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

Gema Rodríguez-Pérez, Jesús M. Gonzalez-Barahona, Gregorio Robles, Dorealda Dalipaj, and Nelson Sekitoleko

GSyC/LibreSoft, University King Juan Carlos, Fuenlabrada (Madrid),
{gerope,ddalipaj,snelson}@libresoft.com
{jgb,grex}@gsyc.com
http://libresoft.es
http://gsyc.es

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 decision 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 example, 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 their decision process much more reliable. 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.

Keywords: Issue Tracking system, Bug triage, Code review system, Tool

1 Introduction

While a software system is being developed, software engineers use version control repositories to produce and manage their code. Developers and testers report issues, which are then stored in other repositories, known as issue-tracking systems, where many kinds of issues can be found.

Issue-tracking systems faciliate the process of solving these bugs, but their shortcoming is the difficulty in distinguishing which of the reports are bug reports or not. 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 report in a issue-tracking system and the corresponding fix-commit may not be a trivial task. Traditionally, the methods used in link recovery [5, **?**] are based on text patterns or the mining of key phrases. Unfortunately, these methods can include many false negatives causing bias in data [7, **?**]. Therefore other methods, such as the 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 these methods suppose that the issues are bug reports.

In this paper, we present a tool to display all the data necessary to the researchers who will decide whether the issue is a bug report or not. The reserchers 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. (FIXME 1a: (Review)the specific, original contribution of the proposal with respect to the state of the art is hard to recognise.) (FIXME 1b: It is a bit unclear what the more specific novelty with the tool. It is understood that it integrates several data sources and presents information to the researcher in the web interface, but does it operate under some more specific "philosophy" beyond being an tool that collects and integrates information at the same place?)

2 The tool

The tool works in a browser, displaying the main characteristics to distinguish bug report from others issues. The researchers will be responsible to classify the issues from Launchpad, as bug report or not, and can thereby explain their decision for each issue. That issues is what will refer to in this paper as tickets.

2.1 Architecture

This tool works with Launchpad as issue-tracking system and Gerrit as code review system. The figure 1 presents the architecture of the tool,and based on this architecture a tool was dveloped using JavaScript, Node, JQuery and HTML5 technologies. The server side is where queries to the API of Gerrit an Launchpad are made, 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.

(FIXED: it is a bit misleading to write that "image 1 presents the architecture used, which has been developed with JavaScript, Node, JQuery and HTML5 technologies.". It is probably not the architecture that has been developed using these technologies, but rather the software is based on the architecture presented in Figure 1.)

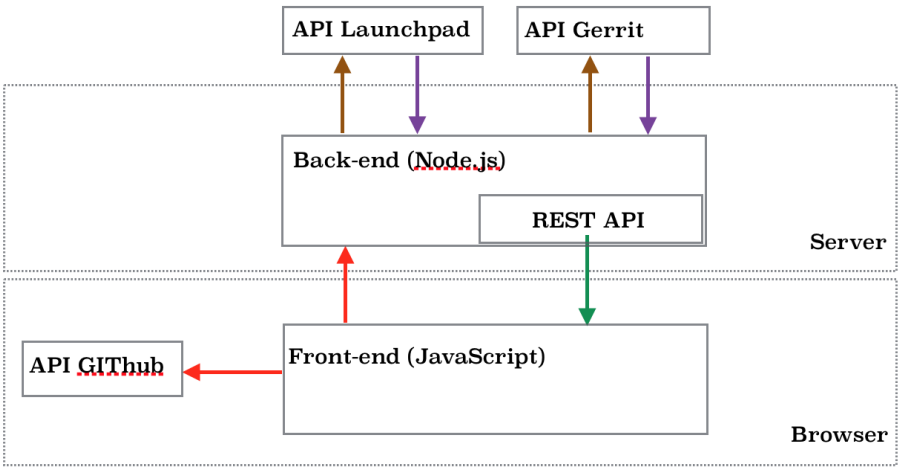

Fig. 1. Architecture of the tool.

2.2 Main Features

Figure 2 presents a screen short of the home page of the tool. Partculary 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 on displaying the main parameters that help in the classification of reports, such as the title and

description of the report, as well as the description of the fix commit. Forexample for ticket ID 1531734 the tool displays information related with the ticket in Launchpad and its corresponding review in Gerrit. (FIXED: The two subsequent sentences are very strange and hard to understand, should really be rephrased for clarity.)

In addition researchers can access original Launchapd and Gerrit web pages of specific tickets under analysis through the links displayed in tool, thereby by having extra information such as the comments written by code review developers that correspond to that partcular ticket. This way, they could track the history of the ticket from when it was open until it was closed.

The tool further facilitates researchers to record and express their opinion about the ticket after reading all the information that was automically displayed. They have to classify the ticket as *Bug report* or *Not Bug report*. Due to unsophisticated description used in the ticket, the researchers could doubt in the classification, for this reason we add an extra option in the classification, *Undecided*. Furthermore, the researchers have a text area to write keywords found in the title, and in the description of the ticket and commit message that support their classification. Finally the they can leave their comment on why they classfied a report as bug, not bug or undecided. Such information will help us building an automatic bug classification system in the future.

The tool can also allow to carry out a blind analysis of the tickets, meaning that the analysis can be done by two or more reserchers in parallel since all analysis data about tickets can is saved in a file in their GitHub accounts [Add a reference for GIT here]. By saving the data in GitHub, we could measure the time that each developer spends in the analysis, which tickets were more difficult to analyze and other metrics that can help us in understanding the current problem of issue misclassification.

The web page provides different functionalities depending on in which tab the researcher is browsing. 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.
2. Tab Analyze: Is the Tab showed in Figure 2 where the user selects/inserts an identifier of a ticket and analyze with all the data displayed if the ticket is a bug report or not.
3. Tab Statistics: This tab extracts the data already, analyzed by each developer involved in the analysis, from their users' accounts in GitHub. It analyzes this data to display a table with the number of tickets classified into the three categories; *Bug Report*,*Not Bug Report* and *Undecided*. To display the statistics of a developer, previously the name of developer has to be selected.
4. Tab Modify: In this tab, the user can see the data saved in his GitHub repository and modify the content of the file that he wants, in case of have inserted a mistake during the analysis.

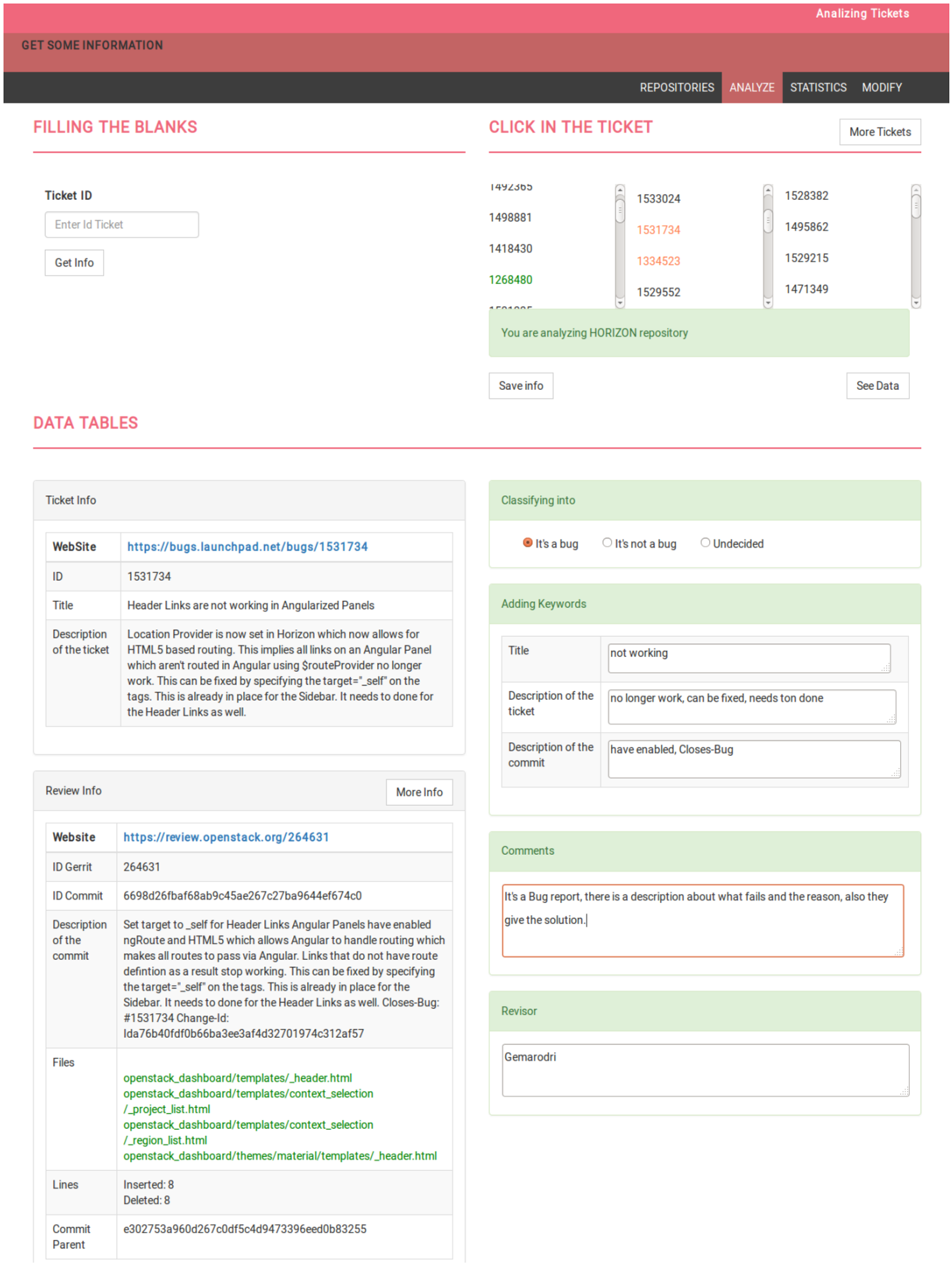

Fig. 2. Screenshot of Analyze Tab

At current state we present the initial version of the tool which is available at;[1], as well as a demonstration video[2]. The licensed under GPL 0 (General Public License) and you can find the code at a GitHub page[3]. Anyone can use the tool regardless of having GitHub account or not.However, it should be noted for the researcher to save, modify data and see statitiscs of analysed tickets automatically, he/she should create a Github repository with the same name as the Openstack project to be analysed.i.e Openstack project name: Nova, Created Github repository name: Nova

(FIXED and integrated above: Language needs to be improved in remainder of this para.) But, only the ones with GitHub account can use the GitHub available functionalities; saved the data analyzed, see the statistics and modified the data saved. The one requirement is open a repository in GitHub with the same name that the project you want to analyze in OpenStack.

3 Results

(FIXED-in future work: (Review)The test of the tool in section 3 involves three developers use of the tool in order to identify which issues are perceived as bug reports. My concern (or observation) is that it is not obvious that the results primarily are connected to the tool (but rather use of the tool), i.e. it could be claimed that the actual object(s) of study are the developers and how similarly the perceive issues being bug reports. What would the results look like if the tool was not used (but rather the data sources in isolation) or some other possible tool? I am not saying that the test/investigation is without value, but I think this should be discussed somewhere in the paper.)

We have manually analyzed 459 different tickets with support of the initial version of the tool, 125 from Cinder, 125 from Nova, 125 from Horizon and 84 from Neutron. All the tickets have been analyzed by two of the three developers and the Table 1 shows the percentage of tickets classified as bug reports for the different developers. These results don't report for some combinations of developers because of in some projects, only a developer analyzed all the tickets and the two remaining analyzed the half of these tickets each one. The ticket Concordance that are not reported were not ready at time of authorising this paper, however they will provided in the future work related to this paper.

(FIXED: Discuss (more clearly) why results are not reported in Table 2 for combinations "D1 and D3" for Nova, "D2 and D3" for Horizon, and "D1 and D2" & "D2 and D3" for Neutron.)

The percentages between Developer 1 and Developer 3 are really similar, whereas the Developer 2 has identified more Bug Reports in his analysis. But, the three results support the misclassification present in bug tracking systems. Furthermore, according to [2]'s work, approximately two of five issues are misclassified in the analysis of Developer 1 and Developer 3.

[1] `bugtracking.libresoft.es`

[2] `https://www.youtube.com/watch?v=q0-TIvL4mqc&feature=youtu.be`

[3] `https://github.com/Gemarodri/BugTracking`

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

Focusing in the concordance between developers analyzing the same ticket, 417 tickets present a double bind review process, obtaining that each ticket was analyzed by two developers. Table 2 shows the percentage 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%

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 higher if they were expert in the project.

All the data from analysis of developers is available in the GitHub repositories of the developers [4][5][6], these repositories have the same name that the projects analyzed in OpenStack.

3.1 Future Work

(FIXED: This section needs to be linguistically improved)

The current tool is limited to OpenStack projects as a pilot projec since we conducting empirical studies based on openstack projects. In future we opt to extend the tool to include extracts of tickets from others systems such as Bugzilla or GitHub to study the misclassification in the OSS projects which use these systems. Also, display to users more information such as the code of the files affected in the fix commit, and the code in the bug seeding moment. Furthermore,

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

we would like to implement an auto classifier based on the semantic of the ticket description and fix-commit description to help researchers in the analysis. This classifier will show a percentage confidence about whether the ticket belongs to bug report or not, but the researcher will always have the last decision. The automatic classification will enable researchers focus only on problematic issues, which can be easily misclassified.

We also opt to investigate what will be the results if the data sources used by the tool to automatically extract tickets are used in isolation to to manually clasify bugs or other possible bug classifying tools. This will help us validate our results and the tool to further improve.

(FIXME 2:it is understood that the current version of the tool is limited to Openstack projects/services and with the server operating against Launchpad and Gerrit. How well can the tool support other OSS projects through use of other kinds of repositories. This is hinted in 3.1)
(FIXME 3: There are three styles specifying authors in the reference list. Make it uniform and conformant with the Springer conference paper requirements. Do not use "et al." in reference list, list all authors. For references 1 and 2 the source is not specified.)

References

1. Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., & Guéhéneuc, Y. G. (2008, October). Is it a bug or an enhancement?: a text-based approach to classify change requests. In Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds (p. 23). ACM.
2. Herzig, K., Just, S., & Zeller, A. (2013, May). It's not a bug, it's a feature: how misclassification impacts bug prediction. In Proceedings of the 2013 International Conference on Software Engineering (pp. 392-401). IEEE Press.
3. J. liwerski, J., Zimmermann, T., & Zeller, A. (2005, May). When do changes induce fixes?. In ACM sigsoft software engineering notes (Vol. 30, No. 4, pp. 1-5). ACM.
4. Nguyen, A. T., Nguyen, T. T., Nguyen, H. A., & Nguyen, T. N. (2012, November). Multi-layered approach for recovering links between bug reports and fixes. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (p. 63). ACM.
5. Zimmermann, T., Premraj, R., & Zeller, A. (2007, May). Predicting defects for eclipse. In Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on (pp. 9-9). IEEE.
6. Zimmermann, T., & Weigerber, P. (2004, May). Preprocessing CVS data for fine-grained analysis. In Proceedings of the First International Workshop on Mining Software Repositories (pp. 2-6). sn.
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.