HAL open science

BugTracking: A Tool to Assist in the Identification of Bug Reports

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

To cite this version:

Gema Rodríguez-Pérez, Jesús M. Gonzalez-Barahona, Gregorio Robles, Dorealda Dalipaj, Nelson Sekitoleko. BugTracking: A Tool to Assist in the Identification of Bug Reports. 12th IFIP International Conference on Open Source Systems (OSS), May 2016, Gothenburg, Sweden. pp.192-198, 10.1007/978-3-319-39225-7_16 . hal-01369071

HAL Id: hal-01369071

https://inria.hal.science/hal-01369071v1

Submitted on 20 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

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. Issue tracking systems are used, in most software projects, but in particular in almost all free open source software, to record 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 studies on the bug reports, managed by a software development project, first of all they need to perform this identification.

The job for researchers here is very different from the bug triaging that researchers do. In the latter case, people with a considerate 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 usually have not that experience in the project, but they have at their use 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. It simplifies large scale manual analysis of issues (in 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 to solve exactly the problem of providing the researchers with all the relevant information needed to decide whether an issue corresponds to a bug report or not. The tool uses information extracted automatically from the projects repositories. It offers a web-based interface which allows collaboration, traceability and transparency of the identification of bug reports. All this makes the process easier, faster, and more reliable.

Keywords: Issue Tracking System, Code Review System, Bug Triage, Tool

1 Introduction

While a software system is being developed, software engineers use version control repositories to produce and manage their code. Researchers 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 facilitate 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 researchers 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 could have used automatic classification systems, as described in [1], but the vocabulary used in the description of the issues could change from project to project, as well as the policy depending on the project. Consequently, data validation is recommended as mentioned in [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, 6] are based on text patterns or the mining of key phrases. Unfortunately, these methods include many false negatives causing bias in data [7, 8]. 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 in all of these methods, it is supposed that the issues are bug reports.

In this paper, we present a tool that displays, to the benefit of the researchers, a collection of all the necessary information needed to decide if an issue is a bug report or not. The tool, through the collection of exhaustive data on bug reports and the corresponding fix-commit, along with researchers extensive knowledge of the system, will help the last in their decision making, leading them into choosing only bug reports. This way they will not recur in any bias induced by non bug reports. To the best of our knowledge, this is the first tool that provides support to the identification of bugs and classification of bug reports. The need of the contribution of this tool arises from the increasing interest that both the academy and industry world is showing in the bug classification as a primary factor in modern software development.

2 The tool

The tool is a web application, therefore it runs in a browser. It displays the main data for distinguishing bug reports from others issues. The researchers will be responsible for classifying the issues from Launchpad as bug reports or not, and can thereby explain their decision for each issue. The issues are what we will refer as tickets during the paper.

2.1 Architecture

The tool integrates information from Launchpad as issue-tracking system and Gerrit as code review system. The figure 1 presents the architecture of the tool. The tool was developed using JavaScript, Node, JQuery and HTML5 technologies. The queries to the API of Gerrit and Launchpad are executed on server side. The responses are displayed on the client side. The end user can view the information displayed and interact with the server through events. Both sides exchange information using JSON files along with using their own REST API. Furthermore, we use a third-party application between GitHub and the browser in order to integrate some functionalities from GitHub.

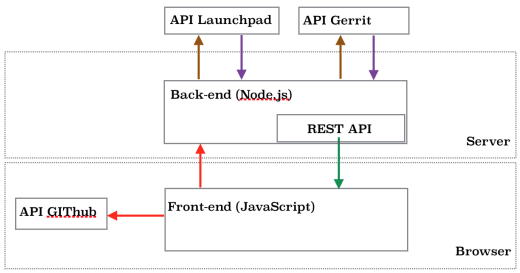

Fig. 1. Architecture of the tool.

2.2 Main Features

Figure 2 illustrates a screen short of the main tab of the tool. In particular, the tool displays the ID of the tickets which are extracted randomly from each issue-tracking repository of OpenStack. Other related information is displayed. Based on all these data, the researcher can 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 title and description of the report, as well as the description of the fix commit. For example for ticket ID 1531734 the tool displays the information related with the ticket in Launchpad and its corresponding review in Gerrit.

There is other additional information that the tool does not displays. If the researchers find it necessary, they can access the Launchapd and Gerrit web pages, respectively of the ticket and review, through the links provided by the tool. Thereby they can access extra information such as the comments written by code review researchers that correspond to that particular ticket. This provides a mean for tracking the history of the ticket from the moment it was opened until it was closed.

The tool further facilitates researchers to record and express their opinion about the ticket after reading all the information that is automatically 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 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, in the description of the ticket and commit message, that support their classification. Finally, they can leave their comment on why they classified a report as Bug, Not Bug or Undecided. Such information, in the future, will help us building an automatic bug classification system.

Another feature of the tool is that it allows to carry out a blind analysis of the tickets. Since all the data analysis inserted about a ticket is saved in a file on ones GitHub account, such analysis can be done by two or more researchers in parallel. By saving the data in GitHub, we could also measure the time that each researcher need for an analysis, which tickets were more difficult to analyze and other metrics that can help us understanding the current problem of issue misclassification.

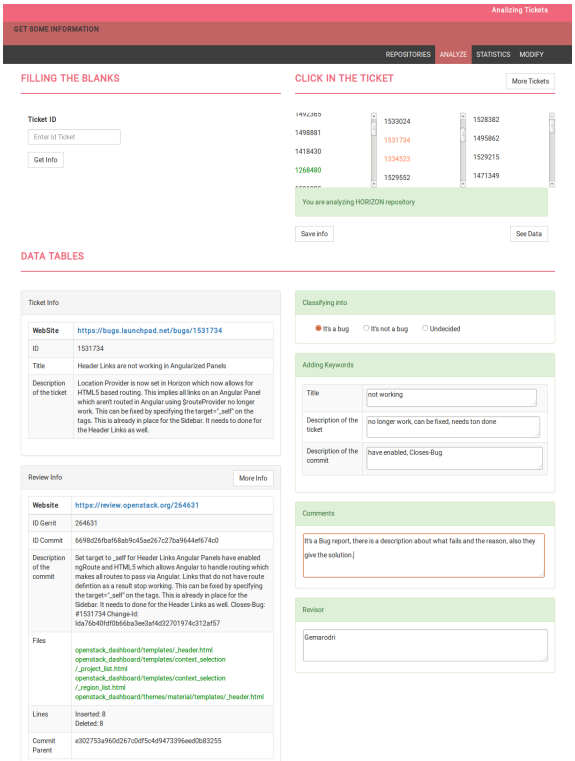

Fig. 2. Screenshot of Analyze Tab

The web page provides different functionalities depending on the tab the researcher is browsing. We explain these functionalities in the following:

1. Tab Repository: In this tab you can choose which repository you want to analyze. Currently the tool supports the four principal repositories of OpenStack: Cinder, Nova, Neutron and Horizon.
2. Tab Analyze: It is the Tab illustrated in Figure 2. It is where all the data from a specific ticket are displayed. The user can either select a random identifier or insert one of his choice. According to the data retrieved from Launchpad and Gerrit, the researcher can classify the ticket.
3. Tab Statistics: This tab extracts the data already analyzed by a researcher involved in the analysis from their user account in GitHub. It analyzes these data and displays a distribution of the classifications in a table;
4. Tab Modify: In case the researcher thinks to have inserted a mistake during the analysis, in this tab he/she can edit any of the data saved in his/her GitHub repository.

At the current state we present the initial version of the tool which is available at;[1], as well as a demonstration video[2]. It is 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 statistics of analysed tickets automatically, he/she should create a GitHub repository with the same name as the OpenStack project to be analysed, for example if the OpenStack project name is Nova, than the GitHub repository name should be Nova.

3 Results

We have analyzed 459 different tickets under the support of the initial version of the tool. 125 tickets where from Cinder, 125 from Nova, 125 from Horizon and 84 from Neutron. All the tickets have been analyzed by two out of the three researchers. The Table 1 shows the percentage of tickets classified as bug reports for the different researchers. These results don't report for some combinations of researchers because of in some projects, only a researcher analyzed all the tickets and the two remaining analyzed the half of these tickets each one.

The percentages between R1 and R3 are really similar, whereas the R2 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 R1 and R3.

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

[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 researcher

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

Table 2. Concordance between each researcher in each repository

	Nova	Cinder	Horizon	Neutron	Total
R1 and R2	(44/63) 70%	(40/52) 77%	(37/62) 60%	-	68%
R1 and R3	-	(46/63) 73%	(48/63) 76%	(26/42) 62%	71 %
R2 and R3	(41/62) 66%	(10/10) 100%	-	-	71%

Table 2 shows that the concordance of the researchers is high, but, also demonstrate the difficulty to classify tickets as bug report or as not bug report, because each researcher can have different opinions about a specific ticket. The concordance could be higher if they were expert in the project.

All data from the analysis are available in the GitHub repositories of the researchers [4], the repositories having the same name of the projects analyzed in OpenStack.

3.1 Future Work

Since we are conducting empirical studies based on OpenStack projects, the current tool is limited to OpenStack as a pilot project. In the future, our aim is to extend the tool at extracting tickets from others bug tracking systems, such as Bugzilla or GitHub where the server will operate against them to analyze the most OSS project. Additionally, we aim to study the misclassification in this OSS projects. We would like to add more features to the tool. One of them would be to display information such as the lines of code changed in the files affected from the fix-commit, along with the code in the bug seeding moment. Furthermore, we would like to implement an automatic classifier for the tickets, based on the semantic of the description of the ticket and the fix-commit. The result will indicate a percentage of confidence about whether a ticket is a bug report or not. However, the researcher will always make the final decision. The automatic classification will enable researchers to focus only on problematic issues, which can be easily misclassified.

We also aim 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 classify bugs or other possible bug classifying tools. This will help us validate our results and the tool to further improve.

[4] `https://github.com/{Gemarodri,ddalipaj,nellysek}`

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.