Code review process analysis in the Xen Project Executive Summary

Bitergia

October 19, 2015

AUTHORS

Daniel Izquierdo Cortázar
Chief Data Officer
dizquierdo@bitergia.com

Jesús González Barahona
Chief Science Officer
jgb@bitergia.com

Objectives of the analysis

Code review in the Xen project is performed in the developers mailing list. During the last years, it had been observed an apparent increase in the number of messages devoted to code review, and in particular, an increase in the number of code review messages per patch serie or individual patch. The main objective of this analysis is to verify or not this apparent increase, to study other parameters of the code review process to determine if it is deteriorating in some way, and to pinpoint the main causes of this deterioration, if any.

This report summarizes the first stage of the analysis, and proposes some lines for a second stage.

Key findings

Time-to-merge is under control

Time to merge increased from 2012 to the first semester of 2014. During the first semester of 2012, 75% of the patch series were merged in less than 15 days, while in 2014 the same percentage was merged in less than 30 days. But since then, the time to merge has decreased, being of about 28 days in the second semester of 2014, and 20 in 2015 (for 75% of patch series, in both cases).

The trend of time-to-merge is similar despite the size of the patch series

The same trend of time to merge (increasing until early 2014, decreasing afterwords) is observed for patch series composed of one, two, three, four or more than four patches. Although there are some differences in the intensity of increase and decrease, all these populations show similar trends.

Methodology

This report aims at providing insights about the code review process in the Xen project. As the process takes place through a mailing list, a tool and some specific scripts have been developed for analyzing it. The approach followed has been:

- Retrieve the mailing list archives, and organize their contents in an SQL database. For this step, the MLStats tool[1] has been used.
- Retrieve the git repository for the project, and organize its metadata in an SQL database. For this step, the CVSAnalY tool[2] has been used.
- Analyze both databases using some Python scripts, producing some new tables with the relevant events and relationships. The data from the mailing lists was used to identify code review processes for patch series, and specific information in the messages was used for tracking specific events in the code review process, such as submission of a new version of a patch series. The data from the git repository was used to determine the commits corresponding to patch series that landed in the code base. Some heuristics were used to link code review messages with their related commits.
- Produce a IPython / Jupyter Notebook with the analysis of the events tables, to obtain evidence that allows to answer the relevant questions.

After the analysis of the database, and the use of heuristics for matching commits to patches, Table 1 shows: the number of patches identified in the mailing list, the number of commits identified in the git repository, and the number of commits linked to one patch in the mailing list.

[1] `http://metricsgrimoire.github.io/MailingListStats/`
[2] `http://metricsgrimoire.github.io/CVSAnalY/`

Year	Number of patches	Number of commits	Commits corresponding to patch
2011	1,559	2,181	581
2012	1,907	2,296	954
2013	2,345	2,503	1,396
2014	2,035	2,332	1,315
2015	2,060	2,204	1,244

Table 1: Number of patches, commits and commits corresponding to patches identified in this study (matched commits)

Of these numbers, commits in the git repository is the most reliable, since it is only a matter of counting. For the number of patches, we have relied on messages with subject starting with "PATCH", and used some heuristics to identify small variances in the subject as corresponding to the same patch. This means that the number could not be completely accurate. For matching commits to patches, we used simple heuristics based on matching the subject line of messages to the first line of the commit message.

In any case, the number of patches identified is close enough (70%-90% during the years of interest) to the number of commits for the results to be meaningful. The number of matched commits is smaller, being for the years 2013-2015 in the range 55%-65%. Assuming the sample of matched commits is not heavily biased, the results seem representative enough to draw conclusions.

Should not be too biased, given we were looking at data that does not require good matching (such as the backlog)

BUT: backlog data is incomplete

Note that we did not want to focus on getting perfect matching first, but get a sense of where the issues were.

Results

This analysis was triggered by some observations that the interactions in the mailing list for Xen code review were increasing during the last years. This could be signaling that the review process was requiring more resources, and that individual code reviews were taking longer until merging in the code base was decided. It could happen as well that some bottlenecks had emerged, causing the project to be less efficient when merging new functionality and resolving issues.

The analysis has focused on:

- Validating the initial observation of an increase in messages per patch series reviewed.
- Looking for evidences of a degraded code review process, in terms of more versions of the patch series being required to accept them, or in longer review processes.
- Characterizing the patch series under review, to learn about how they are growing (or not) in complexity and size, and how this could be influencing the previous parameters.

Validation: Messages per patch series

The analysis has verified that the number of messages per reviewed patch series has been increasing steadily during the last years (see figure 1). In the period 2012 to 2015, the median of messages per patch series has grown from around 2 to about 4. This means that half of the patch series were reviewed with 1 or 2 messages in 2012, while it required between 1 and 4 messages (or even 5) during 2014 and 2015.

The mean number of messages per patch series has increased even more, which given the behavior of the median, means that the dispersion of the number of messages per patch series is growing quickly. In other words, a small fraction of reviews are needing many more than those 1-4 messages needed for half the reviews.

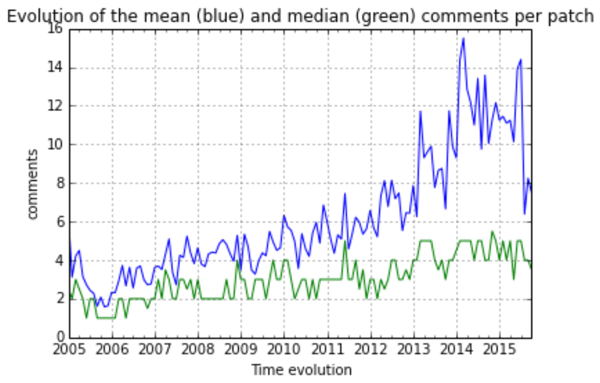

Figure 1: Code review messages per patch series.

This observation, resulting form the analysis of all the code review threads in the mailing list, is consistent with the initial observation that lead to this study.

Characterization of patch series

The analysis has characterized the reviewed patch series from several points of view. The most relevant of those has been finding the number of patches composing a patch series. Figure 2 shows the evolution of the size of patch series (in number of patches) over time. It shows how at least 50% of the patch series were composed by just one patch (median), while the mean number of patches per patch series has grown from 1.5 in 2012 to about 2.5 in 2015. This means that the dispersion of patches per patch series has increased during this period.

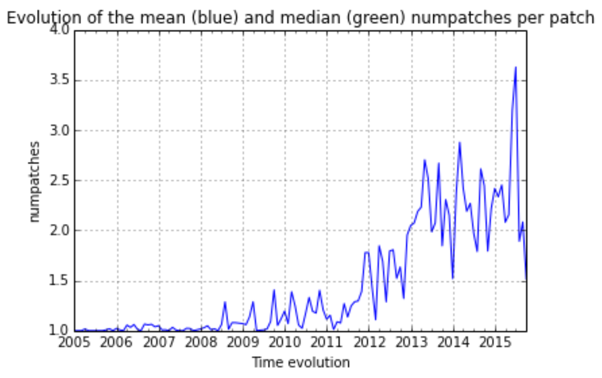

Figure 2: Mean and median number of patches per patch series for each month. Median is constantly at 1.0, in the bottom part of the chart.

The chart in Figure 3 sheds some more light on the matter of the size of the patch series. In it, the total population of patch series has been split in five populations, according to the number of patches they include (one, two, three, four or more than four). From this chart, it can be seen how most of the patch series are clearly those composed of one patch. The next relevant population is that with four patches or more, which has been growing steadily since 2012. The other populations (two, three, four patches) are not very relevant.

Therefore, for analyzing how different patch series sizes (according to the number of patches) affect the duration of the review process, the most relevant will be those composed by one or by more than four patches.

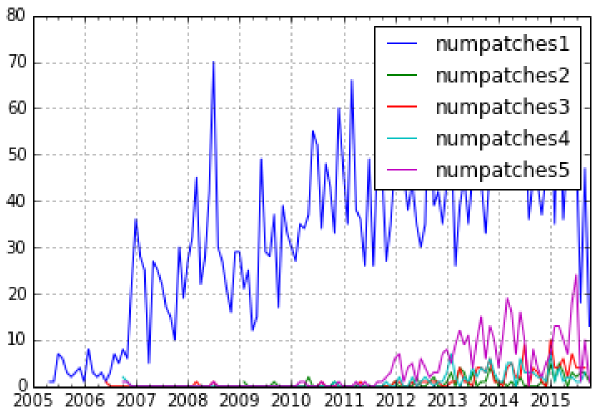

Figure 3: Number of patch sets with one, two, three, four, and five or more patches over time (per month).

The chart in Figure 4 shows the evolution over time of the number of files touched per patch series. This was considered at some point to be a possible cause for a longer duration of review processes, due to increased complexity of the patch series. But the chart shows how in the period 2010-2015, the mean and median number of files per patch series has remained relatively constant (median between 1 and 3, for example). Therefore, during the studied period, this parameter has not changed meaningfully.

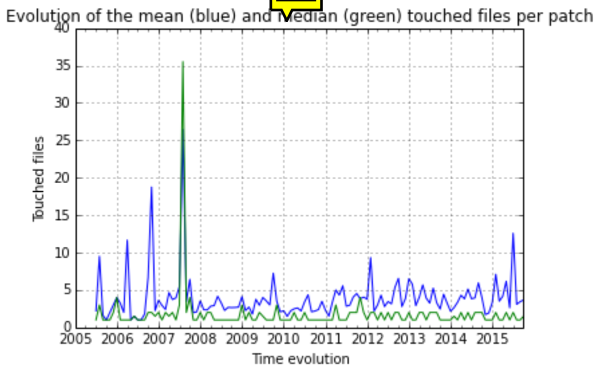

Figure 4: Number files touched per patch series over time (per month).

Duration of review processes

For evaluating the duration of the review processes associated with patch series, two parameters have been studied with detail:

- Time-to-merge. Time from the submission of the first version of the patch series to the merge in the code base of the corresponding commits.
- Number of versions until merge. Number of versions that are sent to the mailing list until the corresponding commits are merged in the code base.

The first parameter gives an indication of how long, in time, is it taking from the proposal of a change to the code, until that change actually lands in the code. The second gives an indication of how many "rounds" (working to produce a patch version, reviewing it) it takes until a change is approved. Both parameters have been calculated only for patch series that got merged (that is, approved patch series).

Time to merge

The analysis has devoted most of the attention to time-to-merge, since this is usually the most relevant parameter, and the one that is to be minimized. Figure 5 shows its evolution per year, for all patch series for which final commits were identified.

The representation of the data in this figure is based on boxplots, which sketch the distribution of the data by representing the border values after splitting the samples in four quarters. Each of the regions delimited by the boxes represent one of these quarters of the total population. The region below the blue box shows the behavior of up to the 25% of the population (the 25% percentile), the red line represents the median (50% percentile), and the top of the blue box delimits 75% of the population. Over that top blue line, the last 25% of the population is represented.

With this interpretation in mind, we can see a constant increase in the time to merge from 2012 to 2014. This is highly visible by following the 75% percentile of the distribution. In 2012 75% of the patch series were merged in less than 20 days, while in 2014 the time for merging that 75% increased up to 30 days. This trend is noticeable as well in the median values. However, 2015 shows a change in trend, with the values coming back close to those of 2012.

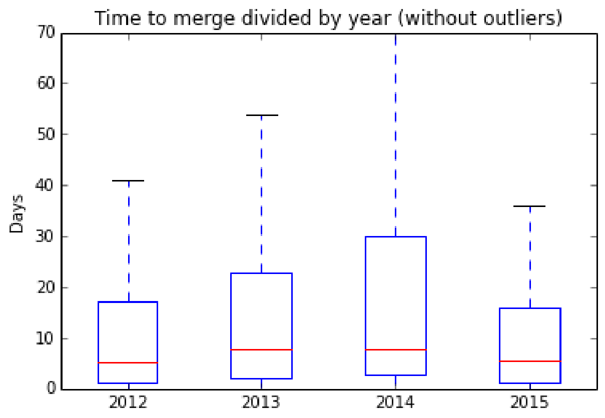

Figure 5: Time to merge a patch series into the git repository, per year.

There are two effects that could influence these numbers. One is the aggregation over periods so large like years, which could mask trends over shorter periods. The other one is that for 2015, still many of the review processes that are going to accepted are still running. This latter effect means that the final number for 2015, once all the patch series initiated this year are accepted, can be different, probably longer, than the number that can be calculated now.

To avoid both problems, periods of 6 months were analyzed as well. For the last of these periods, the first semester of 2015, most of the accepted patch series were already finished during the period of study (which ends in early October), thus minimizing the effect of patch series not yet merged, but which will be merged at some point.

Figure 6 shows this semester-split version of the same dataset. It shows a continuous increase in the time to merge from the first semester of 2012 until the first semester of 2014. The second semester of 2014 shows a change in the trend, with time to merge starting to decrease, until it is in 2015 similar to the second semester of 2012.

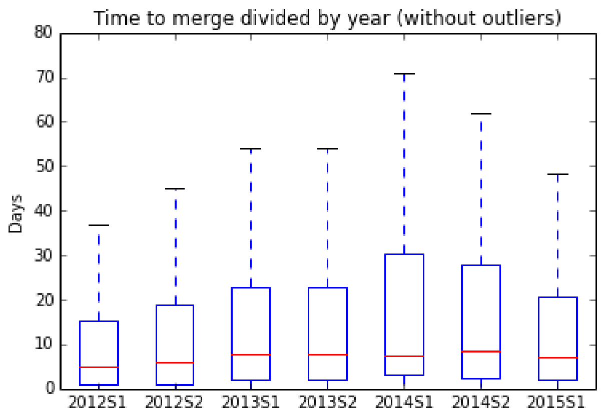

Figure 6: Time to merge a patch series into the git repository, per semester

To study the effect of patch series size (by number of patches), Figure 7 shows the behavior of one-patch patch series, while Figure 8 shows it for patch series with five or more patches.

The one-patch patch series show a stable trend in median values, although a certain increase in time to merge is observed until the second semester of 2014. After that period, time to merge decreases to levels of 2012 during the first semester of 2015. The patch series with more than four patches shows a similar trend, but with longer values during 2014. It also shows longer time to merge, topping at more than 100 days for the 75% percentile during the second semester of 2014.

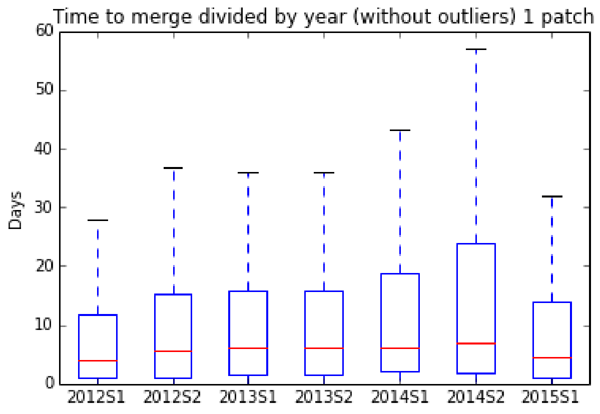

Figure 7: Time to merge split by semester for one-patch patch series, per semester.

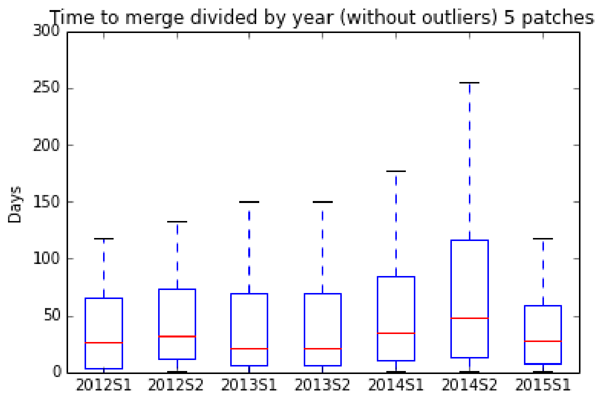

Figure 8: Time to merge split by semester for patch series with five or more patches, per semester.

Iterations per review process

To complete the analysis on duration of code review processes, the number of iterations (patch versions) per code review process was calculated as well. Figure 9 shows the evolution of the mean and median of this parameter, per month. During all the studied period the median remained stable at 1 iteration per review process, meaning that at least 50% of the reviews required only one patch version. However, the mean has been growing, to peak during 2014. This is an indication of a larger spread of the number of iterations, and that some review processes took a larger number of iterations to complete. But even this parameter has started to decrease during the first semester of 2015, staying at about 1.4 iterations per review process.

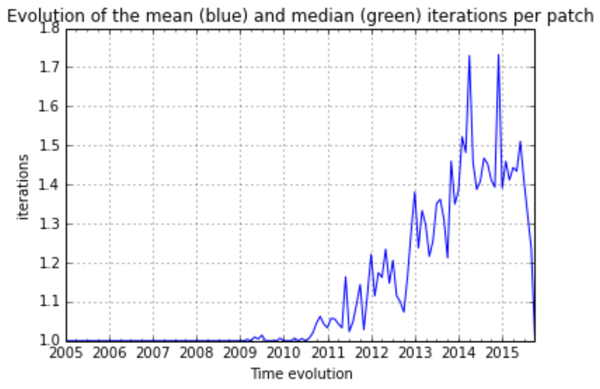

Figure 9: Iterations (number of versions) per accepted patch series over time (per month). Median is steady at 1.0, in the bottom of the chart.

Duration: summary

As a summary of the previous observations, it can be said that the duration of the review process in Xen became longer in the period 2012-2014, but since then it has been controlled, and even decreasing. The data for 2015 is still not complete, and could change due to the effect of reviews still being processed, but for now it shows not only a contention in the duration, but a decrease, both in terms of time-to-merge and of iterations.

References and future work

References with further information, and elements to reproduce the analysis:

- IPython Notebook with more detailed data:
  `https://github.com/dicortazar/ipython-notebooks/blob/master/projects/xen-analysis/Code-Review-Metrics.ipynb`
- Database:
  `http://projects.bitergia.com/xen-reports/xen_reports3.7z`
- Script to produce database:
  `https://github.com/dicortazar/ipython-notebooks/blob/master/projects/xen-analysis/xen_patches.py`

The current report is the finalization of the first stage of analysis. A second stage could be started afterwards. Some lines for future work during this second stage are:

Deployment and automatic update
Currently, the IPython/Jupyter Notebook can be updated automatically from new versions of the review database. The process of generating this database from CVSAnalY and MLStats database can be automated. Since both CVSAnalY and MLStats databases are already being updated frequently, this will ensure that the code review stats are updated with the same frequency.

Improvement of the matching algorithm
Some of the most interesting parameters, such as time-to-merge, depend on the correct matching of threads in the mailing list to commits in the git repository. The current heuristics are matching about 60% of the commits. This figure could be improved significantly.

Analysis of the backlog over time
The current analysis (see the Notebook) includes some preliminary analysis of the backlog of pending code reviews. However, for this analysis to be useful, it should more clearly differentiate between superseded, abandoned, and still alive code reviews, and should provide data about the evolution over time, the age structure of pending reviews, etc.

Production of JSON documents with detailed information
JSON documents with relevant parameters per review can be produced. Using those documents, further tooling can be produced with relative easy, which allows to easily detect reviews inactive for long periods, reviews with long review processes, etc.

Website with drill down information about code review activity
All of this information, once curated and automatically updated, could be provided into a website where any interested person could check all of these data with drill down information per path serie, activity timeframe, developers, and other parameters.