BugTracking: A tool to assist in the identification of bug reports

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

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

Abstract. In most software projects, but in particular in almost all free, open source software projects, issue tracking systems are used for recording many different kinds of issues: bug reports, feature requests, maintenance tickets, and even design discussions. Identifying which of those issues are bug reports, is not a trivial task. When researchers want to conduct some study on the bug reports managed by a software development project, they need to first of all to perform this identification.

The job for researchers here is very different from the bug triaging that developers do. In the latter case, people with a lot of experience in the project make a decission based on the information available at that time (maybe just a short comment by some user), asking if needed for more details. In the former case, researchers are usually not that experienced in the project, but they have at their disposal all the information produced, until the moment the issue was closed. This may include not only all comments and actions on the issue tracking system, but for examples, discussions about a fix in the code review system, or the final fixing patch in the source code management system. Having all that information conveyed to the researchers in an easy, flexible and quick way accelerates and makes much more reliable their decision making process. This simplifies large scale manual analysis of issues (in the hundreds or thousands), helping researchers to ensure that they are really working with what they intend to work: bug reports.

This paper presents a tool designed exactly to solve this problem of providing the researcher with all the relevant information needed to decide if an issue corresponds to a bug report or not. The tool uses information extracted automatically from the project repositories, and offers a web-based interface which allows for collaboration, traceability and transparency of the identification of bug reports, making the process easier, faster, and more reliable.

1 Introduction

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, where many kinds of issues can be found.

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 code 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 where non-bug reports are taken into account.

To distinguish the 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 to 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 present a tool to display all the data necessary to the developers who will 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, 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 their decissions in each ticket.

2.1 Architecture

This tool works with Launchpad as issue-tracking system and Gerrit as code review system. The image 1 presents the architecture used, 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, and interacts with the server through events. Both sides share the information required using JSON files and use their own REST API. Furthermore, to integrate some functionalities from GitHub, we use a third-party application between GitHub and the browser.

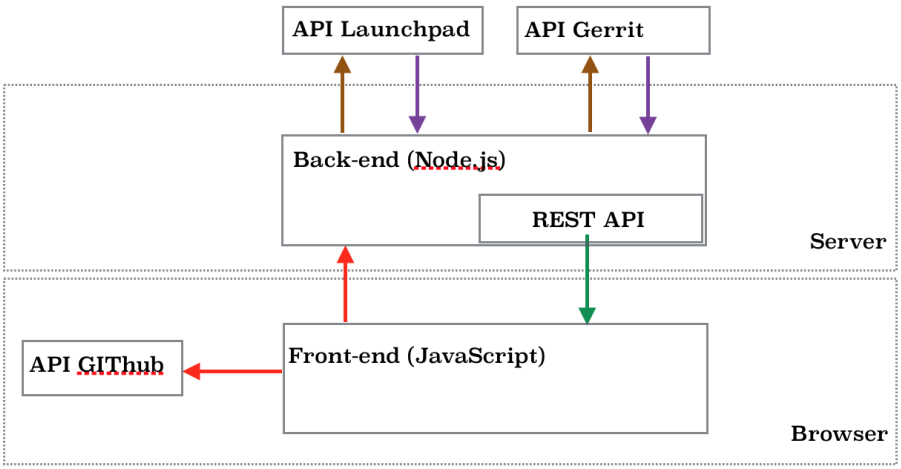

Fig. 1. Architecture of the tool.

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 review in Gerrit. Some information displayed link with the original webpages 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 a file in GitHub user's account. 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.

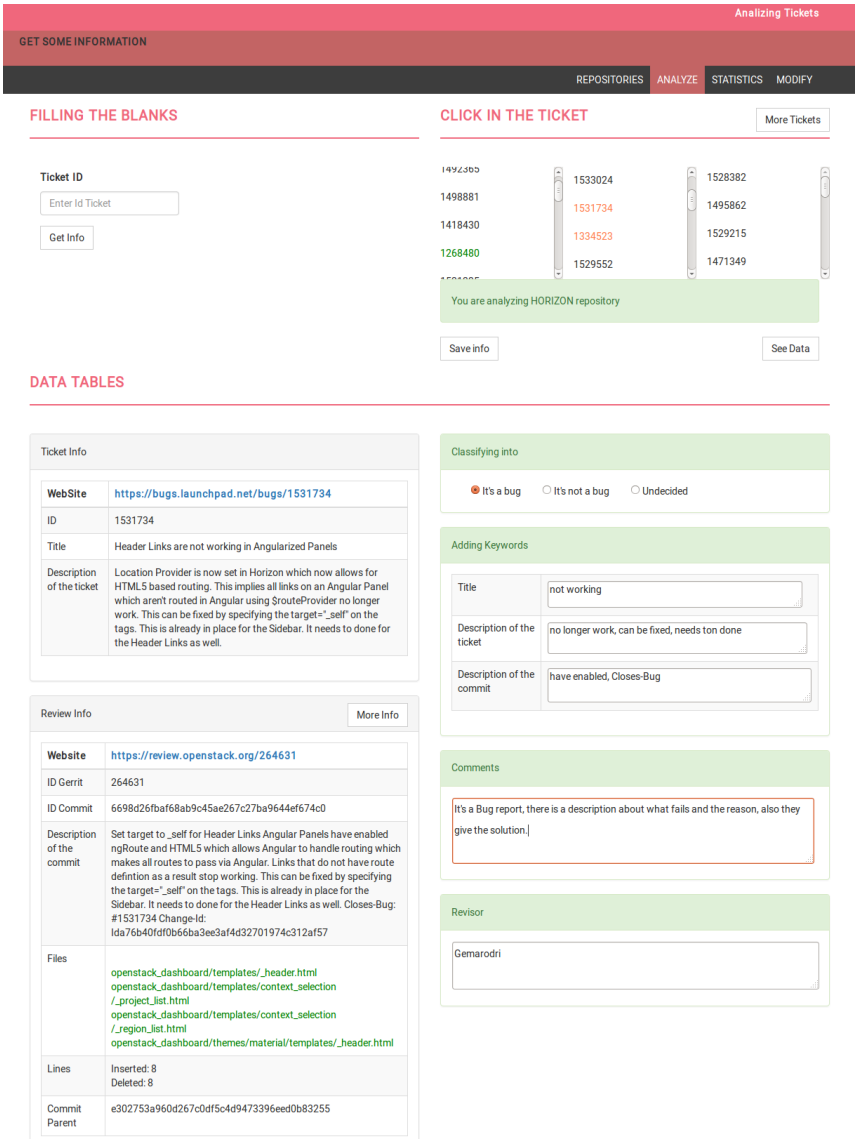

Fig. 2. Screenshot of Analyze Tab

The webpage provide us different functionalities depend on in which tab we are, next we explain these functionalities.

1. Tab Repository: In this tab we choose which repository of OpenStack we want analyze. Currently the tool supports the four principal repositories: Cinder, Nova, Neutron and Horizon, in which there is more activity.
2. Tab Analyze: Is the Tab showed in 2 where the user select/insert an identifyer of a ticket and analyze with all the data displayed if the ticket are a bug report or not.
3. Tab Statistics: This tab collects the data saved in the user's repository in GitHub after analyze each ticket, and displays a table with the number of tickets classified as *Bug Report*,*Not Bug Report* and *Undecided* in the repository and from each developer involved in the analysis, which name has to be selected previously.
4. Tab Modify: In this tab, the user can see all his data saved in the GitHub repository and modify the content of the file that he wants, in case of have inserted a mistake during the analysis.

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, the ones with GitHub account are the only ones that 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 which you want to analyze.

3 Results

We have manually analyzed 459 different tickets with support of the present tool, 125 from Cinder, 125 fron Nova, 125 from Horizon and 84 from Neutron. The table 1 show the percentage of each developer after analyzing the tickets, 417 tickets were analyzed by two different developers,

Table 1. Classification statistics of each developer

	Bug Report	Not Bug Report	Undecided	Total
Developer 1	(184) 55%	(115) 34%	(35) 11%	334
Developer 2	(188) 76%	(54) 22 %	(7) 3%	249
Developer 3	(188) 56%	(116) 35%	(30) 9%	334

The percentages between Developer 1 and Developer 3 are really similar, whereas the Developer 2 has identifyed more Bug Reports in his analysis. But,

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

the three results support the missclassification present in bug tracking systems. Futhermore, according to [2]'s work, approximately two of five issues are misclassified in the analysis of Developer 1 and Developer 3.

Focusing in the concordance between developers analyzing the same ticket, 417 tickets present a double bind review process, obtaining that each tickets was analyzed by two developers. The table 2 show the percentaje of concordance between developers in each repository after the analysis of the tickets. Some repositories do not present double bind between two of developers.

Table 2. Concordance between each developer in each repository

	Nova	Cinder	Horizon	Neutron	Total
D1 and D2	(44/63) 70%	(40/52) 77%	(37/62) 60%	-	68%
D1 and D3	-	(46/63) 73%	(48/63) 76%	(26/42) 62%	71 %
D2 and D3	(41/62) 66%	(10/10) 100%	-	-	71%

The table 2 shows that the concordance of the developers is high but, also demonstrate the difficulty to classify tickets as bug report or as not bug report, because each developer can have different ideas about a specific ticket. The concordance between the developers could be higer if they were expert in the project.

All the data is available in the repositories of github's account of the developers [4][5][6], these repositories has the same name that the projects analyzed in OpenStack.

3.1 Future Work

Future extensions of this tool includes extract information from others systems as Bugzilla or GitHub and study the misclassification in this projects. Also, display to users more information such as the status of the files affected in the fix commit after and before the bug-introducing. Futhermore, we would like to implement an auto classificator based on the semantic of the issue description and fix-commit description to help developers, showing a first idea about whereas the issue belongs to bug report or not, but the developer always has the last decission. After manually analyzing more than 400 tickets, we have seen clear cases of issues that were bug report, which have similar sentences in the description, so the automatic classification will allow developers to focus only on problematic issues, where it can be easily misclassification.

[4] `https://github.com/Gemarodri`
[5] `https://github.com/ddalipaj`
[6] `https://github.com/nellysek`

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.