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 (pp. 121-130). ACM.
8. Nguyen, T. H., Adams, B., & Hassan, A. E. (2010, October). A case study of bias in bug-fix datasets. In Reverse Engineering (WCRE), 2010 17th Working Conference on (pp. 259-268). IEEE.