Characterization of the Xen Project Code Review Process: an Experience Report

Daniel Izquierdo-Cortazar
Bitergia
dizquierdo@bitergia.com

Lars Kurth
Citrix
Lars.Kurth@xen.org

Jesus M. Gonzalez-Barahona
GSyC, Universidad Rey Juan Carlos
jgb@gsyc.es

Santiago Dueñas
Bitergia
sduenas@bitergia.com

Nelson Sekitoleko
Bitergia
snelson@bitergia.com

ABSTRACT

Many software development projects have introduced mandatory code review for every change to the code. This means that the project needs to devote a significant effort to review all proposed changes, and that their merging into the code base may get considerably delayed. Therefore, all those projects need to understand how code review is working, and the delays it is causing in time to merge.

This is the case in the Xen project, which performs peer review using mailing lists. During the first half of 2015, some people in the project observed a large and sustained increase in the number of messages related to code review, which had started some years before. This observation led to concerns on whether the code review process was having some trouble, and too large an impact on the overall development process.

Those concerns were addressed with a quantitative study, which is presented in this paper. Based on the information in code review messages, some metrics were defined to infer delays imposed by code review. The study produced quantitative data suitable for informed discussion, which the project is using to understand its code review process, and to take decisions to improve it.

Keywords

data mining, software process, code review

1. INTRODUCTION

Peer review of proposed changes (patches) is being adopted as a best practice to improve quality in free, open source software projects. Some drivers of this trend are the fact that it is a relatively light-weight process, the existence of tools supporting it, and a fast time-to-production approach, as compared to traditional code inspection that emphasizes face to face meetings and uses of check lists [5] [3] [4] [1].

MSR'16, May 14-15, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2901739.2901778

Recent studies have further highlighted the benefits of performing tool-supported code review: team awareness, finding of alternative solutions to problems, and knowledge transfer [1]. With an increasing number of software development projects emphasizing mandatory code review for every change to the code, a number of factors come into play that may have an impact on product and process quality, cost, and time-to-market [2] [5]: code review coverage and participation [5], length of the review queue, experience of the patch writer/reviewers, size of the patch [1], time-to-merge, neutrality of reviewers, etc. All of them need to be measured and studied in practice to obtain a clear understanding of how the code review process impacts the overall development effort. This is especially true for free, open source software projects, whose development effort is in most of the cases unknown [6].

One of those projects is Xen, which produces virtualization software for the Linux kernel. All proposed changes in Xen follow a code review process which is run in the development mailing list, using some tags and conventions that make it possible to automate, at least in part, its analysis.

During the first semester of 2015, some people in the project had noticed an increase of the messages related to code review. They were concerned of which impact this could be causing on several key aspects of their development efforts. Among them, the most important was the time-to-merge, or time from the moment a change is proposed, to the time it is merged in the code base. Those people were afraid that an increase in the number of messages could mean an increase in the discussions to review a patch, and in the time that process could take.

To understand the situation, and decide which measures to take, the project commissioned a study, which was carried out by the authors of this paper. This study analyzed the Xen code review process in order to address those concerns, and to provide the project with data to evaluate the situation, take decisions, and track the effect of those decisions. This paper presents the main results of the study, and describes how it was performed.

The main objective of this study was to verify the apparent increase in communication related to code review, and to determine if some key parameters of the code review process, mainly time-to-merge, were deteriorating. In case that would be happening, the study aimed to pinpoint the main

causes of this deterioration, so that corrective actions could be put in place.

In particular, some people were suspicious that the root of the problem could be linked to a growing number of review comments for large "combined patches" (patch series), which could be causing a very large growth of their time-to-review during the period of study.

Therefore, we can state that the main research questions were:

RQ1 Is time-to-merge increasing, measured from the moment a change is proposed to the moment it is finally merged into the code base?

RQ2 Is there an impact of size of "combined patches" (patch series), measured as the number of individual patches in them, on time-to-merge for those patch series?

2. INITIAL OBSERVATIONS

The Xen community code review process takes place in mailing lists. Each proposal of change to the code is sent to the mailing list, where it is publicly reviewed by peer developers. In this process the project uses the following terminology:

- *Patch*: the basic unit of change, consisting of the changes to one or more files, that the author consider tightly related.
- *Patch series*: a combined patch, composed of more than one patch, which have some interdependency. Usually, either all the patches in the series are merged, or none is. In this paper, we consider patches submitted individual as patch series of only one patch.
- *Version*: each of the successive proposals for a patch serie. Usually, a patch serie is resubmitted several times, and tries to address the comments by reviewers. Each of these resubmissions is a new version of the patch series.

Table 1 and Table 2 show some parameters that help to understand the activity in the Xen project.

No. of commits	No. of emails	No. of Patches
34,494	257,572	146,977

Table 1: Some parameters related to the activity observed in Xen repositories (2002-2015)

No. of Patch series	No. of patches	No. of matchings
17,722	33,853	13,271

Table 2: Activity directly related to code review for Xen. Matchings are the number of patches that were matched with commits in the git repository (see the section on methodology).

The analysis was focused on the 2012-2015 period, which was the period of concern for the project. The evolution of the number of comments in the mailing list, related to code review, during this period, is shown in Figure 1, which clearly shows an increase during the period, therefore confirming the initial concerns. Previous history is added for adding context.

Meanwhile, Figure 2 shows a slight increase in the number of new patch series (including patch series composed of a single patch). Considering both charts together, the initial concerns of a more complex review process, measured as the number of comments per patch set, was confirmed.

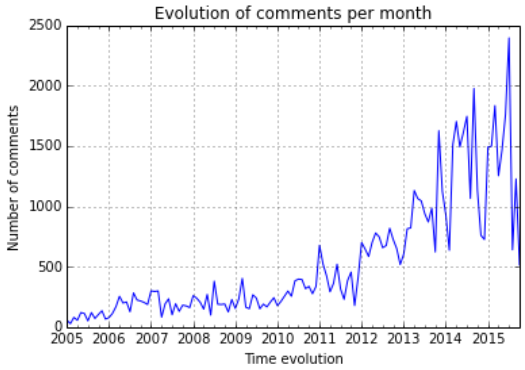

Figure 1: Number of comments per month related to code review

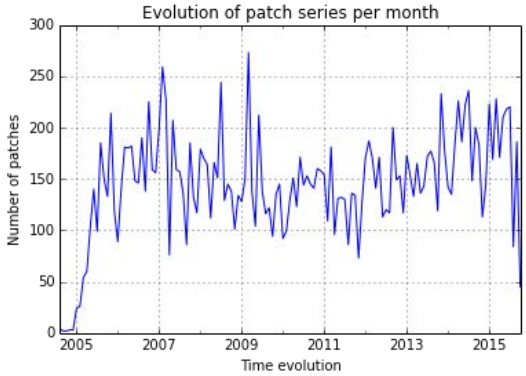

Figure 2: Number of new patch series (combined patches) proposed per month

3. METHODOLOGY

The code review process in the Xen project is similar to that of the Linux kernel, and takes place through a mailing list. In order to convert messages in a mailing list into organized information, we parsed mailing list archives, and matched patches found in them to commits in the git repositories in order to measure time-to-merge. The process was as follows:

Step 1. Retrieval of mailing list information. We used MLStats [7] to retrieve mail messages from the development mailing list[1], and store them in a MySQL database.

Step 2. Retrieval of Git information. We used CVSAnalY [7] to retrieve information from the main Xen git repository[2], and store it in a MySQL database as well.

[1]http://lists.xen.org/archives/html/xen-devel/
[2]http://xenbits.xen.org/gitweb/?p=xen.git

Step 3. Detection and classification of messages related to code review. The relevant messages are those with the keyword *PATCH* in the subject. A new patch series is identified by a message starting a new thread, with its composing patches in replies to it, in the same thread. New versions are detected as a new threads with the same subject field, but a version number. The pattern in all the subject fields related to code review is similar to:

```
[<Keyword PATCH> <version> <patch number>] <subject>
```

Unfortunately, there are small variations of this pattern, which led us to use some regular expressions to match and identify all this information in subject fields.

Messages may be tagged, providing information about its semantic in the review process. Some tags are: 'Acked-by', 'Cc', 'Fixes', 'From', 'Reported-by', 'Tested-by', 'Reviewed-by', 'Release-Acked-by', 'Signed-off-by', and 'Suggested-by'. Although some of these tags are not really important for the results presented in this paper, they could be used to obtain more precise results.

Step 4. Merging information from CVSAnalY (git) and MLStats (mailing list) databases[3], including linking patches to commits. This produced a consolidated database with the following tables:

- patch_serie: detected patch series.
- patch_serie_versions: each of the versions per patch serie.
- patches: all of the available patches.
- comments: comments received by a patch.
- people: people involved in the process.
- tags: tags found in each reply.
- commits: all of the commits.

Step 5. Analysis[4]. This consisted in querying the previous database to obtain evidence to answer the relevant questions.

Table 3 shows the raw numbers about the information in the consolidated database: number of patches identified in the mailing list, number of commits identified in the git repository, and number of commits linked to patches in the mailing list.

Regarding to the numbers, commits is directly provided by the tools, while other metrics rely on regular expressions. This is the case of patches. For matching commits to patches, we used heuristics based on finding the subject line of messages to the first line of the commit message, which could be wrong in some cases, although is a very reliable method according to Xen developers.

The number of patches identified (that is, the number of code review processes found) is, for the period of interest, large enough (up to 50%-60% of the number of commits) to draw conclusions. But of course any conclusion is limited to to those code review processes. The number of patches linked to specific commits is smaller (55%65% for 2013–2015). This is mainly because there are other Xen-related git repositories that were not included in the analysis, but were discussed in the mailing list. We confirmed this, with the help of Xen developers, by a manual analysis of the patches that could be tracked to their corresponding git commit messages.

In addition, some noise was found in the dataset, for example some patches that were reported in 2015 bearing dates of 2016, yet when the most recent commit included in the study was of October 13th 2015.

All this said, a sample of matches commits was analyzed manually, in collaboration with Xen developers, to ensure that no apparent bias was present.

[3]Available as an IPython Notebook https://github.com/dicortazar/ipython-notebooks/blob/master/projects/xen-analysis/xen_patches.py

[4]Available as an IPython Notebook https://github.com/dicortazar/ipython-notebooks/blob/master/projects/xen-analysis/Code-Review-Metrics.ipynb

Year	No. of patches	No. of commits	No. of commits matching to patches
2012	1907	2296	954
2013	2345	2503	1396
2014	2035	2332	1315
2015	2060	2204	1244

Table 3: Number of patches, commits, and commits corresponding to patches identified in the combined database (matched commits)

4. RESULTS

In the following discussion, the second semester of 2015 was removed both due to having only partial data for it, and to ensure that most of the review processes during the period of study had the chance of finishing.

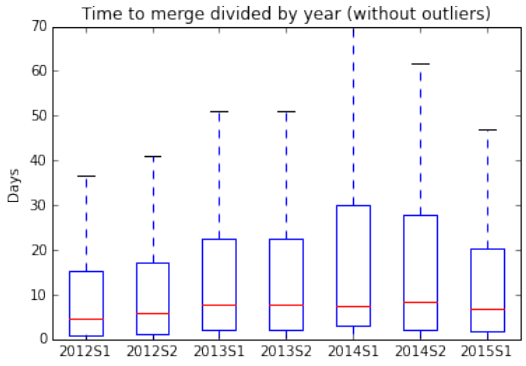

Figure 3: Evolution of time to merge

- Answer to RQ1: Time-to-merge is under control.

  Figure 3 shows the evolution of time to merge by semester. From 2012 to 2014, there is an increase in the time to merge. For example, in 2012 75% of the patch series that were merged, were merged in less than 15 days, while in 2014 this number doubled to around 30 days. But the second semester of 2014 and the first semester of 2015 show a change in this trend, coming back to shorter time-to-merge.

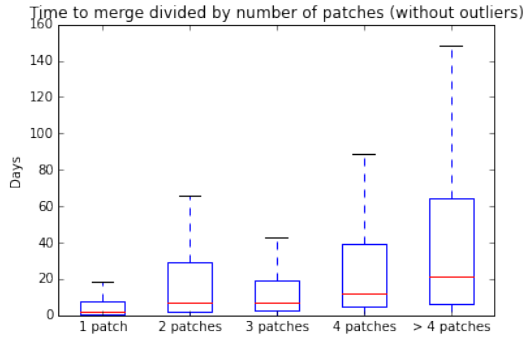

Figure 4: Time to merge patchsets (days) by number of patches

Therefore, although concerns were confirmed for the period until 2014, the situation is reverting to controlled.

- Answer to RQ2: Time-to-merge is behaving in a similar way for all sizes of patch series.

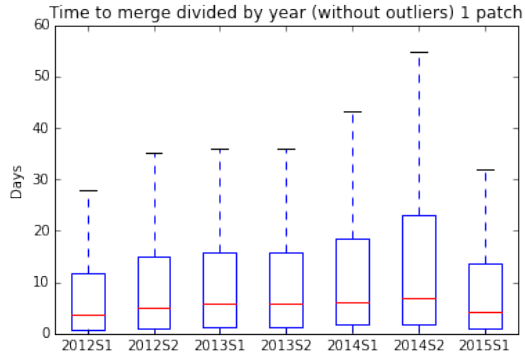

Figure 5: Time to merge (days) for patch series composed of a single patch.

Figure 4 shows time-to-merge when classifying patch series by number of patches per patch serie. As it may be expected, the higher the number of patches per patch serie, the higher the time to merge. But Figure 5 and Figure 6 show how the evolution of time-to-merge for patch series of 1 patch, and of 5 or more patches, is very similar. Although patch series with 5 or more patches take longer to review than patch series of one patch, the trend is pretty similar between them, and to the one found in Figure 3. The peak of the time to merge takes place in 2014 and then starts to decrease.

5. CONCLUSIONS AND FURTHER WORK

The analysis of the code review process in Xen provided data to better understand its impact on time-to-merge. Data show an increase of time-to-merge from 2012 to 2014, and a decrease after that peak, starting during the second semester of 2014. Small and large patch series show the same pattern.

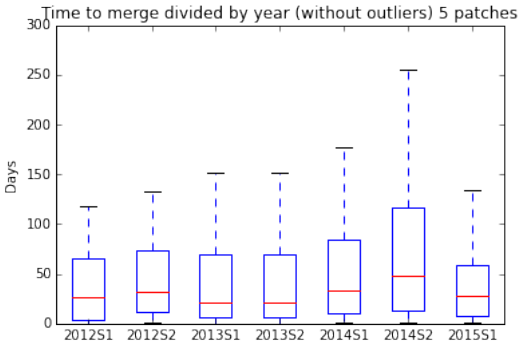

Figure 6: Time to merge (days) for patch series composed of more than 4 patches.

The results were presented to the community, and validated with them[5]. Most of the comments and concerns expressed were focused on the need of increasing the fraction of commits that were linked to patches, but no other serious problems were highlighted.

In addition to time-to-merge, some other metrics were discussed within the community. However, the time to merge was the most representative metric for its needs. Some other metrics that were considered, and could be the subject of further studies, are: time to commit, time to re-work a patch, cycle time (between each pair of versions), time to first review.

The proposed methodology focuses on how the Xen community reviews pieces of code through the mailing lists. There are some other projects with similar code review processes, such as the Linux Kernel where this method may be applied as well.

As further work, more git repositories of interest to the Xen community will be added to the study, such as mini-os, raisin, and osstest, which are reviewed as well in the xen-devel mailing list. In addition, extra work should be done to improve the detection of threads in the review process. We have found some cases where a patch serie is divided into several patches, sent in several threads instead of in the same one as expected. However, as developers are adopting *Patchbomb*, a tool helping in the building of messages related to a patch series, this process is becoming more and more automated, which helps in the identification process.

6. ACKNOWLEDGMENTS

Daniel Izquierdo is partially funded by the Torres Quevedo program of the Spanish Government. Jesus M. Gonzalez-Barahona is partially funded by the Spanish Government, through project TIN2014-59400-R. Nelson Sekitoleko is funded by the Marie SkÅĆodowska-Curie - Research Fellowship Programme of the European Union, through the Seneca Consortium, grant agreement 642954. Lars Kurth has participated in the design and validation of the study, as an expert in the Xen community. The other authors have participated in the design, execution and validation of the study.

7. REFERENCES

[5]http://lists.xenproject.org/archives/html/xen-devel/2015-10/msg01808.html

[1] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering*, pages 712–721. IEEE Press, 2013.

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

[3] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens. Modern code reviews in open-source projects: which problems do they fix? In *Proceedings of the 11th working conference on mining software repositories*, pages 202–211. ACM, 2014.

[4] S. Kollanus and J. Koskinen. Survey of software inspection research. *The Open Software Engineering Journal*, 3(1):15–34, 2009.

[5] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201. ACM, 2014.

[6] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 222–231. ACM, 2014.

[7] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. Herraiz. Tools for the study of the usual data sources found in libre software projects. *International Journal of Open Source Software and Processes*, 1(1):24–45, 2009.