Software Development Process - The practice of linking issues and reviews.

Dorealda Dalipaj
Universidad Rey Juan Carlos
Madrid, Spain
Email: dorealda.dalipaj@urjc.es

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos
GSyC/Libresoft
Madrid, Spain
Email: jgb@gsyc.es

Daniel Izquierdo Cortazar
Bitergia
Madrid, Spain
Email: dizquierdo@bitergia.com

***Abstract*—Researchers working with software repositories, often when building performance or quality models, need to recover traceability links between bug reports in issue tracking repos- itories and reviews in code review systems.**
However, too often the information stored in bug tracking repositories is not explicitly tagged or linked to the issues reviewing them. Researchers have to resort to heuristics to tag the data (for example, to identify if an issue is a bug report or a work item).
In this study we promote a research artifact in software engineering, a reusable unit that can be used to support other research endeavors and has acted as a support material that enabled the creation of the results published in a great number of papers until now - linking issues and reviews of the code review process.
We present two *state-of-the-practice* algorithms on how to link issues and reviews. We selected OpenStack, the open source cloud computing project, as case study because it enforces strict development guidelines and rules on the quality of the data in its issue tracking repository.
We empirically compare the outcome of the two approaches, highlighting the most prominent one.

1. Introduction

Software process data, especially bug reports and code changes, are widely used in software engineering research. The integration of reports and reviews, provides valuable information on the history and evolution of a software project [1].
Additionally, it provides means for conducting studies that qualitatively and quantitatively analyse fundamental parameters for characterization of quality and performance of the code review and the most important metrics having a positive increasing relation with the benefits of this process. It is used to build models which study the impact of characteristics of issue and review discussions on the defect-proneness of a patch, to spot defect-introducing code changes during review, to predict the number and locale of bugs in future software releases, or to investigate technical and non-technical factors influencing modern code review ([2], [3], [4], [5], [6], [7]).
However, to achieve such studies, much of the information stored in software repositories must be processed and cleaned for further analysis. Such processing and cleaning is often done using assumptions and heuristics.
The quality of these assumptions and heuristics represents a threat to the validity of results derived from software repositories. Different studies by Bird et al. [8] and Antoniol et al. [9], have casted doubts on the quality of data produced using such heuristics.
To prevent or minimize such situations, we present two algorithms for linking issues and code review processes. We take into consideration the assumptions that researchers performing this task usually consider as true. The results evidence do not involve subjective context but are material facts. As such, they can be recorded to trace the efficiency and effectiveness of the two methods.

2. Methodology

Our approach can be summed up in the following steps:

1) *Extraction of the information from the data sources.* OpenStack repositories consists in the issue tracking (Launchpad) and code review (Gerrit) repositories. We extracted these data using Metrics Grimoire [1], a toolset to obtain data from repositories related to software development, provided and mantained by Bitergia.
2) *Identification of data needed for the linkage.* In this process we understand the infrastructure of both issue tracking and code review systems under analysis. We individuate the elements which hold the information needed to link issues with reviews. In the code review repository we found that a review holds the identification of the issue it is reviewing. In the bug tracking repository we found that the comments of an issue holds the identification of the review that is reviewing it. We need both information to link reviews with issues and viceversa.

1. https://github.com/MetricsGrimoire/

3) *Dataset preprocessing.* We do not need all the information provided by OpenStack repositories. First, we are interested in linking issues with reviews. Thus we can ignore all the tickets in the bug tracking system that are not classified as bugs. Second, we aim to recover the linkage for fixed issues. Thus we can ignore all issues for which the status is other than *Fix Released*.
4) *Analysis.* We executed both approaches using Jupyter Notebook, a computational environment where we can combine code execution, rich text, mathematics, plots and rich media, along with Pandas, an open source library specialised in providing high-performance data structures and analysis.

2.1. Approach 1 - Linking from review to issue.

This linking approach recovers linkage from code review to issue tracking system. To detect the links between reviews and issues, we referred to the name of the branch on which a code change had been made. It follows the naming convention *bug/989868* where *989868* is the issue identifier. We extracted this identifiers, and then proceded with the linkage. In summary, the *state-of-practice* algorithm is:

identify the *related artifact*;
apply regular expresions to extract *bug_id* from *related artifacts*;
link between issues and bugs matching where bug identification is *bug_id*.

The most inconvenient drawback of this approach is that, in practice, often, developers do not report such identifiers, resulting in missing links.

2.2. Approach 2 - Linking from issue to review.

This linking approach recovers linkage from issue tracking to code review system. To detect links between issues and reviews, we first identified the issues (bugs) and then extracted the identification of the review fixing it.
The first task is not a trivial one. In Launchpad, besides issues, developers work with *specifications* and *blueprints*. We analyzed Launchpad workflow and found a pattern that allowed us to identify which tickets are issues (bugs). These tickets follow a certain evolution of the *state* attribute. Next step, we preprocessed the comments of the issues to extract the review identifiers. In summary, the *state-of-practice* algorithm is:

identify *bug reports*;
exctract *issue_id*, the information related to the issue that is reviewing the bug from the comments of the bug;
link between bugs and issues where issue identification is *issue_id*.

Thanks to the heuristics applied in this case there are no false positives in the resulting dataset.

3. Results and Conclusions

After applying both *state-of-the-practice* algorithms to OpenStack, we linked 72% of the issues with the 1st Approach, and 90.2% with the 2nd Approach.
Clearly one approach outstands the other. We hope these results will be a guideline to other researchers in future helping them to decide how to recover the links at the best of their advantage.
While, on one hand, we tested our approach on a single case study, OpenStack, on the other what enfroces our confidence that this case acts as a representative for the category is that code review process follows the same principles and practices everywhere.

Acknowledgments

The authors would like to thank SENECA EID project, which is funding the research under Marie-Skodowska Curie Actions.

References

[1] M. Fischer, M. Pinzger, and H. Gall. *Populating a release history database from version control and bug tracking systems.* Proceedings of the International Conference on Software Maintenance (ICSM 03). IEEE Computer Society, 2003.

[2] *The Impact of Human Discussions on Just -In-Time Quality Assurance.* Parastou Tourani and Bram Adams. SANER 2016 - 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering.

[3] A. Bernstein, J. Ekanayake, and M. Pinzger. *Improving defect prediction using temporal features and non linear models.* In IWPSE07, pages 11-18, Dubrovnik, Croatia, September 2007.

[4] Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. W. (2015). *Investigating technical and non-technical factors influencing modern code review.* Empirical Software Engineering, 1-28.

[5] H. Joshi, C. Zhang, S. Ramaswamy, and C. Bayrak. *Local and global recency weighting approach to bug prediction.* In MSR07, pages 33-34, Minneapolis, Minnesota, USA, May 2007. IEEE Computer Society.

[6] M.-T. J. Ostrand, F.-E. J. Weyuker, and R. M. Bell. *Predicting the location and number of faults in large software systems.* IEEE Trans. Softw. Eng., 31(4):340-355, 2005.

[7] J. Sliwerski, T. Zimmermann, and A. Zeller. *When do changes induce fixes?* In MSR05, pages 24-28, Saint Louis, Missouri, USA, May 2005. ACM.

[8] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. *Fair and balanced?: bias in bug-fix datasets.* Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Sngineering (FSE 09). Amsterdam: ACM, 2009, pp. 121-130.

[9] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.G. Guhneuc. *Is it a bug or an enhancement?: a text-based approach to classify change requests.* Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds (CASCON 08). Ontario, Canada: ACM, 2008, pp. 304-318.