Using Social Network Analysis Techniques to Study Collaboration between a FLOSS Community and a Company

Juan Martinez-Romo[1], Gregorio Robles[2], Jesus M. Gonzalez-Barahona[2], and Miguel Ortuño-Perez[2]

[1] Dpto. Lenguajes y Sistemas Informaticos, E.T.S.I. Informatica (UNED)
juaner@lsi.uned.es
[2] Grupo de Sistemas y Comunicaciones, Universidad Rey Juan Carlos
{grex,jgb,mortuno}@gsyc.escet.urjc.es

Abstract. Because of the sheer volume of information available in FLOSS repositories, simple analysis have to face the problems of filtering the relevant information. Hence, it is essential to apply methodologies that highlight that information for a given aspect of the project. In this paper, some techniques from the social sciences have been used on data from version control systems to extract information about the development process of FLOSS projects with the aim of highlighting several processes that occur in FLOSS projects and that are difficult to obtain by other means. In particular, the collaboration between the FLOSS community and a company has been studied by selecting two projects as case studies. The results highlight aspects such as efficiency in the development process, release management and leadership turnover.

1 Introduction

Software projects are usually the collective work of many developers. In most cases, and especially in the case of large projects, those developers are formally organised in a well defined (usually hierarchical) structure, with clear guidelines about how to interact with each other, and the procedures and channels to use. Each team of developers is assigned to certain modules of the system, and only in rare cases they work outside their *territory*. However, this is usually not the case in FLOSS projects, where only loose (if any) formal structures can be recognised. On the contrary, FLOSS developers usually have access to any part of the software, and even in the case of large projects they can *move* more or less freely from one module to another with only some restrictions imposed by the common uses in the project, and the rules on which developers themselves have agreed. A large mount of spontaneous interaction structures arise, evolve and disappear without the intervention of a central control, yielding complex networks.

Among complex networks, social network analysis (SNA) appear as a method for analysing the structure and interactions of people and groups of people within extensive organisations. A vast amount of scientific works show applications of these techniques in fields that go from the social sciences to physics and computer networks [20, 10].

The goal of this paper is to apply classical social network concepts to data extracted from FLOSS projects, in particular to FLOSS projects that show a tight collaboration between a company and the FLOSS community. As, even in the presence of a company, FLOSS projects have a loose management style and are based heavily on third-party contributions, social network analyses may provide with some insight into the underlying processes that are responsible for the development. In this way, we have an ad-hoc network of interpersonal dependencies that come from the interactions between the nodes that integrate it. Therefore, this work is mainly descriptive and has the purpose of showing how the application of techniques of social networks can be useful in scopes beyond the original ones.

The rest of this paper is organised as follows. The next section describes related research on this topic. Next, the methodology designed to extract data from source code management systems and a series of technical aspects on which we have based our study is presented. Afterwards, a brief historical review of the main events within the projects will be given and then the results are shown and discussed. A final section with some conclusions and hints about further research closes the paper.

2 Related Research

Much attention in the area of research on FLOSS projects has been focused on the organisational structure of the projects [9, 17, 15], but little attention to the dynamics of the group of developers. A noteworthy contribution in this sense, although not directly addressing the evolution of developer communities, is the *onion model* [5], which shows how developers and users are positioned in communities. This model differentiates 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 [18, 7, 13].

The onion model provides only a static picture of a project, lacking the time dimension that is required for studying processes, among these the ones of joining and leaving a project. A more theoretical identification and description of the roles, including also some dynamics [26], has helped to fill the gap. According to this refinement, a core developer is supposed to go through all the outlying roles, starting as a user, until she eventually reaches the core team. Some research has tried to measure how long this process takes for a volunteer participant (i.e. somebody who is not hired by a company to work on the project), obtaining that the time that passes from the first e-mails to the mailing lists (considered as the first participation) to the first commit (considered as the developer becoming part of the core group) is in the mean of around 30 months [11]. An alternative approach has been proposed [12] after studying and modelling the processes of role migration for some FLOSS communities, focusing on end-users who become developers. This has lead to the identification of various paths for the joining process, concluding that the organisational structure of the studied projects is highly dynamic in comparison to traditional software development organisations. The attraction of human resources to FLOSS projects has also been analysed [6, 24], with different models proposed about how developers enter new projects.

With respect to abandonment, the number of Debian developers leaving the project has been studied [22], and how this affects its evolution (i.e. what happens to software packages that become unmaintained because of the abandonment). The authors propose a half-life parameter, defined as the time required for a certain group of contributors to fall to half of its initial population, which is of 7.5 years for the Debian project.

3 Methodology

3.1 Construction of the developers network

The methodology used in this study is based on retrieving data about the activity of developers from the source code management repository of the project [19, 27]. In the case of FLOSS projects, this usually means either a CVS or a Subversion repository, which is mined using CVSAnalY [23]. This tool retrieves the information about every commit to the repository, and inserts it into a database where it can be conveniently analysed. This information includes, for each commit (modification in a file in the repository): the date, the username of the developer (commiter), and the number of lines involved.

Once the information has been stored in the database, we proceed to construct the developer network. Each vertex represents a particular developer and two vertices will be linked by an edge when both they have contributed, at least, one common software artifact[1]. Edges may be weighted by means of a *weight of the relationship*, defined for instance, as the total number of commits performed by both developers on artifacts to which both have contributed. Therefore, a relation between two developers (vertices) exist when both have worked in the same artifact, and a link will exist between them (edge) whether both have made, at least, a *commit* in the same artifact.

3.2 Indexes

When the networks are constructed based on the previous definitions, and the degrees and costs of relationship have been calculated for linked nodes, we

[1] We will consider in this paper software artifacts at the file level, although others could be chosen, such as directories.

can apply standard SNA concepts to define a wide range of parameters of the network that can help characterising the network and its nodes [21]. Beyond classical parameters, others more sophisticated can be used to extract more specific information. This paper makes use of these parameters, calculated by means of Conan [14], which are introduced next:

- **Distance centrality** [25]: The distance centrality of a vertex, D_c, is a measurement of its proximity to the rest. It is sometimes called *closeness centrality* as the higher its value the closer that vertex is (on average) to the others. The distance centrality can be interpreted as a measurement of the influence of a vertex in a graph because the higher its value, the easier for that vertex to spread information through that network. Observe that when a given vertex is *far* from the others, it has a low degree of relationship (i.e. a high cost of relationship) with the rest.
  Research has shown that employees who are central in networks learn faster, perform better and are more committed to the organisation. These employees are also less likely to turn over. Besides, from the point of view of information propagation, vertices with high centrality are like *hills* on the plain, in the sense that any knowledge is put on them is rapidly seen by the rest and spreads easily to the rest of the organisation.
- **Betweenness centrality** [8, 2]: The betweenness centrality of a vertex, B_c, is a measurement of the number of shortest paths traversing that particular vertex.
  The betweenness centrality of a vertex can be interpreted as a measurement of the information control that it can perform on a graph, in the sense that vertices with a high value are intermediate nodes for the communication of the rest. In our context, given that we have weighted networks, multiple shortest paths between any pair of vertices are highly improbable. The betweenness centrality is just a measurement of the number of shortest paths traversing a given vertex.
  In the SNA literature vertices with high betweenness centrality are known to cover *structural holes*. That is, those vertices glue together parts of the organisation that would be otherwise far away from each other. They receive a diverse combination of information available to no one else in the network and have therefore a higher probability of being involved in the knowledge generation processes.
- **Coordination degree** [1, 16]: The coordination degree of a vertex, measures the ability of this vertex in a graph to interchange information. The coordination degree of a vertex, has a great importance inside our study, since it shows the ability of a certain node to receive information of the network and capture the activity in a project precisely.
  There are several manners to model this magnitude, but one of the easiest ways is to consider the coordination degree to be exponentially related to the distance between the vertices [16]. In this way, we define the coordination degree γ_{ij} between two vertices i and j as $\gamma_{ij} = e^{-\xi d_{ij}}$, where d_{ij} is the

distance between the two vertices and ξ is a real positive constant, measuring the strength of the relationship which we call the coordination strength.
Thus, we can define the total coordination degree of a vertex i in a graph as the sum of all the coordination degrees between that particular vertex and the rest. Namely, $\Gamma_i = \sum_{j=1}^{N} \gamma_{ij}$, where N is the order of the graph (the total number of vertices in that particular graph). The total coordination degree of a vertex is a measure of the amount of information that the vertex is able to receive belonging to that particular network.

– **Centrality Eigenvector** [4, 3]: Eigenvector centrality is a measure of the importance of a node in a network. It assigns relative scores to all nodes in the network based on the principle that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes.
Eigenvector centrality is defined as the principal eigenvector of the adjacency matrix defining the network. The defining equation of an eigenvector is $\lambda v = Av$ where A is the adjacency matrix of the graph, *lambda* is a constant (the eigenvalue) and V is the eigenvector. The equation lends itself to the interpretation that a node that has a high eigenvector score is one that is adjacent to nodes that are themselves high scorers.
The idea is that even if a node influences just one other node, who subsequently influences many other nodes, then the first node in that chain is highly influential. Hence, the eigenvector centrality may give good advice about leadership in the project under study.

4 Case studies: Evolution and Mono

We have selected two FLOSS projects in order to apply the social network analysis techniques on them and to show what kind of information we can extract with such a type of analysis. In both projects, there is a high involvement of a company (Ximian) that has assigned several employees to the development of the project and there is a surrounding FLOSS community whose contributions are always welcome.

4.1 Evolution

Evolution is the official personal information manager and work group information management tool for GNOME. It combines e-mail, calendar, address book, and task list management functions.

The origin of the project goes back to 1998, when it was started by some GNOME volunteers. But it is not until a company called Helix Code identified it as a strategic project that it did not win in importance. In October 1999, Helix Code became Ximian Inc. due to trademark problems. Evolution, then, was mainly developed by Ximian employees although in strong collaboration with the GNOME community for several years. Finally, Ximian was acquired by Novell in September 2004 and the maintenance of the then-mature Evolution tool was "outsourced" to Novell employees in India. Since then, contributions by the GNOME community are much lower, also in part because of the mature state of the tool.

There are other noteworthy events in the history of Evolution summarised in table 1.

Event	**Date**
First commits	early 1998
Ximian takeover	October 1999
Version 0.0	July 2000
Version 1.0	February 2001
Takeover by Novell	October 2003
Death of main developer	January 2004
Version 2.0	October 2004
Version 2.2	April 2005
Version 2.4	October 2005
Version 2.6	April 2006
Version 2.8	October 2006

Table 1. Most important events in the history of Evolution.

4.2 Mono

Mono is a project led by Novell (formerly by Ximian) to create an ECMA standard compliant .NET compatible set of tools, including among others a C# compiler and a Common Language Runtime. Mono can be run on Linux, FreeBSD, UNIX, Mac OS X, Solaris and Windows operating systems.

The dates of some of the most important events for Mono appear summarised in table 2.

5 Results

We have applied the SNA indexes to the two projects; results will be shown in this section. The samples have been taken from the first activity in each versioning system of the projects up to January 2007. Time slots of three months have been taken and two developers who have made at least one *commit* to any file within the same directory will be linked. This link will be weighted by the number of *commits* that both have in that time slot in the given directory. The interpretation of the following figures (each figure corresponds to a different measurement) is the following one: the horizontal axis is time, whereas the vertical z axis represents the result for a given measure. Therefore, each line in

Event	**Date**
First commits	mid-2001
First version of Gtk# checked into CVS	September 2001
Version 0.7	September 2001
Version 0.9	February 2002
GTK+ 2.2	December 2002
.NET Framework 1.1	April 2003
Version 1.0	June 2004
GNOME 2.10	March 2005
GTK+ 2.8	August 2005
.NET Framework 2.0	November 2005
Version 1.2	November 2006

Table 2. Most important events in the history of Mono.

the figure corresponds to the evolution over time of the values of a developer. We had to filter out less active developers as graphs that include all of them are hard to read. Hence, only the top 40 developers will be displayed.

5.1 Analysis of Evolution

If we analyse figure 1(a) representing the *coordination degree*, four phases can be clearly identified. An initial stage, until the takeover by *Ximian*, with a weak activity typical of a project in its early days. A second stage, until the *1.0 release*, where the work shows the highest gain in the history of the project, typical of the effort for developing a first stable version. Since then and until the takeover by *Novell*, a significant stagnation is originated due to a maintenance and debugging period. Finally, and because of the fact that a periodic release policy is introduced, peaks of work of two different dimensions can be observed alternating in time. The highest peaks correspond to new releases, with a high number of commits corresponding to new developments, translations and documentation. Meanwhile, the lows are due to stages of development and maintenance in which the kind of activity is different, being reflected in a smaller number of commits. It is therefore significant that time-based release management (i.e. a new version will be released every 6 months) seems to boost the development in general terms in comparison to feature-based release management (a new version will be made public when all features to be included are finished).

In addition to the own activity of a project, the *average coordination degree* of a network provides a measurement of the efficiency if we compare it with other values, such as the number of commits or the number of active developers. Figure 1(b) provides this comparison.

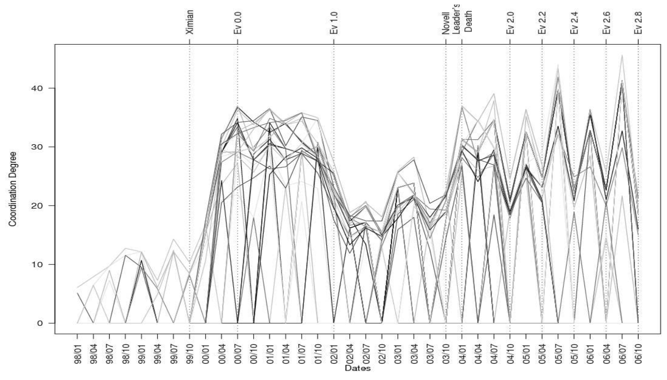
If we pay attention to the curves of the *average coordination degree* (cd mean) and the number of active developers (nodes), we can notice different moments at which one is over the other. The interpretation of these relative positions is the following: when the average coordination degree of the network is over the number of developers, the network structure is efficient. However, when the curve of the number of developers is over the average coordination degree, the social structure of the network is not efficient. In this way we can observe two stages with low efficiency in the network, one of them from the beginning of the project until the takeover by *Ximian*, the other one from the time when periodic releases where introduced (2004) until today. In between, since the takeover by *Ximian* in 1999 and up to the release of version *2.0* (in 2004), the network has an efficient structure. This evidences that a corporate involvement in the development of a FLOSS project may result in a more efficient development, but that both the involved company and the community have to find ways to allow the other party to participate properly. The first phase, where there was no involvement of the company, or the last one, where it seems that the company has chosen to move away from community have lead to an inefficient structure.

Figure 2(a) gives the evolution over time of the *betweenness centrality* for each developer. This figure is useful to identify who is the leader of the project at any given time, but also allows to predict who could be the following leader. If we attend the leadership curve in a particular moment, we can look for the curve just underneath. Usually, as the leader succession takes place gradually, we could figure out who is going to be the next leader of the project if the current one reduces his activity substantially.

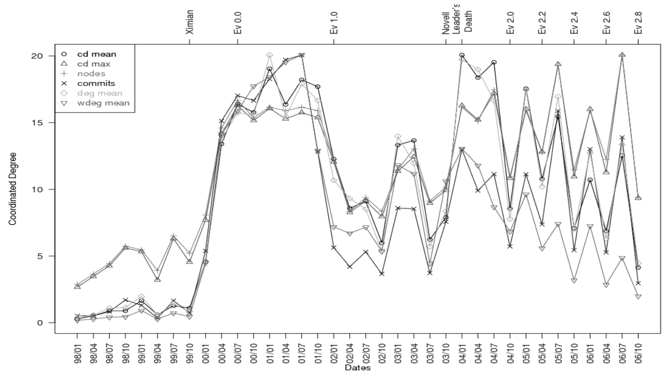
Figure 2(b) is a bar chart with the main leaders of the project for every time period. The measure that has been used is the *weighted betweenness* (the same as in Figure 2(a)). For the computation of this measure, we have to count the shortest paths that cross a vertex, so this measure is dependant on the number of developers in the network. For this reason, although the maximum values evolve at the same pace as the number of developers grows, it is an efficient measurement. From these results, three different stages can be observed. The two first stages in which "unammx" and "ettore" lead the project during several years, and one third stage since mid-2003 in which different leaders succeed themselves in short periods of time (all of them less than one year).

As noted before, the *eigenvalue* can be interpreted like a numerical assignment of the relevance of a node in the network. According to this interpretation, the higher the value of a given node, the higher its relevance. In figure 3(a) we can observe the evolution over time of this measure for each developer. At every moment, we can identify the main leader(s) of the project. If we compare the outline of the curve with the *coordination degree* or with the values of *betweenness*, it is possible to notice that the values of this measure do not depend on the number of developers, and therefore, the activity of the project is not reflected in the outline of the curves.

If we attend to the figure 3(b), we can find again the main leaders in every time slot, this time using the *eigenvalue.* If we compare these results with the ones obtained for the betweenness, we will see high similarities as expected. However, we must emphasise the differences in some time slots in which the



(a) Evolution of the Coordination Degree for Evolution



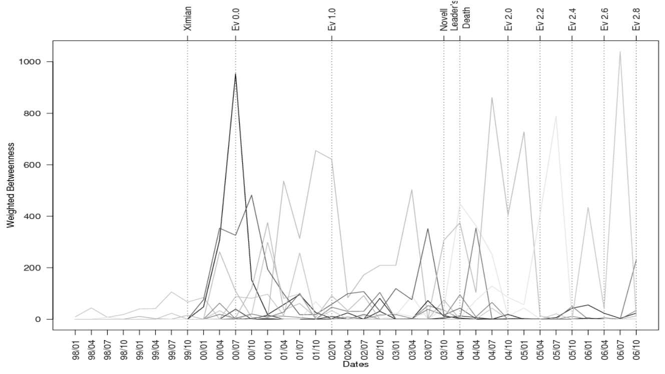
(b) Coordination Degree comparative for Evolution

Fig. 1. Coordination Degree values and a comparative for top 40 commiters in Evolution.

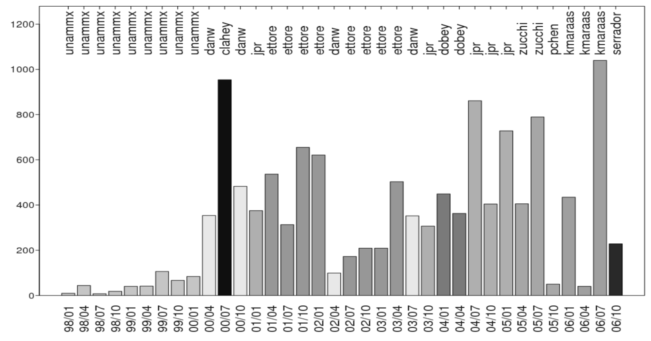
identified leader differs. This is because the *betweenness* focuses on the control of the information flows whereas the *eigenvalue* provides a node valuation from the point of view of centrality. Thus, the periods where leaders remain the main contributors to a project are smaller for the second value, because a greater effort has to be carried out to keep this privileged position. In the same way, the number of different persons who achieve the leadership is higher using *eigenvalues*. So, we can see that gaining control over information flows in the project is an easier task than obtaining a high influence on the network.

5.2 Analysis of Mono

The activity of Mono is marked clearly by five events that can be appreciated in Figure 4(a). The first one is the *Mono 0.7* release (first unstable version) that additionally happens at the same time as the initial check-in of the *GTK#*



(a) Evolution of the Weighted Betweenness for Evolution

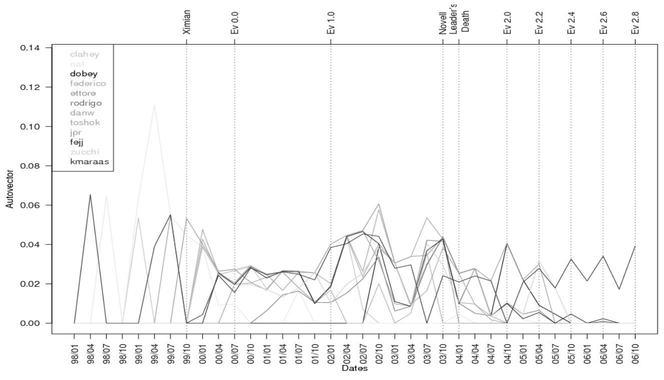


(b) Weighted Betweenness comparative for Evolution

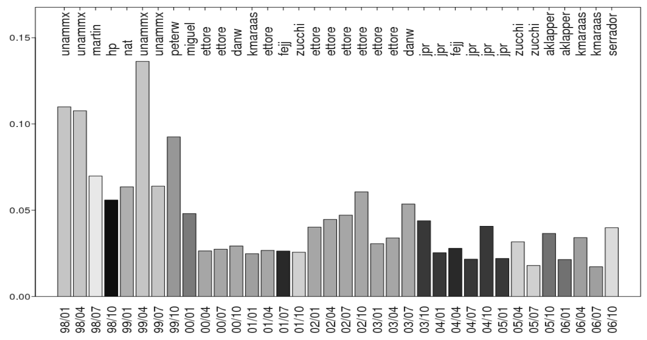
Fig. 2. Weighted Betweenness values and a comparative for top 40 commiters in Evolution.

module in the repository. Then a progressive increase of the activity took place, reaching its highest peak in the middle of 2003. An external fact caused an extraordinary activity: the *.NET1.1* release, which Mono had to react to be able to support this specification in its next version. This adaptation was completed in *Mono 1.0* in 2004. In 2005 another noteworthy moment took place, when the Mono team started to adapt the *.NET 2.0* specifications.

The development of Mono does not have a continuous growth, but it shows momentary efforts to achieve the *.NET* functionality. This kind of development is reflected in its social structure as it is not efficient most of the time. As it can be observed from Figure 4(b) while the number of developers grows smoothly, their activity has several peaks. In any case, compared to the previous case study, we can conclude that in this case Ximian has not achieved such a high community involvement in Mono as in Evolution.



(a) Evolution of the Eigenvalue for Evolution

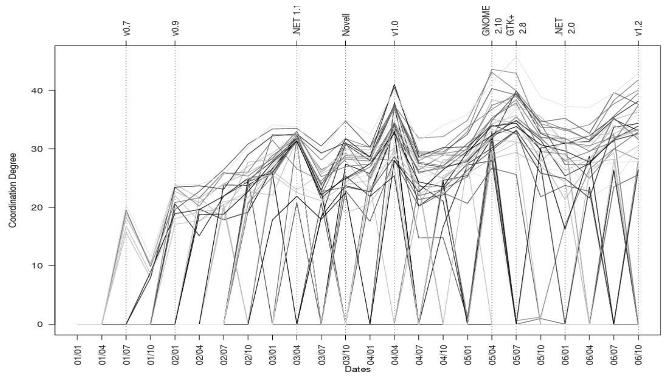


(b) Eigenvalue comparative for Evolution

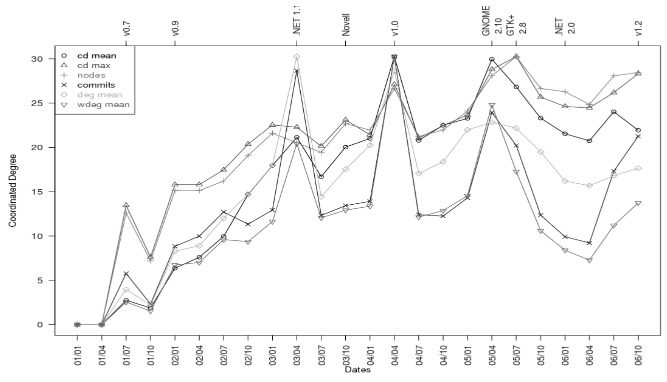
Fig. 3. Eigenvalue values and a comparative for top 40 commiters in Evolution.

The Mono project is composed currently of over 80 developers. As it can be seen from Figure 5(a), and a large difference between the values of the main leader and the ones for the rest of developers exist. The yellow line belongs to Miguel de Icaza, the founder of the project, while the red one to Gonzalo Paniagua, a Novell-hired developer, and the green lines correspond to Raja R. Harinath and Ben Maurer, also Novell employees. Miguel de Icaza is clearly the main leader in Mono as he has the maximum value in 16 quarters, in the early stages and also recently. Gonzalo Paniagua had a more prominent role in the first half of the project as well as in the last phases.

If we use *eigenvalues* (see Figure 6(a)), the difference between the project leaders and the rest of developers are not that high as found using *betweenness*. In any case, if we compare Figure 6(b) with figure 5(b) the main leaders continue being the same. Even though, eigenvalue helps us obtaining additional information on the development community as it allows to rank developers by their activity and their position in the network.



(a) Evolution of the Coordination Degree for Mono



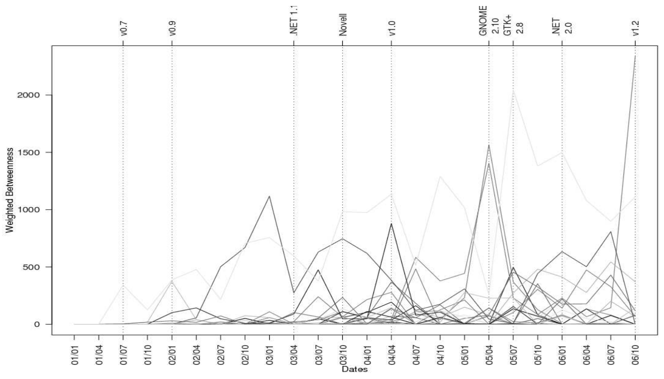
(b) Coordination Degree comparative for Mono

Fig. 4. Coordination Degree values and a comparative for top 40 commiters in Mono.

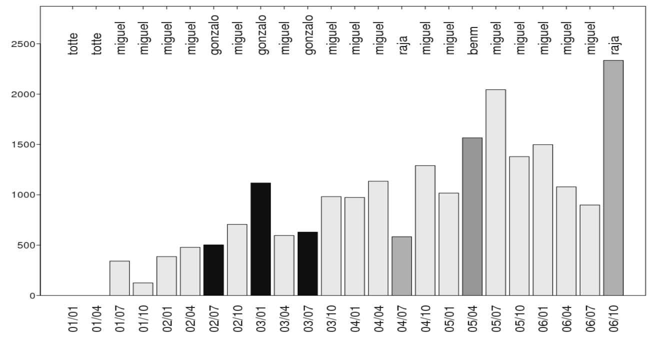
6 Conclusions and further work

In this paper a descriptive study of several social networks of FLOSS projects where a company and the community collaborate has been presented. We have seen that although there is access to the interactions that developers have in the source code management system, the amount of information is that high that we require additional methods to properly analyse the community and extract facts and information from it.

Thanks to the use of social networks analysis techniques we have been able to obtain some information on these projects, in particular, that time-based release management seems to animate the development of projects or that the collaboration between the company and the community can, if properly handled, drive a more efficient development. Future research should be devoted to analyse the differences between the studied projects to try to find out the aspects that drive one of the case studies (Evolution) to have at least for some



(a) Evolution of the Weighted Betweenness for Mono



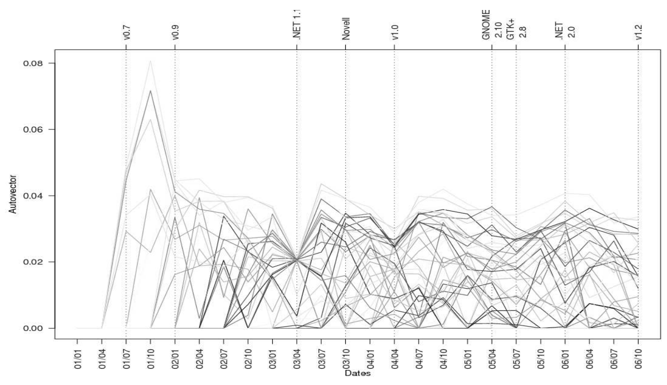
(b) Weighted Betweenness comparative for Mono

Fig. 5. Weighted Betweenness values and a comparative for top 40 commiters in Mono.

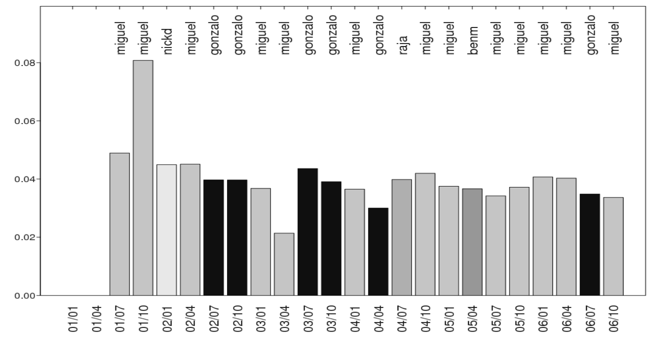
time an efficient development, while this is an exception for the other project (Mono) under study.

In addition, we have identified the most important nodes (developers) in the project and have observed them from from several points of view (leadership, information control flows, etc.) and how they evolve over time. In both case studies, the most prominent positions where held by company employees, showing a low turnover over the years. Future work could be devoted to find out if this behaviour is similar in community-led projects.

All in all, this paper demonstrates that the techniques from the social networks analysis field can be of great interest for the study and characterisation of any kind of network and with particular interest in the study of how the software industry can interact with the FLOSS community entering the development of projects.



(a) Evolution of the Eigenvalue for Mono



(b) Eigenvalue comparative for Mono

Fig. 6. Eigenvalue values and a comparative for top 40 commiters in Mono.

7 Acknowledgements

This work has been funded in part by the European Commission, under the QUALIPSO (FP6-IST-034763), FLOSSMETRICS (FP6-IST-5-033547) and QUALOSS (FP6-IST-5-033547) projects, and by the Spanish CICyT, project SobreSalto (TIN2007-66172).

We would like to thank the QualiPSo Activity 6 partners for comments and suggestions on this research.

References

1. J. A. Almendral, L. Lopez, and M. A. F. Sanjuan. Information flow in generalized hierarchical networks. *Physica A Statistical Mechanics and its Applications*, 324:424–429, June 2003.

2. Jac M. Anthonisse. The rush in a directed graph. Technical report, Stichting Mathemastisch Centrum, Amsterdam, The Netherlands, 1971.
3. Phillip Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2:113–120, 1972.
4. Stephen P. Borgatti. Centrality and network flow. *Social Networks*, 27(1):55–71, January 2005.
5. Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 10(2), February 2005.
6. Jean-Michel Dalle and Paul A. David. The allocation of software development resources in open source production mode. Technical Report SIEPR Discussion Paper No. 02-27, Stanford Institute for Economic Policy Research, February 2003.
7. Trung T. Dinh-Trong and James M. Bieman. The FreeBSD project: A replication case study of Open Source development. *IEEE Transactions on Software Engineering*, 31(6):481–494, June 2005.
8. Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry 40, 35-41*, 1977.
9. Daniel M. 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.
10. Roger Guimera, Albert Diaz-Guilera, F. Vega-Redondo, A. Cabrales, and Alex Arenas. Optimal network topologies for local search with congestion. *Physical Review Let. 89, 248701*, 2002.
11. Israel Herraiz, Gregorio Robles, Juan JosÉ Amor, Teófilo Romera, and Jesús M. González Barahona. The processes of joining in global distributed software projects. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 27–33, New York, NY, USA, 2006. ACM Press.
12. Chris Jensen and Walter 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.
13. Stefan Koch and Georg Schneider. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal*, 12(1):27–42, 2002.
14. Luis Lopez and Juan Martinez-Romo. Conan: Generating social network analysis. `http://tools.libresoft.es/conan`.
15. Luis Lopez, Gregorio Robles, Jesus M. Gonzalez Barahona, and Israel Herraiz. Applying social network analysis techniques to community-driven libre software projects. *International Journal of Information Technology and Web Engineering*, 1(3):27–48, July-September 2006.
16. Luis Lopez and Miguel A. F. Sanjuan. Relation between structure and size in social networks. *Phys. Rev. E*, 65(3):036107, Feb 2002.
17. Gregory Madey, Vincent Freeh, and Renee Tynan. Modeling the Free/Open Source software community: A quantitative investigation. In Stefan Koch and Stefan Koch, editors, *Free/Open Source Software Development*, pages 203–221. Idea Group Publishing, Hershey, Pennsylvania, USA, 2004.
18. Audris Mockus, Roy T. Fielding, and James 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.

19. Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130, October 2000.
20. Mark E. J. Newman. Scientific collaboration networks: I. network construction and fundamental results. *Phys. Rev. E 64, 016131*, 2001.
21. Gregorio Robles. *Empirical Software Engineering Research on Libre Software: Data Sources, Methodologies and Results*. PhD thesis, Escuela Superior de Ciencias Experimentales y Tecnologia, Universidad Rey Juan Carlos, 2006.
22. Gregorio Robles, Jesus M. Gonzalez-Barahona, and Martin Michlmayr. Evolution of volunteer participation in libre software projects: evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems*, pages 100–107, Genoa, Italy, July 2005.
23. Gregorio Robles, Stefan Koch, and Jesus 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)*, pages 51–56, Edinburgh, Scotland, UK, 2004.
24. Gregorio Robles, Juan Julian Merelo, and Jesus M. Gonzalez-Barahona. Self-organized development in libre software: a model based on the stigmergy concept. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim 2005)*, St.Louis, Missouri, USA, May 2005.
25. Gert Sabidussi. The centrality index of a graph. *Psychometirka 31, 581-606*, 1996.
26. Yuwan Ye, Kumiyo Nakakoji, Yasuhiro Yamamoto, and Kouichi Kishida. The co-evolution of systems and communities in Free and Open Source software development. In Stefan Koch and Stefan Koch, editors, *Free/Open Source Software Development*, pages 59–82. Idea Group Publishing, Hershey, Pennsylvania, USA, 2004.
27. Thomas Zimmermann, Peter Weissgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, June 2005.