Evolution of the core team of developers in libre software projects

Gregorio Robles, Jesus M. Gonzalez-Barahona, Israel Herraiz
GSyC/LibreSoft, Universidad Rey Juan Carlos (Madrid, Spain)
{*grex,jgb,herraiz*}*@gsyc.urjc.es*

Abstract

In many libre (free, open source) software projects, most of the development is performed by a relatively small number of persons, the "core team". The stability and permanence of this group of most active developers is of great importance for the evolution and sustainability of the project. In this paper we propose a quantitative methodology to study the evolution of core teams by analyzing information from source code management repositories. The most active developers in different periods are identified, and their activity is calculated over time, looking for core team evolution patterns. Several activity plots and parameters for characterizing these patterns are presented, and applied to several large, well-known libre software projects, where their effectiveness is validated and discussed.

1. Introduction

Employee turnover is known to be high in the traditional software industry since many years ago [1]. However, in libre software[1] projects the study of developer turnover has not been an active research topic. Most of the attention in this area has been focused on the organizational structure of the projects [2], with little attention to the dynamics of the developers.

A noteworthy contribution in this sense, although it does not address the evolution of developer communities, is the *onion model* [3], which shows how developers and users are positioned in communities. In this model, it is possible to differentiate among core developers (those who have a high involvement in the project), codevelopers (with specific but frequent contributions), active users (contributing only occasionally) and passive users [4], [5].

The onion model provides only a static picture of a project, lacking the time dimension that is required for studying joining and leaving processes. Jensen et al. [6] have studied and modeled the processes of role migration for some libre software communities, focusing on end-users who become developers. This has lead to the finding of different paths for the joining process, concluding that the organizational structure of the studied projects is highly dynamic in comparison to traditional software development organizations. With respect to abandonment, the number of developers leaving a project has been studied in [7] by using the half-life parameter, defined as the time required for a certain group of contributors to fall to half of its initial population; in the case of the Debian project the obtained half-life was of 7.5 years.

1. In this paper we will use the term "libre software" to refer both to software licensed under terms compliant with the FSF definition of "free software" or with the OSI definition of "open source software".

Given these precedents, the authors of this paper begun to study projects to better understand the evolution of the group of developers contributing to a libre software project, in particular of the the most active ones. For this, a characterization is proposed based on two extreme scenarios: the "code gods" scenario, in which the composition of the core team is highly stable over time, and the "series of generations" scenario, where several generations succeed each other. In the first case, a project relies heavily on a small number of long contributing developers, which would imply a great risk in case a significant fraction of them leave. In the second case the project shows a series of partially overlapping core teams, with some of the initial members leaving the project, but others joining and filling the gap.

A specific methodology has been designed to quantitatively characterize a project in the spectrum between these two scenarios, and to visualize more in detail the evolution of the core team. The first steps of this methodology, applied to a few projects, are depicted in [8]. An extension and refinement of the methodology is presented in this paper, where it is used on a larger amount of libre software projects.

The rest of this paper is organized as follows. The next section describes the methodology that has been designed to extract data from source code management systems and produce indexes, graphs and maps that help to understand the evolution of the core team. After it, several cases are discussed to illustrate the use of the methodology, and to some extent validate it. These cases are a part of the study of 19 large libre software projects, which are also discussed in combination. A final section with some conclusions and hints about further research closes the paper.

2. Methodology

The methodology used in this study is based on retrieving data about the activity of developers from source code management repositories, which are mined using CVSAnalY [9]. This tool retrieves information about every commit to the repository, and inserts it into a database where it can be conveniently analyzed.

2.1. Summary of the methodology

To characterize the evolution of the core team, first the life of the project is split in periods of equal duration. Then for every period i, the most active developers are identified as $CoreTeam_i$. This is done by calculating the number of commits during that period for the most active developers. For each $CoreTeam_i$, its activity is tracked for the rest of the life of the project (before and after period i). Hence, for each period j, the number of commits is calculated for all the developers in $CoreTeam_i$. Finally, the resulting data (that represents the activity of the each $CoreTeam_i$ for all periods) is plotted in several formats, and collapsed into some indexes that allow comparison and classification.

In all the cases, only commits on source code files have been considered, since the study is focused on the behavior of developers working on source code (for further details, please refer to [10]).

There are some cases which can be a source of problems when applying the methodology: both automatic commiters (present in some projects for routine operations) and different patterns of work may influence the results for specific projects. However, experience after analyzing a large quantity of projects tends to show that these effects can, in general, be neglected.

In addition, $CoreTeam_i$ and $CoreTeam_j$ may include developers in common, as a developer can be a part of the most active group in several periods. This is important to notice to correctly interpret the information provided by the methodology.

2.2. Duration of the periods

The selection of the duration of the periods in which the life of the project is divided is basic for the definition of the notion of “history” of a core team. The smaller the period, the more transient effects can be found (such as developers on vacation, or different activity patterns when the project is close to a release). But the larger the period, the lesser detail is captured, maybe losing important transitions, meaningful for our study.

In addition, projects of different life lengths will have different number of periods, if periods of constant duration are used. This could cause difficulties when comparing results, but will at the same time allow to better understand the behaviors related to the length of the history of projects. The opposite option, using a constant number of periods for projects of any length, can be more useful for comparisons, but will neglect those aspects related to project age.

Because of all these trade-offs, we have not considered a single time span for periods. For the purposes of the study, usually the most significant results are obtained by dividing the history of the project into 10 or 20 periods. These also produce periods of a reasonable duration both for relatively young (about 2 to 4 months for projects with 3 years of development) and old projects (6 to 12 months for projects with around 10 years). To ensure that we are not losing any important information, we have obtained several plots using equal time intervals of three months; as it will be shown in later sections, using the former or the later does not give significant differences.

2.3. Identification of the core team

Libre software projects usually show power law or similar distributions for the number of contributions by developers [4], [11], [12]. In other words, a small fraction of all the developers are responsible for a large fraction of all the activity. Because of that, the criterion chosen for the identification of core teams we have chosen is the fraction of developers who produce more commits.

After considering several alternatives, we have found that fractions of 0.1 and 0.2 (that is, the top 10% and 20%) are large enough to capture developers producing most of the activity (usually more than 50%, reaching in many cases as much as 90% or 95% of the total number of commits).

3. Outputs of the methodology

Our methodology provides both some graphs that help to visualize the results and some data (in the form of arrays and indexes).

3.1. Arrays and indexes

The main output of the methodology is the $AbsoluteMatrix$: a squared two dimensional array, with the number of periods as range. Values for each position x, y in the array are the absolute number of commits for $CoreTeam_x$ in period y. Therefore, positions in the diagonal (where $x = y$) correspond to the activity of each core team during the period in which it is actually the core team. Positions where $y > x$ represent the activity of that core team in periods after that moment, while $y < x$ represent the ’past’ activity of that team.

From other point of view, each row (same x value) represents the history of activity of $CoreGroup_x$. Each column (same y value) gives the activity of all core groups during period y. These clues can help to interpret absolute matrices, and the graphs produced when plotting their contents.

Absolute matrices, carrying absolute number of commits in each position, are sensitive to the total activity of the project. In some cases, this makes it difficult to compare periods of different levels of activity (for instance, when the focus is the importance of a certain core team during all the periods). For these cases, the $NormalizedMatrix$ is produced. It is calculated from the absolute matrix, using for each position its original value divided by the total number of commits in the corresponding period:

$$NormalizedMatrix_{x,y} = \frac{AbstoluteMatrix_{x,y}}{TotalCommits_y}$$

An alternative chance for normalization is to use the activity of the core team of each period instead of the total activity in each period (it can be easily shown how the positions in the diagonal in this matrix have always a value of 1):

$$CoredMatrix_{x,y} = \frac{AbstoluteMatrix_{x,y}}{AbsoluteMatrix_{y,y}}$$

Complete arrays provide a lot of information, but they are also in some cases too detailed and difficult to interpret. Therefore, a single parameter, $Index$, which summarizes the information in a matrix, could be calculated as follows:

$$Index = 100 * \sum_{x \neq y} CoredMatrix_{x,y}$$

Note that high values for this index are indicative for a higher "load" on all positions, which means that the activity of the different core teams is high over their whole history (a situation that is close to the "code gods" scenario, as this happens when the composition of the core group changes seldom). The smaller $Index$ is, the more positions with little activity, pointing out the existence of a heterogeneity of developers composing the core teams (having a "series of generations" scenario).

3.2. Graphs

Several graphs are produced to visualize and help with the interpretation of the previous data:

- Absolute graph. Displays the absolute number of commits for each core group (Y axis) for each interval over time (X axis). This graph is obtained by plotting the data in each $AbsoluteMatrix_x$ row.
- Aggregated graph. Displays the aggregated number of commits for each core group since the beginning of the project (Y axis) versus time (X axis). This graph shows the integral of the absolute graph.
- Normalized graph. Displays the fraction of the total commits performed by each core group for each interval (Y axis) versus time (X axis). This graph shows the same information than the absolute graph, but normalized by the total number of commits performed in each period, and can be obtained by plotting the data in each $NormalizedMatrix_x$ row.
- Heat map. Displays the $CoredMatrix$, with a color (or gray-scale) for each position. Provides a quick yet detailed view of the evolution of core groups over time. The history of each core group can easily be observed, with high activity periods as *hot* areas, while periods with little activity will appear *cold*.
- Normalized 3D map. This is a three dimensional view of the $NormalizedMatrix$, with Z axis representing the normalized activity per position. Provides similar information to the one in the heat map, but can be interpreted with more detail if a 3D browser is available.
- Absolute 3D map. This is a three dimensional view of the $AbsoluteMatrix$, with Z axis representing the activity per position. Similar to the normalized 3D map, but information about the total level of activity for each position is also provided.

The combined observation of these graphs, for different time periods (10 or 20), and using different fractions of developers for identifying core groups (top 10% or 20%), provides a complete landscape of the activity of the core group over time project.

4. Some case studies

The methodology described in the previous section has been applied to 19 different projects. All of them are at least six years old, so that they have enough history to analyze. Table 1 in the Appendix shows a summary of the main parameters of these projects (including several indexes, which will be discussed below and are a good estimator for the behavior of their core teams).

Among them, we have selected three cases to illustrate the use of the methodology more in detail. Two of them are close to the extreme scenarios: code gods (the GIMP) and series of generations (Mozilla). The third project (Evolution) is between these to ends, and shows some peculiarities that are worth mentioning. Results for the rest of the case studies are given in combination in the last subsection.

4.1. Case study: the GIMP

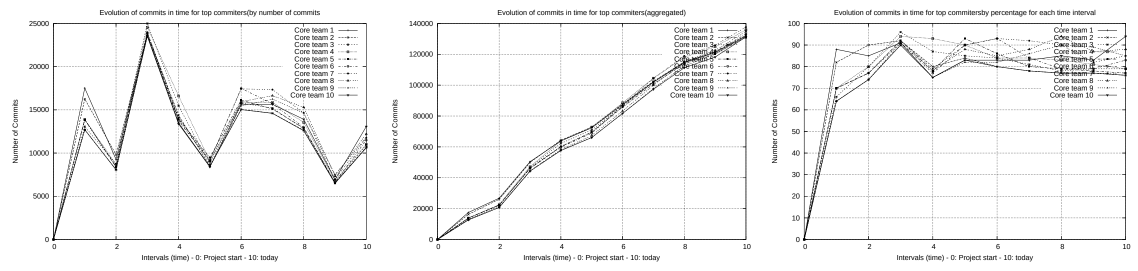
The GIMP can be considered as a canonical example of a project with "code gods". It is a very active project (by number of commits) with many developers involved.

Graphs in figure 1 show the typical pattern of code gods scenarios. The lines in all graphs are almost overlapping, which means that all the core teams have almost the same composition. However, the core team is not always exactly the same. A detailed study of the developers in the core teams yields that one of the most active developers is present in all of them. The second and third most active developers enter during the third interval (which starts around mid 1999) and stay in the project until today.

The normalized graph, also shown in figure 1, provides further information. By construction, the higher curve in each period corresponds to the core team that has been identified in it. In the case of a "code gods" project, the other core groups should be near that maximum (or at the

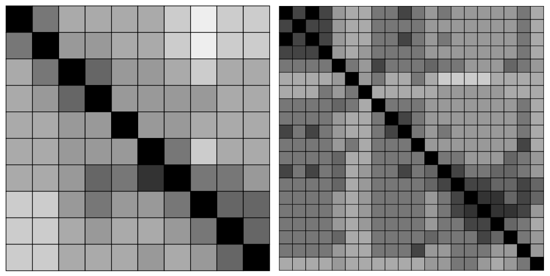
Project	Size	Commits	Commiters	Age	Index 10 / 10	10 / 20	20 / 10	20 / 20
Eclipse	4298K	801403	189	68	22.26	33.35	20.22	32.25
Evolution	300K	125938	369	121	32.19	28.79	29.84	29.35
FreeBSD	2085K	241809	345	164	38.61	40.97	36.37	39.48
Galeon	93K	34826	139	120	35.81	28.40	36.52	30.07
GIMP	603K	187522	229	121	45.20	37.22	51.53	39.01
Gnumeric	253K	106327	206	121	49.12	41.14	51.56	46.76
Kdebase	373K	251241	550	117	31.3	33.83	30.78	35.40
Kdelibs	585K	241916	572	117	36.68	34.42	38.51	35.67
Kdenetwork	318K	157762	387	115	20.64	24.68	19.17	22.80
Kdepim	549K	154833	301	108	34.10	33.51	29.22	30.44
KOffice	906K	251823	308	105	30.57	31.24	34.29	32.70
Mcs	1757K	183242	216	67	28.02	34.92	25.72	33.48
Mono	297K	35085	143	66	57.32	48.33	48.4	50.58
Mozilla	3940K	1007370	835	106	18.59	23.49	18.68	50.58
Nautilus	101K	71920	332	121	19.06	19.52	19.57	20.18
NetBSD	2888K	465060	287	164	27.64	40.31	26.24	36.03
OpenBSD	1734K	140213	183	132	43.78	43.25	42.50	42.86
OpenOffice.org	5149K	129209	107	76	31.64	38.91	29.87	36.36
PostgreSQL	381K	90282	28	127	68.67	67.67	73.63	64.73

Table 1. Summary of parameters for the projects analyzed. Size is in SLOC, age (of the repository) is in months. For indexes, 10/20 means: number of periods is 10, core teams identified as top 20% (0.2 fraction) of developers.



(a) Absolute (b) Aggregated (c) Normalized

Figure 1. Graphs for the GIMP project. A fraction of 0.2 was used for identifying core teams.



(a) 10 periods; 0.2 fraction (b) 20 periods; 0.1 fraction

Figure 2. Heat maps for the GIMP project. Fraction provides the fraction of of top developers to identify the core team.

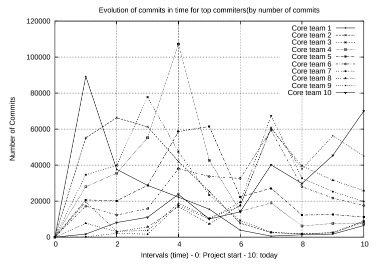
same level if core groups during different time periods have exactly the same composition) as the composition has not changed much over time.

The identification of the code gods scenario is even more evident in the heat maps of figure 2. Except for the diagonal (which is, by construction, always black), the gray color dominates the map, meaning that the composition of the core groups over time is quite similar. The right map, with a higher resolution, shows also the special case of the first core groups: the upper left positions are darker, and are surrounded by lighter ones, showing a change in generations.

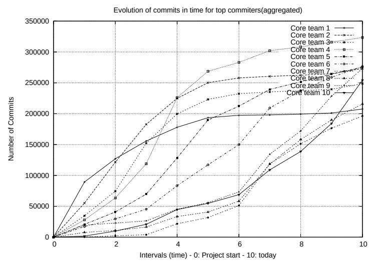
The 3D maps of figure 3 provide some more detail. In the normalized map, the lighter plateau that dominates most of the map is a clear indicator of a stable code gods region. Again, the beginning of the project shows a slightly different pattern, with a different composition of the core group.

It is worth noticing that both normalized and absolute 3D maps, when projected on the XZ plane, produce the normalized and absolute graphs. Moreover, thanks to how the normalized map is colored, when projected on the XY plane, the resulting 2D map should result in the heat map. Therefore, these 3D maps in some sense include all the information in the other graphs and maps.

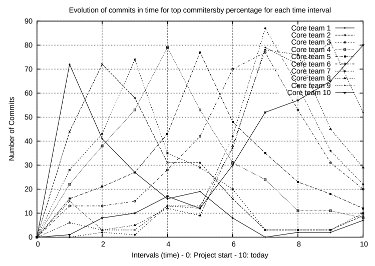
In addition to all this graphical information, the indexes shown in table 1 in the Appendix are also an indicator of a code gods scenarios, being among the highest in the table.



(a) Absolute

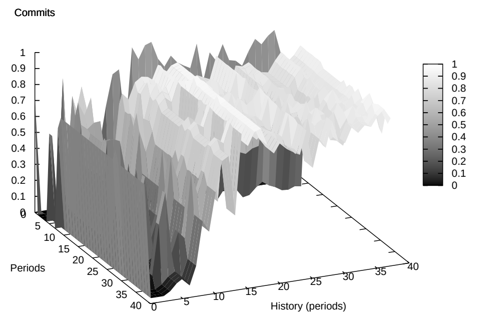


(b) Aggregated



(c) Normalized

Figure 4. Graphs for the Mozilla project. A fraction of 0.2 was used for identifying core teams.



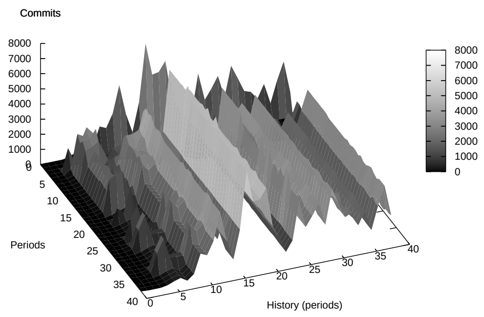


Figure 3. 3D maps for the GIMP project, using quarters as period, and 0.2 as fraction of top developers for identifying core teams. Top is normalized map, bottom is absolute.

4.2. Case study: Mozilla

The Mozilla project is a good example of a series of generations scenario. Figure 4 shows the graphs for the project. Their aspect is, at first sight, clearly different from those of the GIMP. Curves now are very different for each core team. They are not parallel in the aggregated graph, neither almost coincident in the absolute one. In the normalized graph, each team raises from a very small fraction of contributions to a peak of about 80%-90%, and then fades quickly (in about three years) away. In other words, the composition of the core teams varies clearly from period to period. There are clear indicatives of smooth transitions between different core teams: several developers leave or join during each period, but many stay for several periods, ensuring smooth transitions.

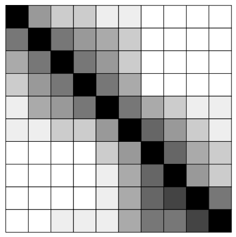


Figure 5. Heat map for the Mozilla project. 10 periods and 0.2 as fraction of top developers identified as core team.

However, even the core teams for the last periods show some activity during the first periods, which means that some very active developers nowadays were already active at those early stages of the project. Correspondingly, the first core team also has activity during the last periods. Therefore, the experience from the beginnings of the project is still available in the core group (at least in the minds of the developers active in core teams during all the life of the project), which is another sign of smooth transitions.

In the heat map (see figure 5) wide areas are white, and most of the activity is concentrated around the diagonal. Again, this shows the succession of generations. Its smoothness can be appreciated but the two or three gray positions around the black diagonal: each core group shows activity before and after its peak. But additional, more subtle information is found in this map. At about periods 5 and 6, a clear transition is observed. Core teams tend to have more activity either before or after those periods. Not surprisingly, this transition happens at 2003, when the corporate support of AOL finished and the Mozilla Foundation was founded. The heat map shows how the composition of teams changed more significantly around that time, although with a certain level of smoothness.

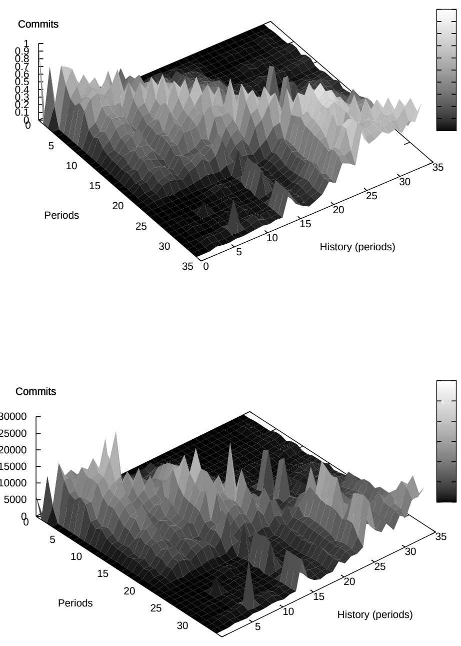


Figure 6. 3D maps for the Mozilla project, using quarters as period, and 0.2 as fraction of top developers for identifying core teams. Top is normalized map, bottom is absolute.

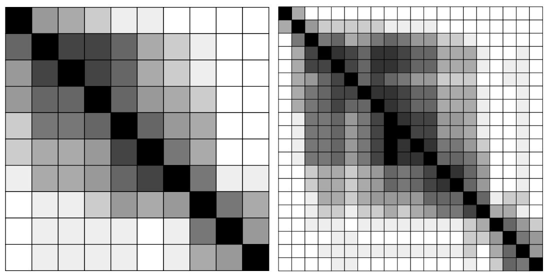
This effect is also apparent in the 3D maps (figure 6, with those wide low, dark areas (that represent low levels of activity for the corresponding core teams). The mountain chain aspect of those maps is a clear indication of changes in the composition of the core teams, which happen ubiquitously for this project. The normalized 3D map is specially illustrative in this respect.

The indexes for this project (offered in table 1) are among the lowest of all the analyzed (except for the 20/20 case).

4.3. Case study: Evolution

Evolution is a good example of a project showing both aspects of the "code gods" and the series of generations scenarios. Evolution started as a community-driven project in December 1998, but was quickly adopted (by the end of 1999) by Ximian (then a small start-up company) as a strategic application. From that point on, Ximian developers had a clear impact on the project. Ximian was acquired by Novell in August 2003, with most of the team working in Evolution also joining the new company.

Both the absolute and aggregated graphs in figure 7 show clearly the low activity of the core team at the beginning of the project, and the impact (in the rise of the activity) of Ximian developers joining the project. At the end of the project, again low levels of activity are found, probably after the spreading of Ximian developers within Novell. These three periods also correspond to differences in the composition of the core team. The normalized graph (also in figure 7) shows how the first core team quickly fades out to less than 50% of activity after less than two years, while the last core teams rise their peak level after periods with very little activity. Between these two ends, the periods in the middle (specially those from 2 to 6, which correspond to the life of Ximian as a company) are close to a code gods scenario (parallel lines in the aggregated graph, almost overlapping curves in the normalized graph). In summary, we can observe an epoch of code gods, while the first and last periods are more close to the series of generations.



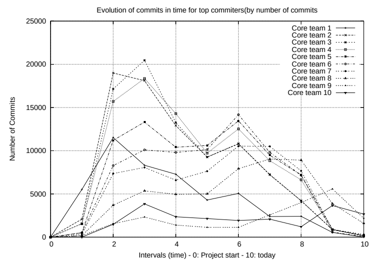
(a) 10 periods; 0.2 fraction (b) 20 periods; 0.1 fraction

Figure 8. Heat maps for the Evolution project. Fraction provides the fraction of of top developers to identify the core team.

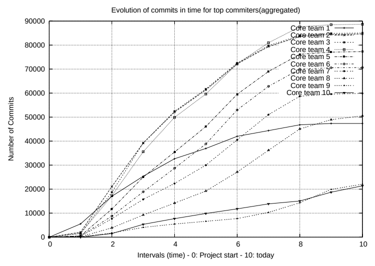
Heat maps in figure 8 show the same pattern. The transition from the first periods to the Ximian epoch is smooth but clear. The large square of gray positions (corresponding to 6 core teams in the 10x10 map) points out the code gods period, while the small square in the bottom right suggest the beginning of a new code gods era. The transition to this last square is clearly more sharp than the one at the beginning of the project.

3D maps (see figure 9) are even more clear. The absolute map shows a low elevation at the beginning of the project, and a much higher mountain chain after it. Behind, some small hills correspond to the last periods. The transitions between the three epochs, and the differences in smoothness are obvious.

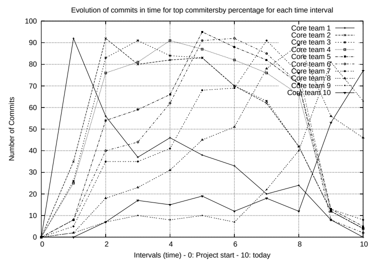
Finally, the indexes for Evolution are closer to a code gods case than to a series of generations one, which is reasonable if we consider that most of the time, the former is the more similar scenario. However, the aspects related to the transitions are not captured by the indexes (which is not surprising, being the index a really terse summary of a complex pattern).



(a) Absolute

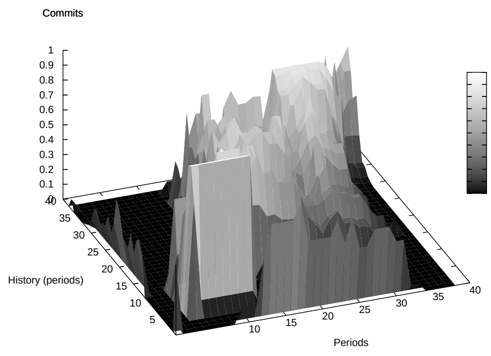


(b) Aggregated



(c) Normalized

Figure 7. Graphs for the Evolution project. A fraction of 0.2 was used for identifying core teams.



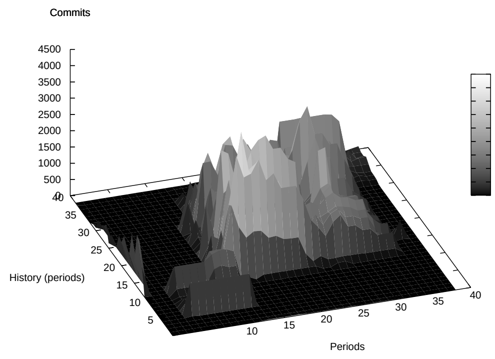


Figure 9. 3D maps for the Evolution project, using quarters as period, and 0.2 as fraction of top developers for identifying core teams. Top is normalized map, bottom is absolute.

4.4. Observations on other projects

After presenting the results of applying the methodology to three projects, some more general observations on the 19 projects analyzed (summarized in table 1 in the appendix) are offered in the following paragraphs.

Especially interesting is figure 10, which plots the indexes of the different projects versus their age. It servers two purposes: to offer a quick graphic summary of the projects, and to show that there is no relationship between age and index.

Showing the lack of correlation between age and index is important, since it could seem intuitive that projects tend to the series of generations scenario with time (given enough time, the chances of developers leaving the project, and others coming in, could be higher). However, we cannot observe this behavior from the studied projects. Those with a larger history in the sample (NetBSD and FreeBSD) show clear differences in their indexes, and in any case they are much more “code gods like” than others with much shorter histories, such as Nautilus or Mozilla. It is also important to notice that for the projects around the 120 months mark, widely varying indexes can be found, ranging from the clear cases of series of generations (Nautilus, Kdenetwork, Mozilla) to those of code gods (OpenBSD, GIMP and Gnumeric). Postgres has (with difference) the highest index. One could argue that this is due to its policy where most commits are performed by a small group of developers, which in many cases commits patches produced by external contributors, but this is really no limitation to the methodology as we are mainly studying if and how this most active group changes over time and not specifically, although also important, its relevance. In this regard, we could state that the methodology is insensitive to *gate keepers* (which is actually a good characteristic as it is a bias that is difficult to minimize).

Some other correlations for the indexes (versus project size, number of developers, number of commits, etc.) have been explored, with no clear result. With the data in our study, it seems that the reasons for the differences in the evolution of the core teams, from project to project, are not related to the characteristics of the software being developed, neither to the level of activity. Other reasons for this differences, maybe linked to the specific policies and procedures of the projects, remain to be found.

Fractional graphs for some of the studied projects are also shown (see table 2 in the appendix). They have been sorted according to their index, from Nautilus, the most close to a series of generations scenario (top left) to Mono and Gnumeric (clear cases of code gods). Each of them is an interesting case, which can be analyzed from the graphs (and the corresponding heat and 3D maps) in detail, as

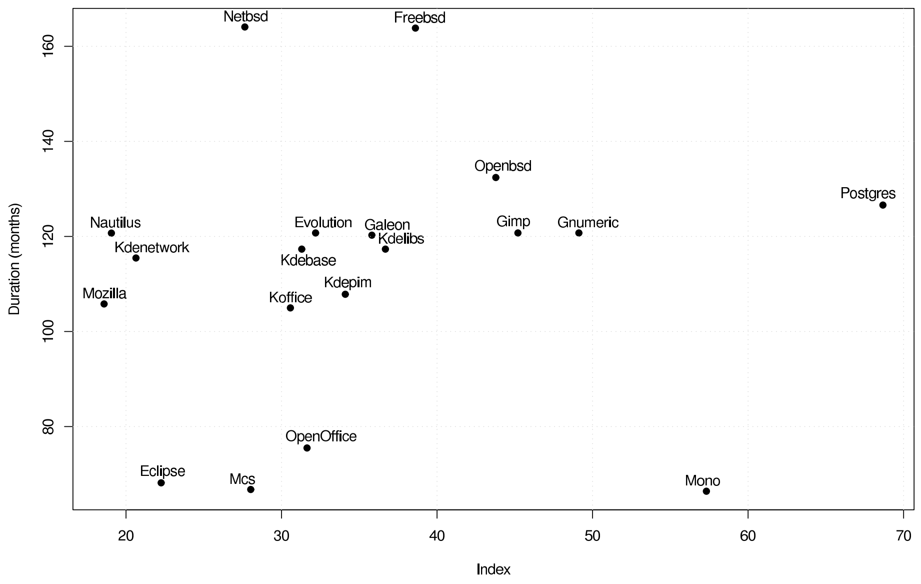


Figure 10. Plot of indexes versus project age, for the projects included in the study. Indexes are calculated by splitting project life in 10 periods, with core teams of top 0.1 of developers.

were Mozilla, GIMP and Evolution. Just as a very brief summary, a single aspect can be highlighted: the differences in smoothness in transitions, which can be inferred from the different rates of ascending and descending curves before and after reaching their peak.

To complete the general discussion of cases, figure 11 shows four plots with different methods for calculating the index, for all the studied projects. Although of course the sample of projects is neither large nor unbiased enough to raise any conclusion about the distribution of indexes, some details can be highlighted. For instance, both plots at the top are quite similar, and the same can be said for the plots at the bottom. Therefore, the index can be considered as almost independent from the fraction of developers considered for identifying the core team (at least if it is between 0.1 and 0.2).

However, the index is less immune to changes in the size of the periods considered: the plots on the top present clear differences with those on the bottom (and a more detailed analysis shows not only differences in the plots, but also in the order of projects). In fact, from an exhaustive analysis of all the projects considered in this study, the indexes calculated with 10 periods match better the real stories of the projects.

5. Conclusions and further research

For the study presented in this paper, a methodology has been designed that allows for a simple yet powerful analysis of the evolution of the core team of libre software projects. The methodology is quantitative, and can be automated, only requiring that the development is performed using a source control management system, and that the researcher has access to the corresponding repository. Fortunately, this is the case for a large fraction of libre software projects, including the most relevant ones.

The methodology can be used to rank projects according to their distance to the two extreme cases of “code gods” and “series of generations”, using the produced indexes. But it provides also a lot of insight on the evolution of the core teams, by showing visually (both in graphs and maps) the activity patterns of the developers forming the core team in each period of the life of a project. This information can be used to identify levels of smoothness in transitions, to detect break points in the evolution of the core team, to understand the differences in activity of the core team in different periods, or to estimate unevenness in the contributions of the most active developers when compared to the rest of them.

In addition, we have applied the methodology to 19 relevant libre software projects, and used these case examples to validate it and illustrate some of its benefits. We also have shown the impact of some events in the history of specific projects on the evolution of their core team (such as the influence of the strategy of companies in the cases of Evolution and Mozilla).

Some factors not specifically discussed in this paper could influence the appropriateness of the methodology. Among them, the relevance of using the number of commits as a proxy for the activity and importance of developers. For validating it, we have studied some other parameters, such as the number of changed lines, without finding meaningful differences. However, an important problem remains open:

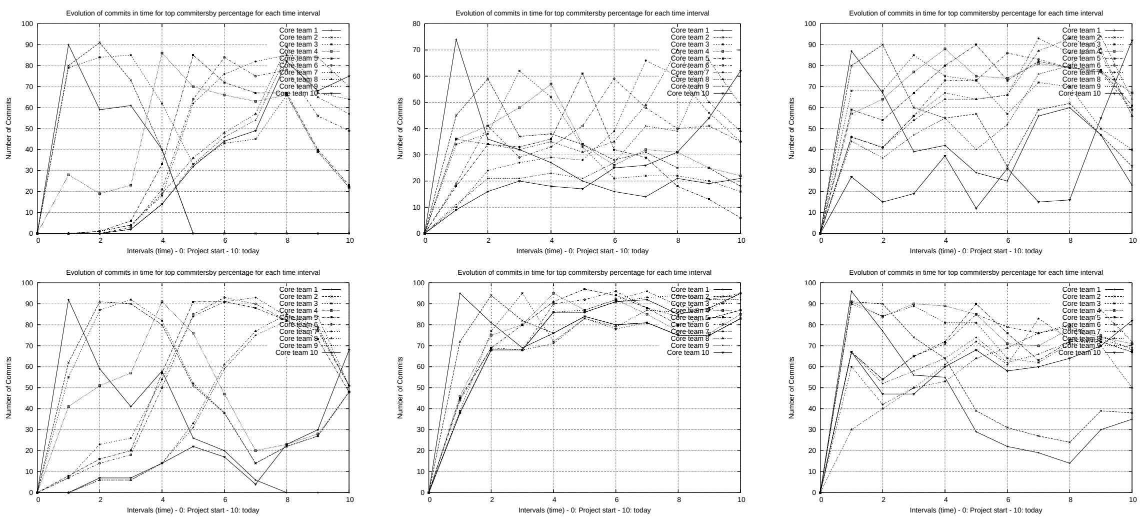


Table 2. 3x2 matrix with fractional generation plots for 6 libre software systems. From left to right and top to bottom, they are Nautilus, Eclipse, KDE base libraries, Galeon, Gnumeric and Mono. Projects closer to the series of generations scenario have been situated at the top, while those with code god patterns are at the bottom.

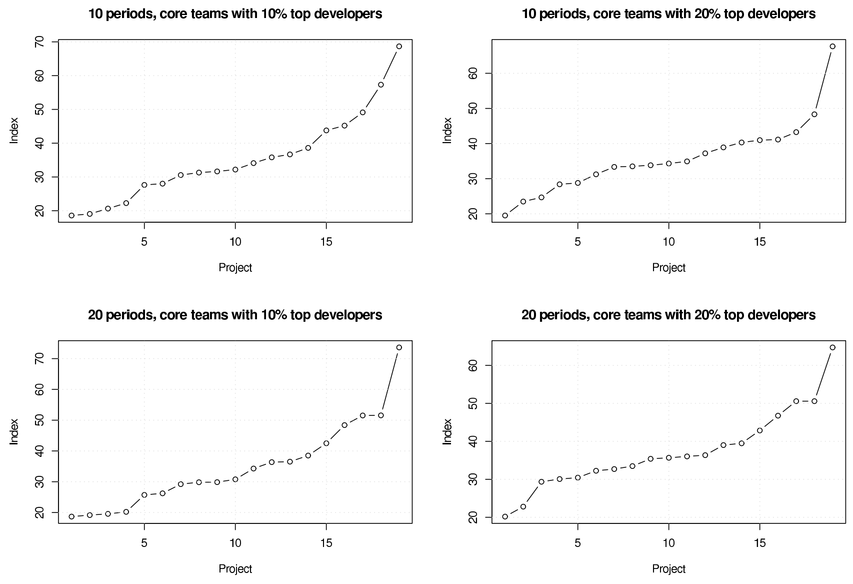


Figure 11. Plots of indexes for all the studied projects, using different numbers of periods and criteria for identifying core teams.

to which extent other, non-coding activities (such as discussion, writing of documentation, or even mediation between developers) should be considered to better identify the core team of developers. This should be the focus of further research.

Another open field for research is the use of the methodology in classical (non-libre) software projects. The fact that many developers in libre software projects are volunteers can provide very interesting information about the natural behavior of programmers, as these developers are self-selected (i.e., there is no traditional, mandatory task assignment as it can be found in the commercial world). In this regard, one of the findings that should be further researched is the amount of time for turnover. From our limited set of projects we have seen that, for those projects with several generations, the time span for a generation ranges from three to five years. This could be indicative for a programmers moving to a different project to keep his motivation and interest on his work high. Having developers enrolled in companies (such as the cases of Mozilla and Evolution) and volunteer developers in these projects could give further insight to this question in subsequent research.

The sample of projects considered in our study is small, which obviously opens opportunities for validating the methodology with a larger, and more diverse collection of projects. Of course, the larger the projects the more interesting the findings are (since it is difficult to understand all the details of such complex projects without the help of methodologies and tools). But even in the case of small projects some interesting results could be quickly and automatically obtained.

In any case, from our work we can conclude that the study of the behavior of human resources in libre software projects and in software engineering in general, and the relationship between its join/leave patterns and the evolution of the project, is a field worth to explore. This paper tries to be a first step in this direction, focused on studying its dynamics, and on finding how projects cope with the changes caused by it.

Acknowledgment

This work has been funded in part by the European Commission, through projects FLOSSMetrics, FP6-IST-5-033982, QUALOSS, FP6-IST-5-033547, and Qualipso, FP6-IST-034763.

References

[1] B. W. Boehm, Ed., *Software risk management*. Piscataway, NJ, USA: IEEE Press, 1989.

[2] D. M. Germán, “The GNOME project: a case study of open source, global software development,” *Journal of Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215, 2004.

[3] K. Crowston and J. Howison, “The social structure of free and open source software development,” *First Monday*, vol. 10, no. 2, February 2005.

[4] A. Mockus, R. T. Fielding, and J. D. Herbsleb, “Two case studies of Open Source software development: Apache and Mozilla,” *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[5] T. T. Dinh-Trong and J. M. Bieman, “The FreeBSD project: A replication case study of Open Source development,” *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, June 2005.

[6] C. Jensen and W. Scacchi, “Modeling recruitment and role migration processes in OSSD projects,” in *Proceedings of 6th International Workshop on Software Process Simulation and Modeling*, St. Louis, May 2005.

[7] G. Robles, J. M. González-Barahona, and M. Michlmayr, “Evolution of volunteer participation in libre software projects: evidence from Debian,” in *Proceedings of the 1st International Conference on Open Source Systems*, Genoa, Italy, July 2005, pp. 100–107.

[8] G. Robles and J. M. González-Barahona, “Contributor turnover in libre software projects,” in *Open Source Systems Conference, June 8-10, 2006, Como, Italy*, 2006, pp. 273–286.

[9] G. Robles, S. Koch, and J. M. González-Barahona, “Remote analysis and measurement of libre software systems by means of the CVSAnalY tool,” in *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, Edinburgh, Scotland, UK, 2004, pp. 51–56.

[10] G. Robles, J. M. González-Barahona, and J.-J. Merelo, “Beyond executable source code: The importance of other source artifacts in software development (a case study),” *Journal of Systems and Software*, vol. 80, no. 9, pp. 1233–1248, September 2006.

[11] R. A. Ghosh and V. V. Prakash, “The orbiten free software survey,” *First Monday*, vol. 5, no. 7, May 2000.

[12] S. Koch and G. Schneider, “Effort, cooperation and coordination in an open source software project: GNOME,” *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.