The Processes of Joining in Global Distributed Software Projects

Israel Herraiz, Gregorio Robles, Juan José Amor,
Teófilo Romera and Jesús M. González Barahona*
Universidad Rey Juan Carlos
Madrid, Spain
{herraiz, grex, jjamor, teo, jgb}@gsyc.escet.urjc.es

ABSTRACT

Libre (free / open source) software is a good example of global software development. Thousands of projects, in a wide range of domains which involve hundreds of thousands of developers and contributors from all around the world. Some large (both in size and developer community) libre software projects have shown evidence of producing code with complete functionality and fast evolution (with linear or superlinear growth), while maintaining low defect density. Many companies are exploring how to benefit from this situation, considering several approaches related to libre software development. For instance, some of them have hired full-time developers, focusing their work on some libre software projects they consider strategic.

However, before joining the core development team of the project, these hired developers have to follow a process of software comprehension, and get used to the rules and communication mechanisms used in the project. We were interested in the differences between this case and that of volunteer developers working in the same project, Therefore, we studied the duration and basic characteristics of this joining process for the developers of GNOME (a well known, large, libre software project). In our analysis, we have found two groups with clearly different joining patterns. Moreover, we have related those patterns to the different behaviors of volunteer and hired developers: volunteers tend to follow a step-by-step joining process, while hired developers usually experience a "sudden" integration. Some reasons for this different behavior are also discussed.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*programming teams, time estimation*; K.6.3 [**Management of computing and information systems**]: Software Management—*software development, software process*

General Terms

Management, Measurement, Human Factors

Keywords

global software development, libre software, membership integration, empirical studies, onion model

1. INTRODUCTION

Libre (free, open source)[1] software has gained a lot of attention from the public and the software engineering community during the last years. Libre software development processes seem to be better in the sense of fast growth [4] and low defect density [8]. Because of these and other reasons, some companies are interested in libre software projects, maybe allocating employees to contribute in a libre software project which is strategic for the company, maybe outsourcing the development and trying to spin off a new community around a product released as libre software.

We have focused on the first case, and have analyzed the situation in GNOME, a well known and studied [3, 7] libre software project. GNOME is sponsored by several companies via its foundation[2]. These companies are interested in the success of GNOME, and in most cases contribute to the development with full time employees.

In order to be able of improving and increasing the functionality (in other words, to be able of writing code), these hired developers must comprehend a large software artifact, and learn the communication mechanisms used by the project. As it is often found in libre software projects, the main communication stream in GNOME is its collection of mailing lists. Most of the feedback is managed through a bug tracking system (Bugzilla), and all the code is stored in a version control system (CVS, at the time of writing this paper, switching to Subversion). Therefore, these developers must read the source code of the module they are

*This work has been funded in part by the European Commission, under the CALIBRE CA, IST program, contract number 004337. Israel Herraiz has been supported in part by Consejeria de Educación of Comunidad de Madrid and European Social Fund, under grant number 01/FPI/0582/2005

GSD'06, May 23, 2006, Shanghai, China.

[1]Through out this paper we will use the term "libre software" to refer to any code that conforms either to the definition of "free software" (according to the Free Software Foundation) or "open source software" (according to the Open Source Initiative).
[2]See `http://foundation.gnome.org`.

interested in contributing to, and the corresponding mailing lists, and be aware of reported feedback in the bug tracking system. These activities leave some traces that we can measure and record for further analysis, being able of estimating the duration of the learning process (measured as the delay between the first contact with the project, and the first contribution to its code).

The structure of this paper is as follows. We start with a review of related work. After it, the reader can find a section presenting the hypothesis and research targets defined for the study, followed by another one describing the methodology and the data sources that have been used. Next, the results of the study are presented, and briefly discussed. The paper ends with a section with conclusions and further lines of research.

2. RELATED RESEARCH

To the knowledge of the authors there has not been any other paper that specifically targets joining processes and strategies in libre software projects from an empirical point of view, at least from the software engineering point of view. From the innovation research area we can nonetheless cite von Krogh et al. [11], which has targeted the joining and specialization process that occurs in libre software projects and how this affects innovation. The authors of this paper studied the case of the Freenet project (which develops a peer-to-peer system), and identified the tasks that developers usually fulfill when entering a project. For instance, they found that there exists an implicit "joining script" given by the level and type of activity, which makes the integration of new members more likely to be successful.

Probably the most well known model about the process of joining a libre software project is the 'onion model' [1], which represents how actors are positioned in communities as core developers, bug fixers and reporters, mailing lists contributors or plain users (see figure 1). Some studies have already reported and quantified this structure for several libre software projects ([8] for the Apache httpd server and the Mozilla web browser, [2] for the FreeBSD distribution). According to these studies the core is composed of a small group of about a dozen developers. Surrounding the core group there is a group of contributors, about an order of magnitude larger, that send bug fixes and eventually submit some code. Still an order of magnitude larger is the number of casual contributors who occasionally report bugs or perform other small tasks. Finally, we find the surrounding user community which may serve as a pool for future developers or casual contributors.

The onion model will be very important for our study, since we are going to characterize our results in accordance to it. But it lacks of a characteristic which is essential for our purposes: it is a static picture. It has been observed in libre software projects that the core group does not persist for long periods of time, but that there are new generations taking the lead over [5]. In this regard, the model has been complemented by Ye et al. with a more theoretical identification and description of the roles in it, adding dynamism [12] to the original model. According to this idea, a core developer is supposed to go through all those roles, starting as a user, until she eventually reaches the core.

Jensen and Scacchi [6] have also studied and modeled the processes of role migration for some libre software communities, focusing in the case of end-users who become developers for some projects selected as case studies. They have found different paths for the same process, and concluded that the organizational structure of the studied libre software projects are very dynamic in comparison to traditional software development organizations. In comparison to this work, they have identified a richer set of roles (for instance, "code sheriff" in the Mozilla community) that even include marketing and governance issues. In this paper we will focus only on development activities.

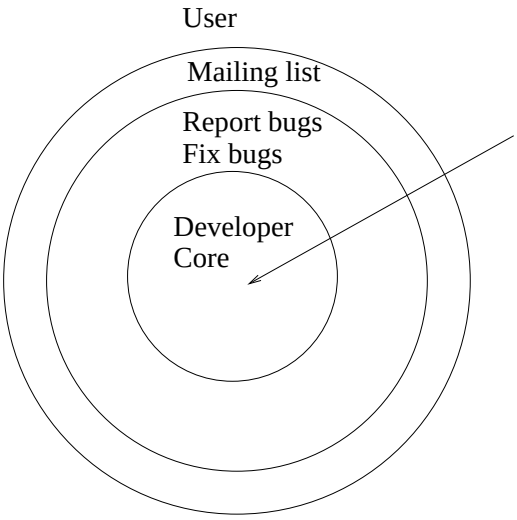

Figure 1: The Onion model

3. ASSUMPTIONS AND RESEARCH TARGETS

The base assumption of our study is that the process followed by a developer before making it to the core group, takes him through all the outer roles described by the onion model. This is a a trajectory that goes sequentially from outer positions (within the user base) to the most inner positions (in the core group). In other words, we suppose that:

- A developer starts as a passive user. At this stage she occasionally visits the project's web site, and maybe reads mailing lists. Since at this moment she will not submit any e-mail message nor bug reports, activity cannot be traced.
- If the developer becomes familiar to the project and can be classified as an advanced user, she may occasionally send some messages to mailing lists (which can be tracked). At this stage the program comprehension has started, since some understanding of the software is needed to participate in discussions.
- Having at least some partial knowledge of the project, she may start to report bugs or submit comments and later even fix some of them.
- If the contributions are meaningful, usually the next step for the developer is to be given an account with write access to code repository of the project through its version control system. The developer will then be able to commit code, activity which is also traceable.

- If the developer collaborates in the project with a high level activity during a certain amount of time, she may eventually be considered as a part of the core group.

Our main goal in this paper is to verify if the described sequence can be observed in real cases, and under which circumstances. For that, we will characterize that sequence by observing activity in mailing lists, bug tracking systems and version control repositories.

In addition, the empirical study is designed to quantify other aspects of this process. Specifically, this research was targeted to answer the following questions:

- Is the onion model a good representation of the joining process of a libre software developer?
- If it is, how long does it take to go through the different steps of integration?
- Do different patterns (different sequence or timing) exist?
- If different patterns are identified, how does this depend on the 'nature' of the developer? Is the integration process the same for volunteers and employees?

4. METHODOLOGY

Our study focus both on the whole population of developers of GNOME and on a sample of selected developers who have been studied with more detail.

For each user in the CVS (the popular version control system, also used in GNOME), we computed the dates of the first and last commit. Then we estimated the addresses she was using in the mailing list, and computed the dates of the first and last message in the mailing lists. Last we matched the CVS user with the user in the Bugzilla and we queried the dates of the first and last bug reports.

This was made for the whole population found in the CVS. But we also focused in a sample of selected developers who meet certain criteria (given below). For each developer in this sample we computed the same dates than for the whole population, but more carefully, making sure that all the e-mail addresses, and Bugzilla and CVS users corresponded to the studied developer. We also computed the dates of the first and last bug fixes[3].

The dates of the first instances of these events would be used to computed the progress metric (and thus the duration of the different stages in membership integration).

With this information we are able to verify whether the progress metrics comply with the progress predicted by the onion model, both for the whole population and for the selected sample. The results for the population are less reliable, as we only estimated the e-mail. Further ahead, for the selected sample, we also compute statistically this information trying to find common patterns, which when found are analyzed according to the characteristics of each developer (by getting information about them from members of the project or indirect evidences).

For the study we have selected the GNOME project, a desktop environment for UNIX-like systems which was launched in 1997. GNOME can be considered a large project, with more than 50M lines of code, and almost 800 modules in its CVS repository. Together with GNOME we have computed the GIMP, a popular image manipulation program which is in fact prior to GNOME and whose GTK+ tool kit was adopted later by GNOME. The interested reader can refer to [3] for a detailed description of the GNOME project as an example of global software development.

In the rest of this section we define how we have selected the target population and the sample of developers to be studied with more detail, and describe the data sources used for the research.

4.1 Selection of the target population and the sample of developers to be studied with more detail

Once we had chosen GNOME as case study, we queried the full list of users in the CVS, and selected this set as target population. From this population we extracted a sample to be studied with more detail. The developers in this sample fulfilled some criteria:

- The first is commit is not later than April 2001, in order to have enough data to study.
- The last commit was in year 2004, in order to select only currently active developers.
- The main contribution of the developer is code. We are interested in people who must comprehend a software artifact in order to make contributions to the project.
- For the same reason, the developer must not be the creator of the modules where she is writing, and must have started to write code in the module at least one year after its creation. Otherwise, the developer may not have had the need to comprehend a software artifact, but to build it from scratch.
- The developer must be very active in the last years. We decided to discard developers with less than 1000 commits for this reason.

4.2 Mailing lists

The data gathered from the mailing list archives has been parsed and dumped into a database. For each message we have recorded sender (name and e-mail address), subject, date and message id number. We have performed our analysis on the 109 mailing lists included in the GNOME and in the GIMP archive[4] and computed 464,953 entries in our database (one entry per message), written by 36,299 different posters (that is, with different e-mail addresses). The first message recorded was sent on May 30th 1996[5] and the last on November 16th 2004.

Once we had fed the database, we queried for the selected developers by name and by username in order to find the e-mail addresses they used (some of them had ten or more). To join information from mailing lists with the one given

[3]In our methodology we consider comments on bugs reported in the bug tracking system as fixes. Although this is not an accurate assumption it provides a further level of involvement by a developer which is enough for our purposes.

[4]The archives can be found at `http://mail.gnome.org/archives/` for GNOME and at `http://gimp.org/mail_lists.html` for the GIMP

[5]GNOME began on August 1997, but the first message was stored in the mailing lists of the GIMP, which began earlier than GNOME.

by the bug tracking system and CVS, we used some automatic identification methods for integrating data from various sources [9] as well as some non-automatic verification methods.

4.3 Bug Tracking system

For the analysis of the GNOME bug tracking system we retrieved all the bugs which are publicly available through the Bugzilla web interface[6] and obtained 123,739 bug reports submitted by 41,835 reporters with 382,271 comments authored by 10,580 contributors. Anonymous posts are not allowed in Bugzilla. The first bug dates from February 1999, while the last considered for this study was reported November 2004.

Due to the assignation of e-mail addresses accomplished for the analysis of the mailing list archives, the identification of developers in the bug tracking system was straightforward in most cases, both for the whole population and the extracted sample. In any case, this is a task which is not that difficult, since developers usually have a Bugzilla account that is uniquely bound to an e-mail address. It is used by the bug tracking system to notify the developers about new bugs or status changes on already existing bugs. So if a developer changes her e-mail address, all the entries performed by her will appear to have been done with the new e-mail address.

4.4 Version control system

For the analysis of CVS we used the CVSAnalY tool [10], which makes a detailed analysis of the logs of CVS repositories. In the case of GNOME we have identified 1,067 developers with write access to the CVS repository working on the 767 existing modules. The CVS repository opened in November 1997 and the date for the last commit considered in this study is from April 2004. During this time 2,006,162 commits on 184,729 files have taken place[7].

5. ANALYSIS AND RESULTS

Applying the aforementioned methodology we got first a population of 1067 developers. We could only obtain the dates for messages in the mailing for 754 developers, as the automatic identification of e-mail addresses was not successful for all the CVS user names. For the same reason, we could only obtain the date of the first bug report for a set of 609 developers.

With this set of 754 developers we obtained the differences between the dates of the first commit and the first message. In the table 1, a summary of the statistics of the time differences between the first commit and the first message is shown.

The same process was repeated for the time differences between the first bug report and the first commit, with the set of 609 developers. A summary of the statistics for these time differences is shown on table 2.

As the kurtosis values in tables 1 and 2 show, the probability distribution for each variable is different. In the case of the differences between bug reports and commits, it is a normal distribution; but not in the case of the differences between messages and commits. If we obtain the histogram for each variable, and plot them in the same figure, we obtain the figure 2. In that figure, it seems that the distribution of time differences between commits and messages is the result of the addition of, at least, two probability distributions (one with approximately zero mean, and other with approximately mean at 48 months). As the kurtosis value for this variable is different than zero, at least one of the distributions must be non-normal. We can also deduce that, taking into account that there at least two different distributions, there must be at least two groups of developers with different features. As we will show with the selected sample, we think that the main difference is that the two groups present different joining patterns into the project.

Mean	12.91
Standard Deviation	28.39
Variance	806.24
Kurtosis	37.29
Minimum	−73.37
Maximum	360.97
Count	754

Table 1: Summary of statistics for the time differences between the first commit and the first message (time in months)

Mean	−4.49
Standard Deviation	22.47
Sample Variance	504.77
Kurtosis	0.044
Minimum	−72.80
Maximum	65.07
Count	609

Table 2: Summary of statistics for the time differences between the first commit and the first bug report (time in months)

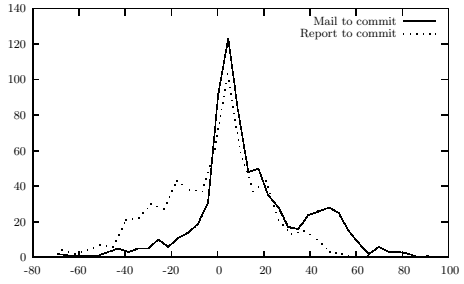

Figure 2: Distribution of time elapsed between events (horizontal axis) and number of developers (vertical axis). Time measured in months. It seems that the e-mail to commit distribution is the addition of at least two different probability distributions.

Regarding to the onion model, it seems that most of the developers have committed a change in the CVS before they ever sent a bug report (52.5% of developers with negative time difference), and so do not comply with the onion model

[6]See `http://bugzilla.gnome.org`

[7]The results of the whole CVS analysis done with CVSAnalY for GNOME and for other large libre software projects can be found at `http://libresoft.urjc.es/index.php?menu=Results`

in this aspect. In the case of the time difference between e-mails and commits, most of developers (75.6%) seem to comply with the onion model in this aspect, because their first contact with the project is an e-mail message to the mailing lists.

In general, the full population of developers does not comply with the onion model, as looking at the means of the two distributions, it shows that bug reports appear earlier than messages in the mailing lists (and it is supposed that sending a message requires less knowledge of the project). Moreover, the mean time elapsed between all the events (first bug report, first e-mail message and first commit) is short, although the standard deviation is of the same order than the mean and there are evidences of the existence of several groups of developers.

So, as it looks like there are two groups of developers, and the full population does not seem to comply with the onion model, a more profound analysis is needed. For this purpose we extracted the above mentioned sample, and repeated the same procedure (although we also included comments to bug reports, and supposed that if a developer writes a comment in a report, is due to a bug fix). This time, instead of identifying the e-mail addresses by automatic methods, we looked carefully for the e-mail addresses and Bugzilla user for each developer in the sample. We selected a first sample of developers who seemed to meet the selection criteria. However, we screened it carefully to discard some developers who in fact failed to comply with all the criteria. The analysis started by identifying common patterns, grouping the developers according to how similar their joining processes were. We also studied which of those patterns fitted the assumptions of the onion model, and tried to identify common characteristics and parameters of the different groups of developers found in the previous step.

From the 1,067 developers with write access to the GNOME CVS repository, after automatically confronting their patterns of activity in CVS, mailing lists and bug report system with the selection criteria, we got a set of 32 which could meet those criteria.

By further analyzing this set, identifying and screening developers one by one, we found that 12 of them did not really meet the selection criteria for several reasons. For instance, we found developers that had been active in the CVS earlier than the data suggested, because at that time they were using a different username; others that contributed to a module which had almost no activity during the first year; and even one which had not participated in Bugzilla, the bug tracking system of the project. This screening, therefore, lead us to a sample of 20 developers (see table 3).

Developer Nature	Number
Volunteers	8
Employees	8
University staff	4

Table 3: Nature of the developers in the sample

We found that we can classify the activity patterns of the 20 developers in two groups. The first one, composed of 7 developers, includes all who clearly followed the predicted sequence according to the onion model (this group will be referred from now on as "Group 1"). Table 4 gives the dates of the first contribution for each developer, and their global contribution to the mailing lists, version control system and bug tracking system. The main contribution of these developers is code (the mean percentage of code commits is 78.37% with a standard deviation of 16.83%).We can say that developers in this group follow a slow, gradual joining process. The first activity we track for any of them is an e-mail message. Later, they report a bug, at some point after that, they fix a bug. Finally they get access to the CVS repository and commit a change there. This process lasts usually between two and three years.

The second group, composed of 12 developers, has a pattern that does not fit the predicted sequence. On the contrary, they showed an almost simultaneous start-up in mailing lists, bug tracking system and CVS repository (which will be referred from now on as "Group 2").

Table 5 shows the date of first contributions in the various sources considered in this study as well as their total number. These developers are mainly coders (the mean percentage of code commits is 78.12% with a standard deviation of 13.91%). At the same time, developers in this group follow a rather fast process with sudden activity at all levels, usually completing the whole process in less than one year, and with no special ordering of first track of activity in mailing lists, bug tracking systems and CVS. Four of them begin their participation in all systems almost at the same time.

There is still a single developer (again, her main contribution is code, with a percentage of code commits of 67.21%) who shows a completely different joining pattern as it can be seen from table 6, which cannot be considered in any of the previous group. This seems to be a sort of "rara avis": it took her almost three years from its first message to a mailing list to the first commit to CVS. Two years later she began to use the Bugzilla system. We do not consider this developer in the subsequent analysis as we consider him as an exception.

To identify the two groups, we computed some progress metrics for each developer. Considering the date of their first message in a mailing list as the beginning of the process, we calculated the time elapsed until the first participations (report and fix) recorded in the bug tracking system, and until the first commit in the CVS repository. The aggregated results of these metrics for each group are shown in table 7.

With these results we can already answer the first question we proposed. For the given sample, Group 1 complies with the predictions of the onion model for the process of joining the project, while for Group 2 the pattern is clearly not compliant. Regarding to the mean time that the joining process takes, for the group 1, as table 7 indicates, the mean time is about 30 months (the time ellapse both from the first e-mail and first bug report to first commit). So, for a volunteer developer (at least, for the volunteer developers in the sample), it takes up to 30 months to gain enough knowledge to be able to write code in the project. The diagrams for the joining pattern of some of the developers of this group are shown in the figure 3.

For the group 2, the time needed to become integrated is much shorter: the developers perform their first commit much earlier. This indicates a faster integration process, which could be explained by developers being either hired by companies or being university staff. Supposedly, that would mean that they are experienced developers, and therefore already knowledgeable in the uses and processes of the

Developer	1st message	1st report	1st fix	1st commit	# commits	# messages	# bugs	# comments
1	01/09/99	03/11/01	03/11/01	12/01/01	1603	360	158	975
2	12/23/98	02/03/01	08/16/01	08/23/01	1982	141	59	993
3	09/08/99	07/18/01	05/21/01	06/19/00	973	138	32	407
4	12/03/99	04/12/01	03/13/01	05/01/01	8044	965	154	3508
5	01/01/97	02/18/01	02/18/01	02/06/01	6949	1499	18	123
6	06/05/98	04/24/99	10/29/99	04/29/99	9485	1214	558	2080
7	11/13/96	03/20/01	03/21/01	02/15/01	3720	163	26	367

Table 4: Some statistics about the selected developers (Group 1)

Developer	1st message	1st report	1st fix	1st commit	# commits	# messages	# bugs	# comments
8	01/07/01	09/13/01	06/06/01	04/06/01	2884	5529	459	17
9	07/14/00	02/11/01	02/22/02	05/17/01	5877	103	15	129
10	11/08/00	04/12/01	12/11/01	04/11/01	6830	618	235	1490
11	06/06/01	01/29/01	05/25/01	01/26/01	5384	1551	62	1963
12	01/26/00	02/22/01	05/23/00	03/04/00	6789	5189	35	3062
13	10/06/99	05/26/00	04/03/00	02/16/00	6421	115	85	75
14	06/09/99	06/15/00	01/27/01	06/30/99	29862	2464	48	4149
15	02/05/01	03/27/01	02/07/01	02/05/01	11250	6561	180	1670
16	09/30/99	07/30/01	12/03/00	03/11/99	37705	165	26	1331
17	06/15/98	12/13/01	04/14/00	01/16/99	9384	838	45	1088
18	08/31/99	05/23/00	11/12/00	10/21/98	6379	709	2	16
19	12/12/98	02/01/01	11/07/00	02/17/98	32489	2799	117	6951

Table 5: Some statistics about the selected developers (Group 2)

Name	Volunteers	Employees	Univ	Tot
Group 1	7	0	0	7
Group 2	0	8	4	12
Rest	1	0	0	1

Table 8: Groups by nature of the developers

Figure 3: Activity diagram for some developers in Group 1. For each, four bars are shown, each representing a period of activity (CVS, top; Bugzilla, those in the middle; mailing lists, bottom)

community, and capable of learning quickly. Maybe it also implies more confidence by their partner developers (since in GNOME, as in many other libre software projects, access to the CVS is granted once the developers perceive that the candidate has enough knowledge about the project). Therefore, the joining pattern in this group is much different than in group 1 (not following the onion model), as shown in figure 4.

The main difference in each group is the nature of developers (table 8). In the sample, there were 8 volunteers, 7 of them being part of group 1. All the hired developers and university staff are part of group 2. Thus, there is a clear conclusion that, for those developers in the GNOME project that achieve to become part of the group of core developers, with the restrictions already mentioned, only volunteers fit the progress predicted by the onion model.

6. CONCLUSIONS AND FURTHER WORK

We have studied the patterns found in the joining process of the whole population of GNOME developers. For them, we have shown that there is not a clear "common" joining pattern, and that its behavior does not seem tocomply with the predictions of the onion model. Moreover, as the probability distributions show, it seems that there are more than one group of developers with specific characteristics with respect to their comprehension process.

To obtain more detail about these groups, we selected a specific sample of core developers of the GNOME project: those who are highly active, mainly devoted to coding, and not starters of the modules to which they contribute. All of them had to comprehend a mature software artifact before they were able of contributing.

When we focused in this sample of core developers, we found significant differences between two different joining patterns: that of volunteers, which follow the onion model, and that of hired developers, which do not. Furthermore, hired developers get integrated into the project and gain enough knowledge to write code in it much faster than volunteers (whose integration process takes up to 30 months).

Developer	1st message	1st report	1st fix	1st commit	# commits	# messages	# bugs	# comments
20	11/13/96	07/18/01	02/24/01	09/02/99	5753	8673	6	19

Table 6: Some statistics about the selected developers (rest)

Samples	t to commit	sd. dev.	t to bug report	sd. dev.	t to bug fix	sd. dev.
Whole sample	13.49	19.86	21.92	18.29	22.42	16.48
Group 1	29.58	17.58	29.35	16.18	35.01	13.69
Group 2	0.19	6.75	12.57	12.52	13.11	8.727

Table 7: Progress metrics for each group (in months)

Figure 4: Activity diagram for some developers in Group 2

Therefore, in short, we can say that the onion model is followed only by the volunteer developers in our sample, but not by those working for the project as employees. However, for all the developers following the model, the observed joining pattern is quite similar.

Although the sample selected for the study could seem tiny at first sight, it is important to notice that it has been designed to cover one of the most interesting cases of developers joining a project. However, our selection criteria leaves outside very interesting cases, such as the developers of Ximian (now Novell), probably the company with most influence in the development of GNOME, because many of them belong to the group that was active in the first stages of the project. It would be of course worthwhile to design another sample including at least some of them. Those developers, in our opinion, are the best candidates to be hired for companies interested in the project, as they have been contributing since its beginning.

It is for sure also interesting to extend this study to other large, long-term libre software projects, such as KDE, Apache, etc, to find out whether the conclusions shown here are particular for the GNOME project or general for the libre software phenomenon.

Finally we think that companies can learn an important lesson from these results. When one is interested in contributing to a libre software project which is strategic for them, hired developers can gain enough knowledge of the project as to begin to contribute in a short time, even if they did not have a previous contact with it.

7. ACKNOWLEDGMENTS

We thank Carlos Perelló, who is part of the GNOME community, for his assistance and suggestions.

8. REFERENCES

[1] K. Crowston and J. Howison. The social structure of open source software development teams. In *Proceedings of the International Conference on Information Systems*, Seattle, WA, USA, 2003.

[2] T. Dinh-Trong and J. M. Bieman. Open source software development: A case study of FreeBSD. In *Proceedings of the 10th International Software Metrics Symposium*, Chicago, IL, USA, 2004.

[3] D. German. The GNOME project: a case study of open source, global software development. *Journal of Software Process: Improvement and Practice*, 8(4):201–215, 2004.

[4] M. W. Godfrey and Q. Tu. Evolution in Open Source software: A case study. In *Proceedings of the International Conference on Software Maintenance*, pages 131–142, San Jose, California, 2000.

[5] J. M. Gonzalez-Barahona and G. Robles. Unmounting the "code gods" assumption. Technical report, Universidad Rey Juan Carlos, 2003. `http://libresoft.urjc.es/html/downloads/xp2003-barahona-robles.pdf`.

[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] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[8] 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*, 11(3):309–346, 2002.

[9] G. Robles and J. M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. In *Proceedings of the International Workshop on Mining Software Repositories*, St. Louis, Missouri, USA, May 2005.

[10] G. Robles, S. Koch, and J. M. Gonzalez-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)*, Edinburg, Scotland, UK, 2004.

[11] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: A case study. *MIT Sloan Working Paper No. 4413-03*, 2003.

[12] Y. Ye, K. Nakakoji, Y. Yamamoto, and K. Kishida. The co-evolution of systems and communities in free and open source software development. In S. Koch, editor, *Free/Open Source Software Development*, pages 59–82. Idea Group Publishing, Hershey, PA, USA, 2004.