Applying Social Network Analysis Techniques to Community-driven Libre Software Projects

Luis López-Fernández, Gregorio Robles, Jesus M. Gonzalez-Barahona and Israel Herraiz
Grupo de Sistemas y Comunicaciones, Universidad Rey Juan Carlos
{llopez,grex,jgb,herraiz}@gsyc.escet.urjc.es[2]

November 2005

Abstract

Source code management repositories of large, long-lived libre (free, open source) software projects can be a source of valuable data about the organizational structure, evolution and knowledge exchange in the corresponding development communities. Unfortunately, the sheer volume of the available information renders it almost unusable without applying methodologies which highlight the relevant information for a given aspect of the project. Such methodology is proposed in this paper, based on well known concepts from the social networks analysis field, which can be used to study the relationships among developers and how they collaborate in different parts of a project. It is also applied to data mined from some well known projects (Apache, GNOME, and KDE), focusing on the characterization of their collaboration network architecture. These cases help to understand the potentials of the methodology and how it is applied, but also show some relevant results which open new paths in the understanding of the informal organization of libre software development communities.

Keywords: Mining software repositories, software understanding, social networks analysis, community-driven development

Introduction

Software projects are usually the collective work of many developers. In most cases, and specially in the case of large projects, those developers are formally organized 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 certain modules of the project, and only in rare cases they work outside that realm. However, this is usually not the case with libre software[1] projects, where only loose (if any) formal structures are acknowledged. On the contrary, libre software developers usually have access to any part of the software, and even in the case of large projects they can move freely to a certain extent from one module to other, with only some restrictions imposed by common usage in the project and the rules on which developers themselves have agreed to.

In fact, during the late 1990s some voices started to claim that the success of some libre software projects was rooted in this different way of organization, which was referred to as the "bazaar development model", described by Eric Raymond (Raymond, 1997) and later complemented by some more formal models of non-hierarchical coordination (Elliott and Scacchi, 2004, Healy and Schussman, 2003). Some empirical studies have found that many libre software projects cannot follow this bazaar-style model, since they are composed of just one or two developers (Krishnamurthy, 2002, Healy and Schussman, 2003), but the idea remains valid for large projects, with tens or even hundreds of developers, where coordination is obviously achieved, but (usually) not by using formal procedures. These latter cases have gained much attention from the software engineering community during the last years, in part because despite apparently breaking some traditional premises (hard-to-find requirement studies, apparently no internal structure, global software development, etc.) final products of reasonable quality are being delivered. Large libre software projects are also *suspicious* of breaking one of the traditional software evolution *laws*, showing linear or even super-linear growth even after reaching a size of several millions of lines of

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

[1]In this paper the term "libre software" will be used to refer to any software licensed under terms that are compliant with the definition of "free software" by the Free Software Foundation, and the definition of "open source software" by the Open Source Initiative, thus avoiding the controversy between those two terms.

code (Godfrey and Tu, 2000, Robles et al., 2005a). The *laws* of software evolution state that the evolution of a system is a self-regulating process that maintains its organizational stability. Thus, unless feedback mechanisms are appropriately introduced, the effective global activity tends to remain constant and incremental growth declines. The fact that several studies on some large libre software projects show evidence that some of these *laws* are disobeyed may be an indicative of an efficient organizational structure.

On the other hand, the study of several large libre software projects has shown evidences about the unequal distribution of the contributions of developers (Mockus et al., 2002, Koch and Schneider, 2002, Dinh-Trong and Bieman, 2005). These studies have identified roles within the development community, and have discovered that a large fraction of the development work is done by a small group of about 15 persons, which has been called the ”core” group. The number of developers is around one order of magnitude larger, and the number of occasional bug reporters is again about one order of magnitude larger than that of developers (Mockus et al., 2002, Dinh-Trong and Bieman, 2005). This is what has been called the *onion* structure of libre software projects projects (Crowston et al., 2003). In this direction, it has also been suggested that large projects need to adopt policies to divide the work, giving rise to smaller, clearly defined projects (Mockus et al., 2002) . This trend can be observed in the organization of the CVS[2] repository of really large libre software projects, where the code base is split into modules with their own maintainers, goals, etc. Modules are usually supposed to be built maintaining the interrelationships to a minimum, so that independent evolution is possible (Germán, 2004).

In this paper, a new approach is explored in order to study the informal structure and organization of the developers in large libre software projects. It is based on the application of well known social networks analysis (SNA) techniques to development data obtained from the versioning system (CVS). According to the classical Conway’s *law*, organizations designing systems are constrained to produce designs which are copies of their communication structures (Conway, 1968). Following this line of reasoning, the relationships among modules will be studied, and the dual case of those among developers. Our target is the advancement of the knowledge about the informal coordination structures that are the key to understanding how these large libre software projects can work in the apparent absence of formalized structures, and where are the limits of those ways of coordinating and exchanging information. We have designed a methodology following this approach, and have also applied it to some well known projects. Although the aim of our approach is mainly descriptive, not proposing novel models for project evolution or agent behavior, just trying to describe in as much detail as possible the organizational structure of libre software projects, our work is illustrative of the power of the SNA techniques. To attain this goal, our approach is similar to that presented in (Madey et al., 2002, Xu et al., 2005): we consider libre software projects as complex systems and characterize them by using mathematical formalisms. As a result, some interesting facts related to the organizational structure of libre software projects have been uncovered.

The remainder of this paper is organized as follows. The next section contains a basic introduction to SNA, and how we pretend to apply its techniques to the study of libre software projects based on the data available in their CVS repositories. Section Error: Reference source not found specifies in detail the methodology for such a study, followed by section Error: Reference source not found with a brief introduction to a set of classical social network analysis parameters. After that, section Error: Reference source not found presents the main characteristics of the networks corresponding to the three projects used as case examples: Apache, GNOME and KDE. This serves as an introduction to the more detailed comments on several aspects of those projects, presented in sections Error: Reference source not found, Error: Reference source not found, Error: Reference source not found, Error: Reference source not found, and Error: Reference source not found. At the end, section Error: Reference source not found offers some conclusions, comments some related work, and discusses further lines of research.

Application of SNA to libre software projects

The study and characterization of complex systems is a fruitful research area, with many interesting open problems. Special attention has been paid recently to complex networks, where graph and network analysis play an important role. This approach is gaining popularity due to its intrinsic power to reduce a system to its single components and relationships. Network characterization is widely used in many scientific and technological disciplines, such as neurobiology (Watts and Strogatz, 1998), computer networks (Albert et al., 2000) or linguistics (Kumar et al., 2002).

[2] Concurrent Version System (CVS) is the source code management (also known as versioning) system used in most libre software projects, although lately a new generation of tools, including for instance Subversion, is gaining popularity. In those projects, the CVS repository is usually freely readable over the Internet.

Although some voices argue that the software development process found in libre software projects is hardly to be considered as a new development paradigm (Fuggetta, 2003), without doubt the way it handles its human resources differs completely from traditional organizations (Germán, 2004). In both cases, traditional and libre software environments, the human factor is of key importance for the development process and how the software evolves (Gîrba et al., 2005), but the volunteer nature of many contributors in the libre software case makes it a clearly differentiated situation (Robles et al., 2005b).

Previous research on this topic has both attended to technical and organizational points of view. German used data from a versioning repository in time to determine feature-adding and bug-correcting phases. He also found evidence for developer territoriality (software artifacts that are mainly, if not uniquely, *touched* by a single developer) (German, 2004).

The intention of other papers has been to uncover the social structure of the underlying community. The first efforts in the libre software world are due to Madey et al. (Madey et al., 2002), who took data from the largest libre software projects repository, SourceForge.net, and inferred relationships among developers that contributed to projects in common. A statistical analysis of some basic social network parameters can also be found by López et al (Lopez et al., 2004) for some large libre software projects. Xu et al. (Xu et al., 2005) have presented a more profound topological analysis of the libre software community, joining in the same work characteristics from previous articles: data based on the SourceForge platform and a statistical analysis of some parameters with the goal of gaining knowledge on the topology of the libre software phenomenon. This has also been the intention of González-Barahona et al. in (González-Barahona et al., 2004) where a structure-finding algorithm was used to obtain the evolution in time of the organization of the Apache project. Wagstrom et al. (Wagstrom et al., 2005) propose to use the knowledge acquired from analyzing libre software projects with SNA for the creation of models that help understand the underlying social and technical process.

Methodology

The first problem to solve when using SNA is getting the information to construct the network to analyze. One specially interesting kind of data sources are the records maintained by many computer-based systems. For instance, (Guimera et al., 2003) analyzes informal networking on organizations using tracks of e-mail exchanges. Therefore, from the many kinds of records available about the activity of a libre software project, those provided by the CVS system where source code is stored have been the ones chosen. Those records offer information about who modified the code, and when and how, in many cases from the very beginning of the project, in some cases over a total period of time above 10 years.

The information in the CVS repository of a project includes an accurate and detailed picture of the organizational structure of the software, and of the developers working on it. When two developers work on the same project module, they have to exchange (directly or maybe indirectly) information and knowledge to coordinate their actions and produce a working result. It seems reasonable to assume that the higher their contributions to the module, the higher the strength of their informal connection.

Based on this assumption, a specific kind of social networks has been considered, those called affiliation networks. They are characterized by showing two types of vertices: *actors* and *groups*. When the network is represented with actors as vertices, each one is usually associated with a particular person, and two of them are linked together when they belong to the same group. When the network is represented with groups as vertices, two groups are connected when there is, at least, one actor belonging, at the same time, to both groups. In our case, actors will be identified as developers, and groups as software modules. The "belong to" relationship will be in fact "has contributed to". This approach will result in a dual view of the same organization: as a network of modules linked by common developers, and as a network of developers linked by common modules. Similar approaches have been used for analyzing other complex organizations, like the network of scientific authors (Newman, 2001a, Newman, 2001b) or the network of movie actors (Albert and Barabasi, 2002).

To finish the characterization of our networks, weighted edges are being considered. This means that it is not only taken into account whether a node has some relationship with any other, but also the strength of that relationship. In our case, the weight will be related to the size of contributions to common modules (in the case of developers) and to the size of contributions by common developers (in the case of modules). It should be noted that from the methodological point of view, the use of weights is a major contribution of this paper in comparison with previous works describing SNA techniques applied to libre software (Madey et al., 2002, Xu et al., 2005, Wagstrom et al., 2005). As we will see in this paper, the use of weights is indicated as the distribution of work follows a very unequal distribution, in the range of a Pareto distribution[3] (Ghosh and Prakash, 2000). Our assumption at this point is that considering a link between two major contributing developers that equals the one between two random chosen developers, introducing an important bias in the results regarding the distribution of work observed in libre software environments.

Once we have identified how we want to use SNA for libre software projects, a well defined methodology is propposed in order to apply those ideas to any libre software with a public CVS repository. The process begins by downloading the relevant information from the CVS repository[4] . This information includes, for each commit (modification in a file in the repository): the date, the identifier of the developer (commiter), and the number of lines involved. Using all those records, the following networks are defined for characterizing the organization of the project:

- **Modules network**. Each vertex represents a particular software module (usually a directory in the CVS repository) of the project. Two modules are linked together by an edge when there is at least one commiter who has contributed to both. Those edges are weighted using a *degree of relationship* between the two modules, defined as the total number of commits performed by common commiters.
- **Commiters network**. In this case, each vertex represents a particular commiter (developer). Two commiters are linked by an edge when they have contributed to, at least, one common module. Again, edges are weighted by a *degree of relationship* defined as the total number of commits performed by both developers on modules to which both have contributed.

The definition being used for the *degree of relationship* is an attempt to measure the *closeness* of two vertices. The higher this parameter, the stronger the relationship between those vertices. In this sense, *cost of relationship* between any two vertices can also be defined as the inverse of their *degree of relationship*. In this sense, the *cost of relationship* defines a distance between vertices: the higher it is, the more difficult it is to reach one of them from the other. More formally, given a (connected) graph *G* and a pair of vertices *i* and *j*, we define the distance between them as $d_{ij} = \sum_{e \in P_S} c_r$, where *e* are all the edges in the shortest path from *i* to *j*, and is the *cost of relationship* of any of those edges.

Parameters

Once the networks are constructed based on the previous definitions, and the degrees and costs of relationship have been calculated for linked nodes, standard SNA concepts can be applied in order to define the following parameters of the network (the interpretation of the main implications of each parameter is also offered):

- **Degree**. The degree, *k*, of a vertex is the number of edges connected to it. In SNA, this parameter reflects the popularity of a vertex, in the sense that most popular vertices are those maintaining the highest number of relationships. More revealing than the degree of single vertices is the distribution degree of the network (the probability of a vertex having a given degree). This is one of the most relevant characterizations because it provides essential information to understand the topology of a network (and if longitudinal data is available, the evolution of the topology). For example, it is well known that a random network follows a Poisson's distribution, while a network following a preferential attachment growth model presents a power law distribution (Albert and Barabasi, 2002). In our context, the degree of a commiter corresponds to the number of other commiters sharing modules with her, while the the degree of a module is the total number of modules with which it shares developers.
- **Weighted degree**. When dealing with weighted networks, the degree of a vertex may be tricky. A vertex with a high degree is not necessarily well connected to the network because all its edges may be weak. On the other hand, a low degree vertex may be strongly attached to the network if all its links are heavy. For this reason the weighted degree of a vertex, , is defined as the sum of the weights of all the edges connected to it. The weighted degree of a vertex can be interpreted as the maximum capacity to receive information of that vertex. It is also related to the effort spent by the vertex in maintaining its relationships.
- **Clustering coefficient** (Watts and Strogatz, 1998). The clustering coefficient, *c*, of a vertex measures the transitivity of a network. Given a vertex *v* in a graph *G*, it can be defined as the probability that any two neighbors of *v* are connected (the neighbors of *v* are those vertices directly connected to *v*). Hence

[3]A Pareto distribution is known to be given when the 20% most active is responsible for 80% of the output.
[4] For downloading this information we have used the CVSAnalY tool described in (Robles et al., 2004)

$$c(v) = \frac{2E(v)}{k_v(k_v - 1)} \qquad (1)$$

where is the number of neighbors of *v* and *E*(*v*) is the number of edges between them. The intuitive interpretation of the clustering coefficient is somehow subtle. If the total number of neighbors of *v* is , the maximum number of edges than can exist within that neighborhood is $k_v(k_v - 1)/2$. Hence, the clustering coefficient represents the fraction of the number of edges that really are in a neighborhood. Therefore it can be considered as a measurement of the tendency of a given vertex to promote relationships among its neighbors. In a completely random graph, the clustering coefficient is low, because the probability of any two vertices being connected is the same, independently on them sharing a common neighbor. On the other hand, it has been shown that most social networks present significantly high clustering coefficients (for instance, the probability of two persons being friends is not independent from the fact that they share a common friend) (Albert and Barabasi, 2002, Watts, 2003).

From an organizational point of view, the clustering coefficient helps to identify hot spots of knowledge exchange on dynamic networks. When this parameter is high for a vertex, that vertex is promoting its neighbors to interact with each other. Somehow it is fostering connections among its neighborhood. High clustering coefficients in networks are indicative for *cliques*. Besides, the clustering coefficient is also a measurement of the redundancy of the communication links around a vertex.

- **Weighted clustering coefficient** (Latora and Marchiori, 2003). The clustering coefficient does not consider the weight of edges. We may refine it by introducing the weighted clustering coefficient, , of a vertex, which is an attempt to generalize the concept of clustering coefficient to weighted networks. Given a vertex *v* in a weighted graph *G* it can be defined as:

$$c_w(v) = \sum_{i \neq j \in N_G(v)} w_{ij} \frac{1}{k_v(k_v - 1)} \qquad (2)$$

where is the neighborhood of *v* in *G* (the subgraph of all vertices connected to *v*), is the degree of relationship of the link between neighbor *i* and neighbor *j* (if there are no links), and is the number of neighbors. The weighted clustering coefficient can be interpreted as a measurement of the local efficiency of the network around a particular vertex, because vertices promoting strong interactions among their neighbors will have high values for this parameter. It can also be seen as a measurement of the redundancy of interactions around a vertex.

- **Distance centrality** (Sabidussi, 1996): The distance centrality of a vertex, , 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. Given a vertex *v* and a graph *G*, it can be defined as:

$$D_c(v) = \frac{1}{\sum_{t \in G} d_G(v,t)} \qquad (3)$$

where $d_G(v,t)$ is the minimum distance from vertex *v* to vertex *t* (i.e. the sum of the costs of relationship of all edges in the shortest path from *v* to *t*). 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. So, the term $\sum_{t \in G} d_G(v.t)$ will increase, meaning that it does not occupy a central position in the network. In that case, the distance centrality will be low.

Research has shown that employees who are central in networks learn faster, perform better and are more committed to the organization. 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 organization.

- **Betweenness centrality** (Freeman, 1977, Anthonisse, 1971): The betweenness centrality of a vertex, , is a measurement of the number of shortest paths traversing that particular vertex. Given a vertex *v* and a graph *G*, it can be defined as:

$$B_c(v) = \sum_{s \neq v \neq t \in G} \frac{\sigma_{st}(v)}{\sigma_{st}} \qquad (4)$$

  where is the number of shortest paths from *s* to *t* going through *v*, and is the total number of shortest paths between *s* and *t*. 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 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. So, the term $\sigma_{st}(v)/\sigma_{st}$ takes usually only two values: 1, if the shortest path between *s* and *t* goes through *v*, or 0 otherwise. So, 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 organization 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.

High values of the clustering coefficient are usually a symptom of *small world* behavior. The small world behavior of a network can be analyzed by comparing it with an equivalent (in number of vertices and edges) random network. When a network has a diameter (or average distance among vertices) similar to its random counterpart but, at the same time, has a higher average clustering coefficient, it is defined as a small world. It is well known (Watts, 2003) that small world networks are those optimizing the short and long term information flow efficiency. Those networks are also especially well adapted to solve the problem of searching knowledge through their vertices.

Table 1 summarizes the various SNA parameters that have been presented in this section, their meanings and the information they provide. These parameters, and their distributions and correlations will characterize the corresponding networks. From their study, a lot can be learned about the underlying organization and structure that those networks capture. An attempt to illustrate this is found in the following sections by studying several cases on real libre software projects.

Parameter	Meaning	Interpretation
Degree of relationship	Common activity among two entities (measured in commits)	How strong the relationship is
Cost of relationship	Inverse of the degree of relationship	Gives the cost of reaching one vertex from the other
Degree	Number of vertices connected to a node	Popularity of a vertex
Distribution degree	Probability of a vertex having a given degree	Topology of the network (Poisson or power law distributions)
Weighted degree	Degree considering weights of the links among vertices	Maximum capacity to receive information for a vertex. Effort in maintaining the relationships
Clustering coefficient	Fraction of the total number of edges that could exist for a given vertex that really exist	Transitivity of a network: tendency of a vertex to promote relationships among its neighbors. Helps identifying hot spots of knowledge interchange in dynamic networks
Weighted clustering coefficient	Generalization of the clustering coefficient concept to weighted networks	Local efficiency of the network around a vertex. Redundancy of interactions around a vertex
Distance centrality	Measurement of the proximity of a vertex to the rest	Gives the influence of a vertex in a graph. The higher the value the easier it is for the vertex to spread information through the network
Betweenness centrality	Number of shortest paths traversing a vertex	Measurement of the information control. Higher values mean that the vertex is an intermediate node for the communication of the rest. Vertices with high values are known to cover “structural holes”
Small world	Diameter (or avg distance among vertices) similar but higher avg clustering coefficient than random network	Optimizes short and long term information flow efficiency. Especially well adapted to solve the problem of searching knowledge through their vertices

Table 1: Summary of the SNA parameters described in this paper, their meaning and their interpretation.

Case studies: Apache, KDE and GNOME

Apache[5] , KDE, and GNOME are all well known libre software projects, large in size (each one well above the million of lines of code), in which several subprojects (modules) can be identified. They have already been studied from several points of view (Mockus et al., 2002, Koch and Schneider, 2002, Germán, 2004). Here, they will be used to show some of the features of our proposed methodology for applying SNA to software projects.

The use of versioning systems is fortunately the case for most large libre software projects. Some approaches on how to gather information from versioning repositories, in particular CVS (Zimmermann and Weißgerber, 2004, German, 2004, German and Hindle, 2005, Zimmermann et al., 2005), have been presented, and are used in this study. Therefore, focus is set on what to do once that information is available, and not on how to gather it.

[5]Throughout this article, references to Apache cover all projects lead by the Apache Foundation and not just the HTTPd server, usually known as the Apache web server

Project name	Modules (Vertices)	Edges	Average	% of edges (avg)
Apache	175	2491	14.23	8.13
KDE	73	1560	21.37	29.27
GNOME	667	121,134	181.61	27.23

Table 2: Number of vertices and edges of the module networks in the Apache, GNOME and KDE projects

Project name	Commiters (Vertices)	Edges	Commiters per module	Avg Number of edges
Apache	751	23,324	4.3	31.06
KDE	915	205,877	12.5	225.00
GNOME	869	N/A	1.3	N/A

Table 3: Number of vertices and edges of the commiter networks in the Apache and KDE projects

Tables 2 and 3 summarize the main parameters of both. In the case of commiter networks the GNOME case has been omitted.

By comparing the data in both tables some interesting conclusions can already be drawn. It may be observed, for instance, that the average number of commiters per module is greater in KDE (12.5) than in Apache (4.3), meaning more people being involved in the average KDE subproject. It can also be highlighted that the average degree on the commiters networks is in general larger than in the modules ones. This is specially true for KDE, which raises from a value of 21.4 in the latter case to 225 in the former. In the case of Apache it only raises from 14.2 to 31.1. Therefore, we can conclude that in those cases, commiters are much more linked than modules. The percentage of modules linked gives an idea of the synergy (in form of sharing information and experience) in a network as many modules have commiters in common. It can be assumed that this happens because of the technical proximity between modules. Regarding our case studies, KDE and GNOME show percentages near 30%, while the average Apache module is only linked to 8% of the other modules in the versioning system. So, Apache is specially fragmented in several module families that have no commiters in common. KDE and GNOME have a higher cohesion, while there is more dispersion in Apache.

In the following sections some specific aspects of all those networks will be studied, with the idea of illustrating both how the methodology is applied and which kind of results can be obtained from it.

Degree in the modules network

Table 4 shows that the number of modules for Apache (175), KDE (73) and GNOME (667) differ significantly. These projects are similar in software size (at least in order of magnitude), so the number of modules depends mainly on the various strategies that the projects follow when creating a new module. KDE has a structured CVS; applications that belong together are usually grouped into one module (so, for instance, there exists the *kdenetwork* module for many network applications or the *koffice* module for the various office suite programs). Apache has modules at the application level. Finally, GNOME follows a more *chaotic* approach, resulting in many more modules. Almost every application, even components (there are almost a dozen different GIMP add-ons with their own module) can be found to be a module in themselves.

The most popular characterization of network degree is the distribution degree *P*(*k*), which measures the probability of a given vertex having exactly *k* edges. However, the representation of *P*(*k*) in networks of a small size like ours is usually messy[6]. In these cases, the specialized literature prefers to use an associated parameter called the *cumulative distribution degree*, *CP*(*k*), which is defined as $CP(k)=\sum_{k}^{\infty}P(i)$ and is usually represented in a log-log scale.

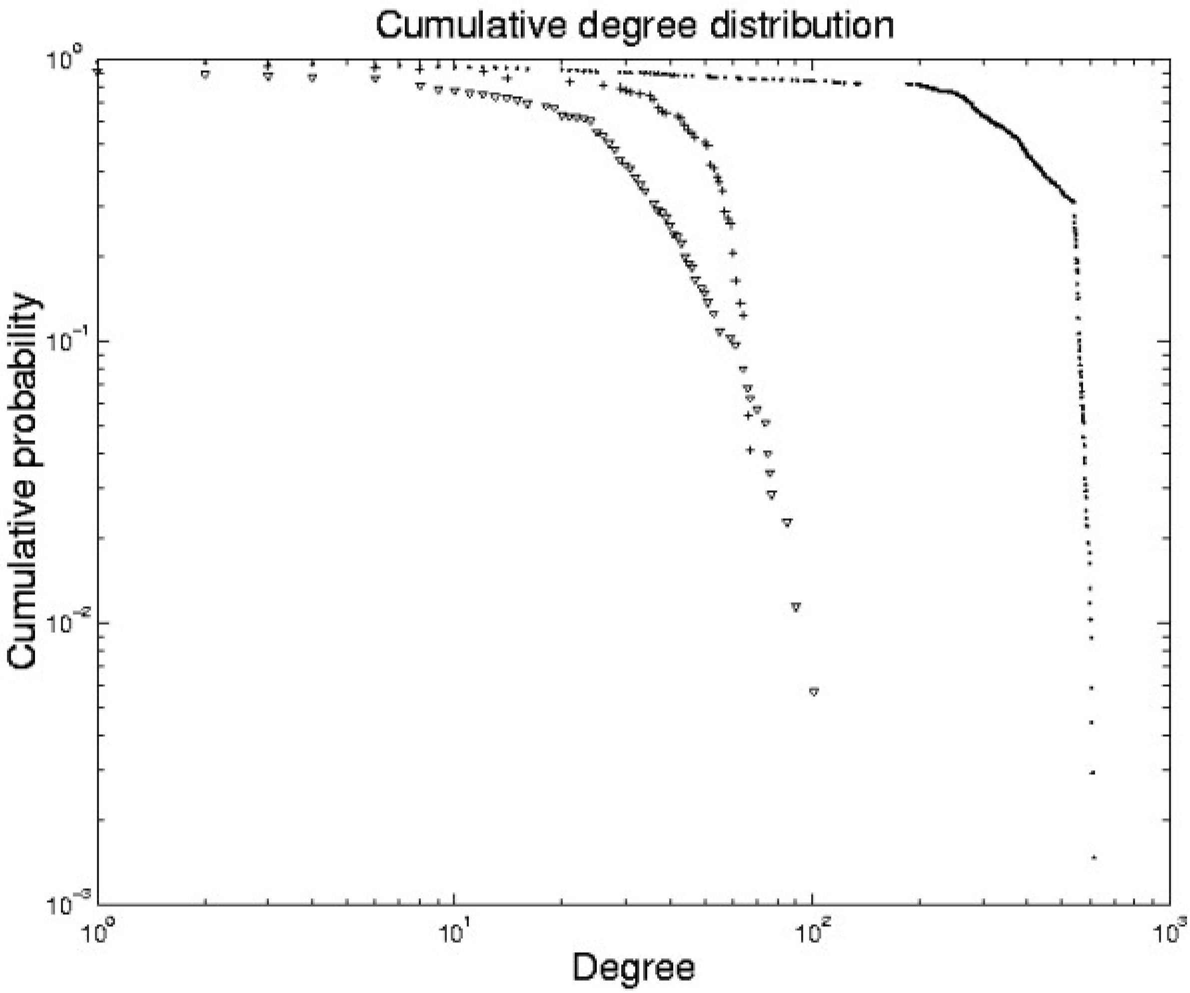
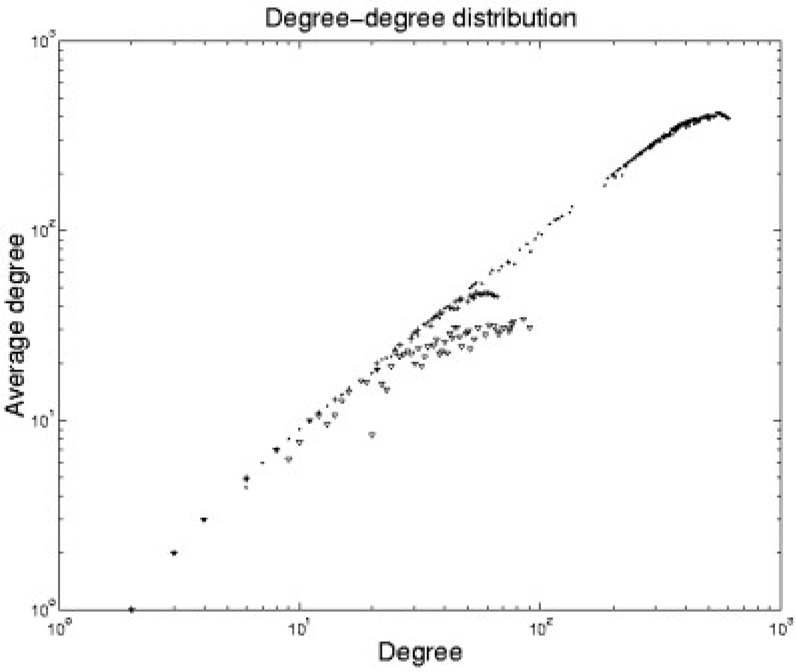


Figure 1: Cumulative degree distribution for Apache (▽), KDE (+) and GNOME (·)

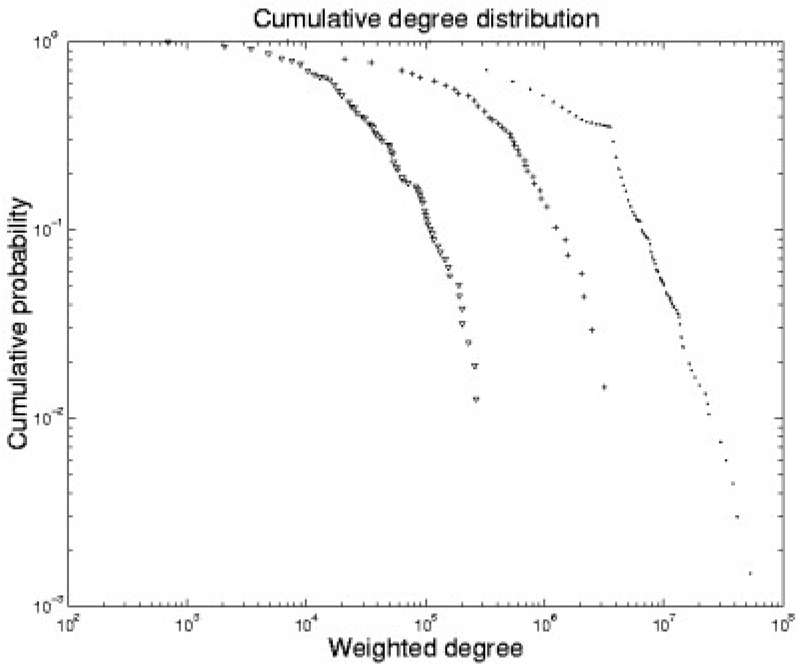
Fig. 1 shows the cumulative distribution degree for our three networks. As it can be observed, all of them present a sharp cutoff, which is a symptom of an exponential fall of the distribution degree tail. From a practical point of view, this means that none of our networks follow a power law distribution. This is quite a remarkable finding, because the specialized literature has shown that most social networks present power laws for this parameter. This implies that the growth of the network does not follow the traditional random preferential attachment law. Thus, it is difficult to come to any conclusions at this point; maybe by using a weighted network approach, as as shown later, we could infer more information about the network topology.

[6]Social network analysis has been applied to networks with hundreds of thousands, sometimes millions of vertices. In this sense, our network is of a small size even if we are handling large libre software projects.

Starting with the degree of the vertices, an analysis of the assortativity of the networks can also be carried out. The assortativity measures the average degree of neighbors of vertices having a particular degree. For this reason it can also be called the *degree - degree distribution*.



(a) Assortativity



(b) Cumulative weighted degree

Figure 2: Assortativity (degree - degree distribution) for Apache (), KDE (+) and GNOME (·). Cumulative weighted degree distribution for Apache (▽), KDE (+) and GNOME (·)

Fig. 2 represents this parameter for our networks. As it can be observed, all three networks are elitist, in the sense that vertices tend to connect to other vertices having a similar degree ('rich' with 'rich' and 'poor' with 'poor'). This is specially clear in the case of GNOME, where the curve approaches a linear equation with slope 1. Apache project deviates slightly from that behavior, showing some higher degree modules connected to other modules of a lower degree.

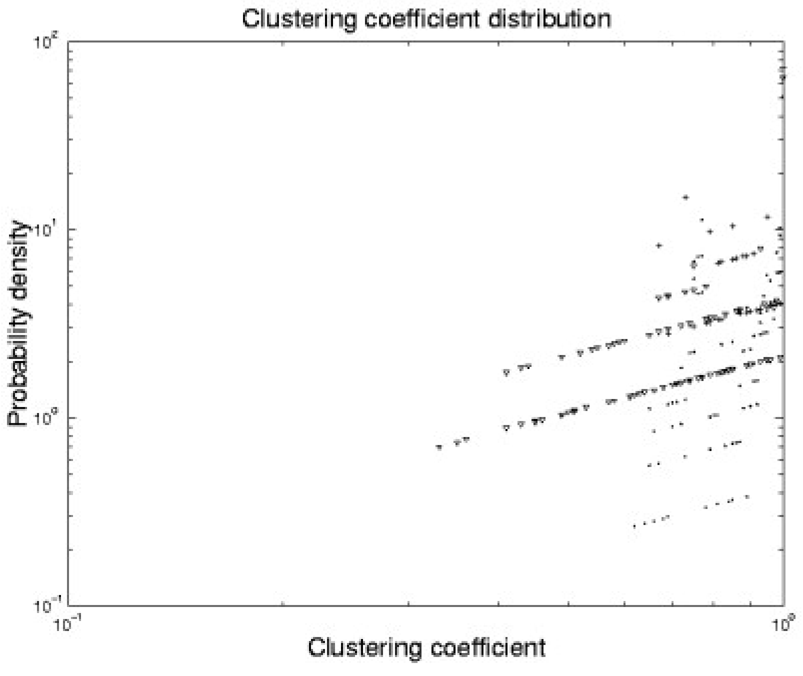
The previous analysis assumes unweighted networks. If weighted edges are considered now, similar conclusions are obtained. Fig. Error: Reference source not found represents the cumulative weighted distribution degree of the networks. Comparing this picture with fig. 1 it may be remarked that the sharp exponential cut-offs have disappeared. This is specially clear in the case of GNOME, where the curve tail can be clearly approximated by a power law. The interpretation for this finding is that the growth of that network could be driven by a preferential attachment law based on weighted degrees. This means that the probability of a new module to establish a link with a given vertex is proportional to the weighted degree of that vertex. That is, the commiters of new modules are, with high probability, commiters of modules which are well connected (have high weighted degree) in the network. It should be noted that the use of weights has given a more realistic picture.

From fig. 1 it can be remarked that the sharp cutoffs for Apache and KDE are close to each other. This means that the maximum number of relationships in both projects are similar. Nevertheless, observing fig. Error: Reference source not found, it can be seen that the KDE tail is clearly over the Apache tail. This fact implies that KDE weighted links are, on average, stronger than those of Apache. This can be quantitatively verified: we have calculated the average edge weight for the three projects obtaining 1,409.27 for Apache, 11,136.82 for KDE and 7,661.18 for GNOME.

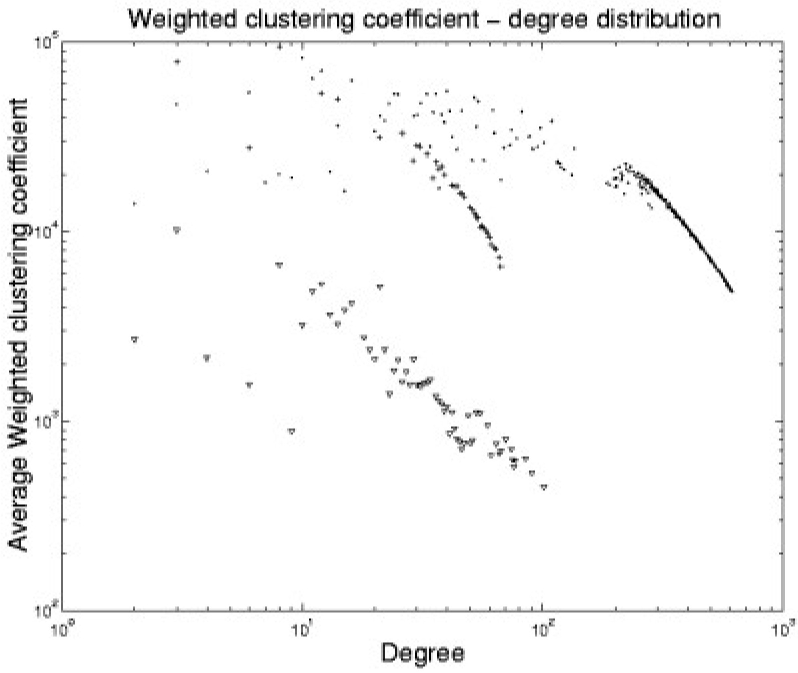
If multiplied, the average edge weight and the number of modules, the figures obtained are the total amount of commits performed by developers that contribute to at least two modules: 105,695 for Apache, 812,988 for KDE and 5,110,007 for GNOME. This gives an idea of the modularity of the modules as a lower number of commits is indicative for developer work being more focused on a low number of modules. While the figures for Apache are not surprising (we have already noticed with previous parameters that there exists/is a high level of structure in the Apache project), the difference between KDE and GNOME is astonishing. The organization of the KDE CVS repository yields in more independent modules than the ones found in the one for GNOME.

Clustering coefficient in the modules network

For the analysis of the clustering coefficient, we have represented its distribution in fig. 3.



(a) Clustering coefficient distribution



(b) Average weighted clustering coefficient

Figure 3: Clustering coefficient distribution for Apache (▽), KDE (+) and GNOME (·). Average weighted clustering coefficient as a function of the degree of vertices for Apache (▽), KDE (+) and GNOME (·)

In table. 4 the average distance <*d*> among vertices are represented, together with the average clustering coefficients <*cc*> for our three networks and their equivalent random counterparts (<*rd*> is the random average distance and <*rcc*> is the random average clustering coefficient). As it can be observed, the three networks satisfy the small world condition, since their average distances are slightly above to those of their random counterparts; but the clustering coefficients are clearly higher.

Project name	<*d*> / <*rd*>	<*cc*> / <*rcc*>
Apache	2.06 / 1.47	0.73 / 0.19
KDE	1.31 / 1.11	0.88 / 0.65
GNOME	1.46 / 1.10	0.87 / 0.54

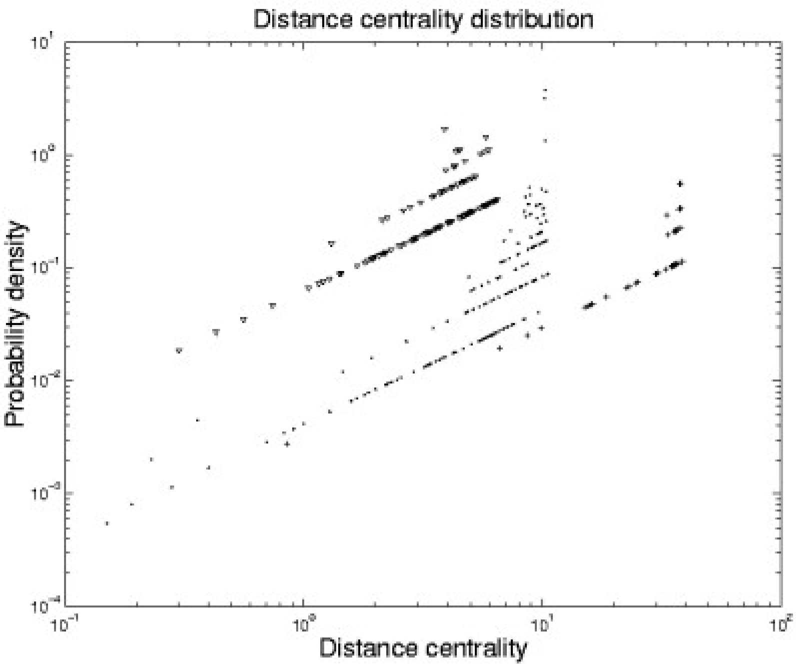
Table 4: Small world analysis for the module networks

As it can be observed, the average random clustering coefficients for KDE and GNOME are very close to the real ones, due to the high density of those networks. This could be an indication of over-redundancy in their links. That would mean that the same efficiency of information could be obtained with less relationships (i.e. eliminating many edges in the network without significantly increasing the diameter or reducing the clustering coefficient). In this sense, the Apache network seems to be more *optimized*. To interpret this fact, the reader may remember that links in this network are related to the existence of common developers for the linked modules. It should be noted that redundancy is probably a good characteristic of a libre software project as it may lose many developers without being affected heavily. It may be especially interesting to have over-redundancy in projects with many volunteers, as in those environments turn-over may be high. Future research should focus on investigating whether over-redundancy is a good or bad parameter in the case of libre software projects. On the other side, how much of this redundancy is due to the taking of a static picture of the project should be researched; it may well be that the redundancy we have observed is the result of different generations of developers working on the same file in different periods of time.

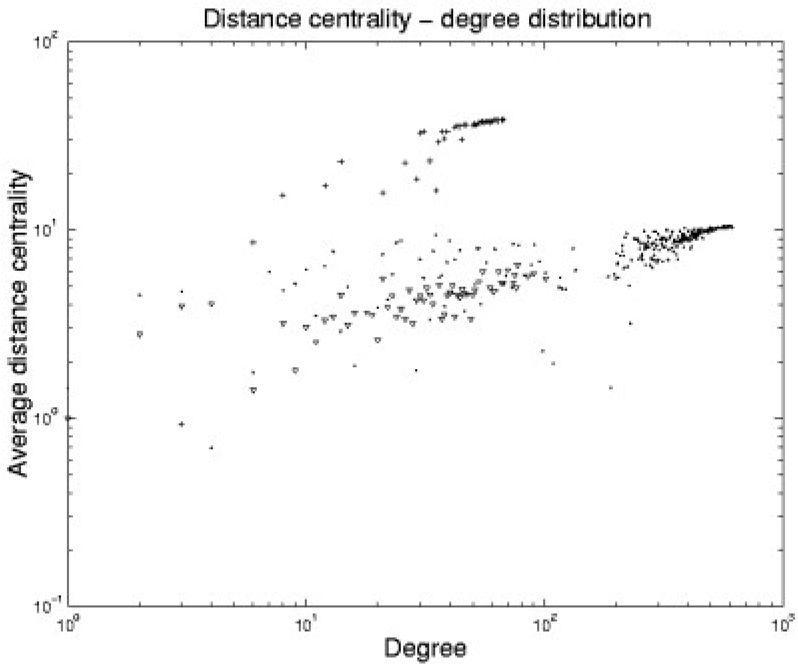
Some interesting conclusions can also be obtained by looking at the weighted clustering coefficient. In fig. Error: Reference source not found we can observe the average weighted clustering coefficient as a function of the degree of vertices (the weighted degree - degree distribution). As we have already noticed, the KDE and GNOME networks have a similar local redundancy, which is higher than the one of Apache. High redundancy implies more fluid information exchanges in the short distances for the first two projects. Besides, the weighted clustering coefficient lowers with the degree in all cases, according to a power law function. We can infer that highly connected vertices cannot maintain their neighbors as closely related as poorly connected ones. This happens typically in most social networks because the cost of maintaining close relationships in small groups is much lower than the equivalent cost for large neighborhoods.

Distance centrality in the modules network

The analysis of the distance centrality of vertices is relevant because this parameter measures how close a vertex is to the rest of the network. In fig. 4 the distance centrality distribution for our three networks can be observed. They follow multiple power laws, making higher values of the parameter most probable. This is an indication of well structured networks for the fast spread of knowledge and information.



(a) Distance centrality distribution



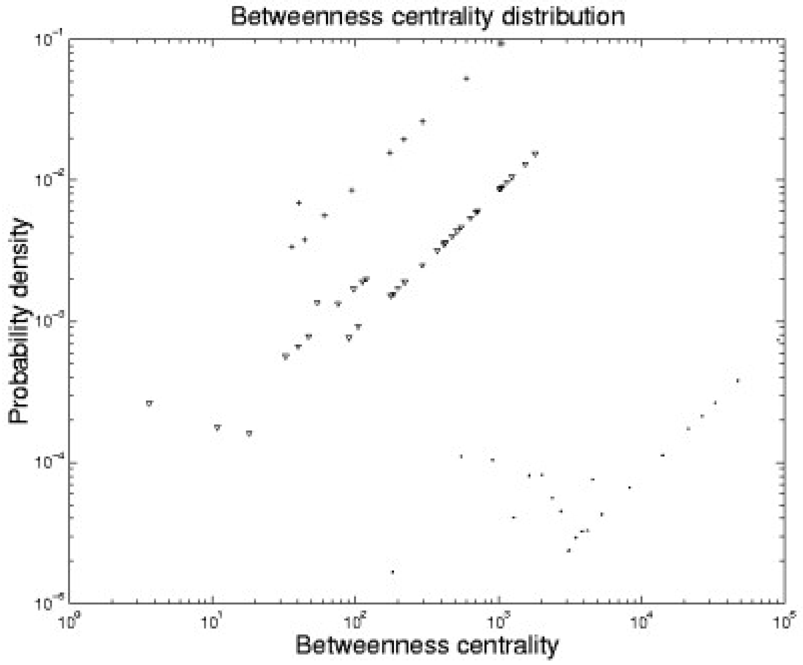
(b) Average distance centrality

Figure 4: Distance centrality distribution for Apache (▽), KDE (+) and GNOME (·). Average distance centrality as a function of the degree of vertices for Apache (▽), KDE (+) and GNOME (·)

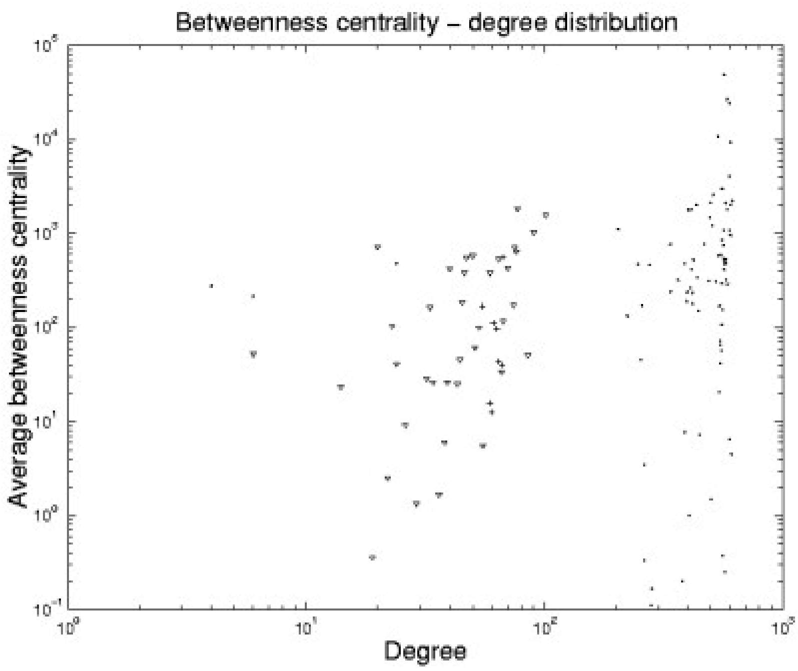
We can also analyze the average distance centrality as a function of the degree (average distance centrality - degree distribution), which is shown if Fig. Error: Reference source not found. It can be observed that in all three cases the average distance centrality grows with the degree following, approximately, a power law of low exponent. This means that, in terms of distance centrality, the networks are quite *democratic*, because there is not a clear advantage of well connected nodes compared to the rest. Curiously enough, the Apache and GNOME curves are quite similar, while the KDE one is clearly an order of magnitude over the rest. This could be an effect of the lower size of this network, but is also an indication of a specially well structured network in terms of information spread. So, even if KDE showed to be more modular as has been seen for a previous parameter, its structure seems to maximize information flow.

Betweenness centrality in the modules network

The distance centrality of a vertex indicates how well new knowledge created in a vertex spreads to the rest of the network. On the other hand, betweenness centrality is a measurement of how easy it is for a vertex to generate this new information. Vertices with high betweenness centrality indexes are the crossroads of organizations, where information from different origins can be intercepted, analyzed or manipulated. In fig. 5 the betweenness centrality distribution for our three networks can be observed. In the same way, this was the case for distance centrality, as it grows following a multiple power law. Nevertheless, there is a significant difference between the distribution of these two parameters. Although the log-log scale of the axis of fig. 5 does not allow to visualize it, the most probable value of the betweenness centrality in all three networks is zero. Just to show an example, only 102 out of 677 vertices of the GNOME network have a nonzero betweenness centrality. So, the distance centrality is a common good of all members of the network, while the betweenness centrality is owned by a reduced elite. This should not be surprising at all, as projects usually have modules (i.e. applications) which have a more central position and attract more development attention. Surrounding these modules other, minor modules may appear.



(a) Betweenness centrality distribution



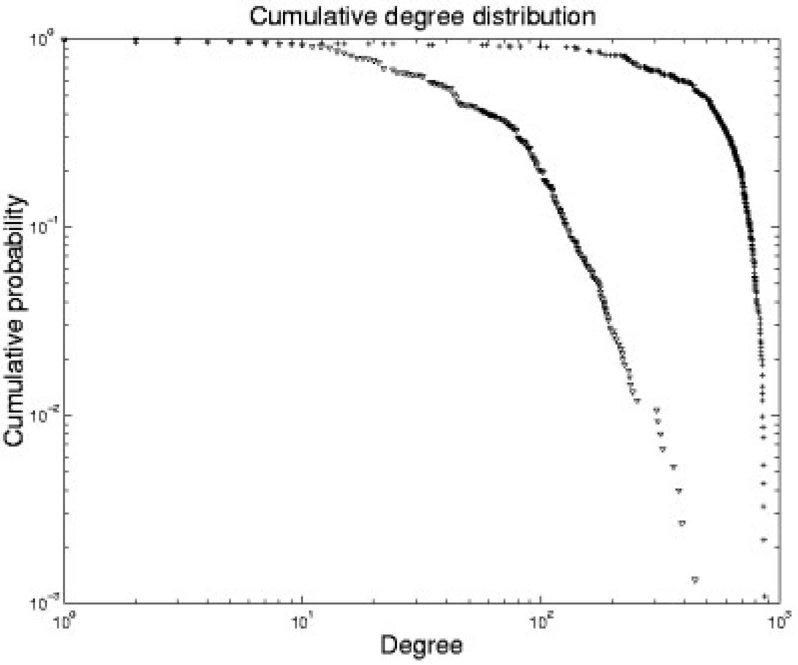
(b) Average betweenness centrality distribution

Figure 5: Betweenness centrality distribution for Apache (▽), KDE (+) and GNOME (·). Average betweenness centrality distribution for Apache (▽), KDE (+) and GNOME (·)

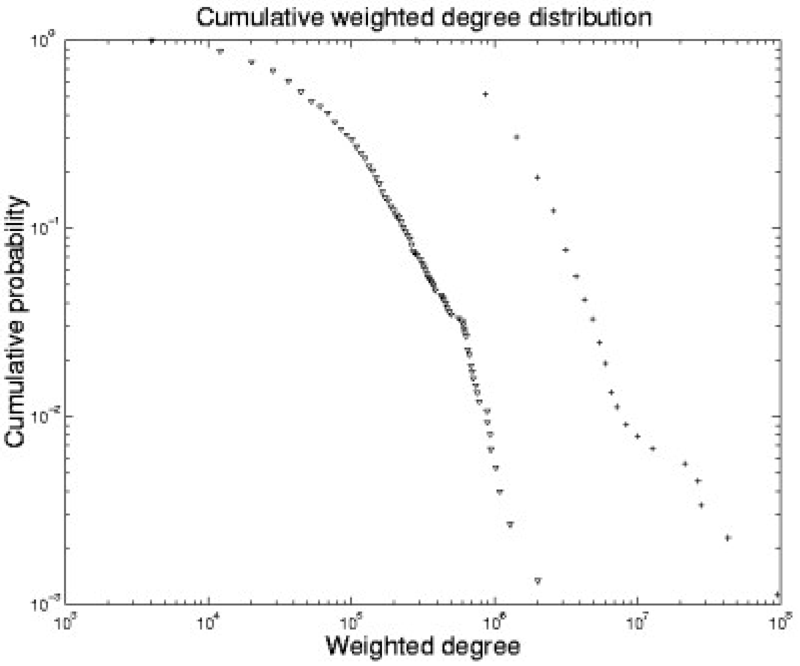
This fact can also be visualized in fig. Error: Reference source not found, where we represent the average betweenness centrality as a function of the degree. It can be clearly seen that only vertices of high degree have nonzero betweenness centralities.

Commiter networks

The analysis of commiter networks draw similar conclusions to those shown for module networks, and therefore they are not going to be commented in detail. For instance, the cumulative degree distribution for the two commiter networks is shown in fig. 6, which has clearly the same qualitative properties than for this parameter for the module networks shown in fig. 1. The same holds true for the commiter cumulative weighted degree distribution depicted in fig. Error: Reference source not found, or for the average degree as a function of degree, depicted in fig. 7, where it can be noticed how both networks maintain the elitist characteristic also observed in the case of modules.



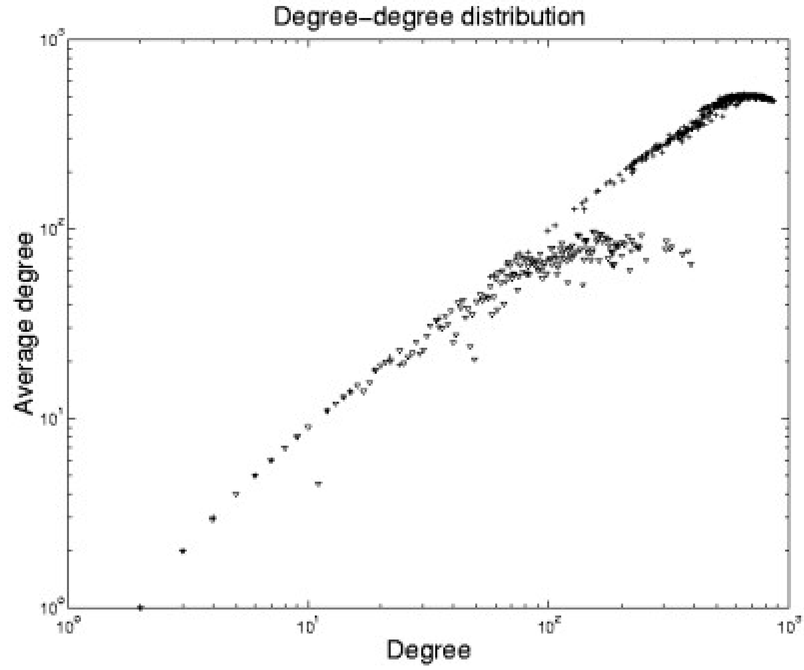
(a) Cumulative degree distribution



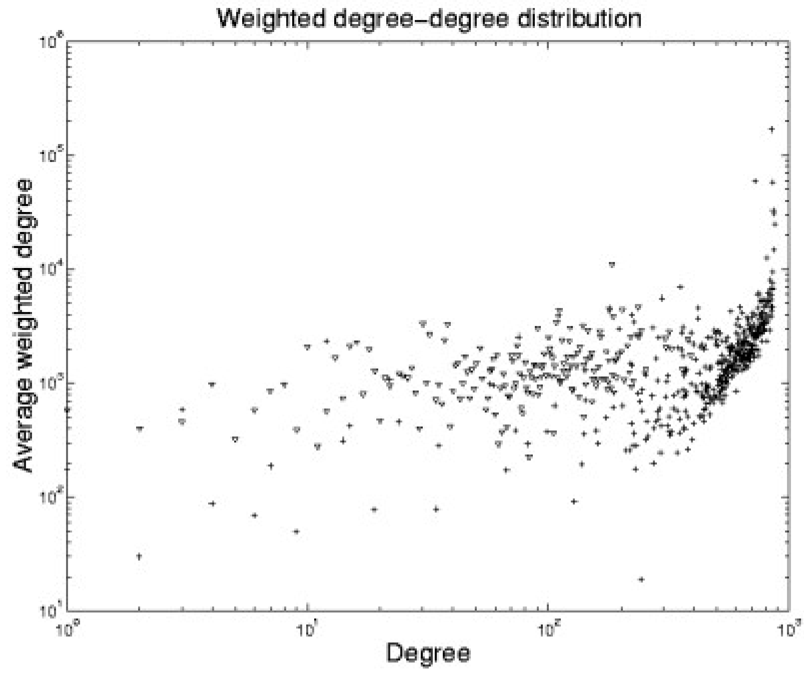
(b) Cumulative weighted degree distribution

Figure 6: Cumulative degree distribution for Apache (▽) and KDE (+). Cumulative weighted degree distribution for Apache (▽) and KDE (+)

An interesting feature of commiter networks can be seen in fig. Error: Reference source not found. The average weighted degree of authors remains more or less constant for low values of the degree. Nevertheless, in the case of KDE, it increases meaningfully for the highest degrees. The implication is that commiters with higher degrees not only have more relationships than the rest, but also their relationships are much stronger than the average. This indicates that authors having higher degrees are more involved in the project development and establish stronger links than the rest. At the same time, as we observed in fig. 7, they only relate to other commiters that are involved in the project to the same degree as they are. If this behavior is found in other large libre software projects, it could be a valid method to identify the leading "core" group of a libre software project. On the other hand, the Apache project seems to promote a single category of developers, given that the weighted degree does not depend so clearly on the degree of vertices. It may be also that because of the fragmentation of the Apache project in many *families* of modules, it is easy to developers to reach a point where they do not have the possibility to get to know more developers.



(a) Degree - degree distribution



(b) Average weighted degree as a function of degree

Figure 7: Degree - degree distribution for Apache (▽) and KDE (+). Average weighted degree as a function of the degree for Apache (▽) and KDE (+)

Table 5 digs into the small-world properties of commiter networks. As we can observe, both networks can still considered to be small world. The Apache case is specially interesting, because an increase in the average distance is observed. This characteristic plus the large value of the clustering coefficient may indicate that the network is forming cliques.

Project name	*<d> / <rd>*	*<cc> / <rcc>*
Apache	2.18 / 1.60	0.84 / 0.08
KDE	1.47 / 1.10	0.86 / 0.52

Table 5: Small world analysis for commiter networks

Conclusions, lessons learned and further work

In this paper an approach to the study of libre (free, open source) software projects has been presented, based on the quantitative and qualitative application of social networks analysis to the data retrieved from source code management repositories. Since most libre software projects maintain such repositories, and allow for public read-only access to them, this analysis can be repeated for many of them. However, given its characteristics, it will be most useful for large projects, well above the hundreds of thousands of lines of code and dozens of developers.

We have designed a detailed methodology which applies this SNA-based approach to the study of CVS data, and which can be automated. It starts by downloading the required information from CVS, and produces as an output several graphs and tables which can be interpreted to gain knowledge about the informal organization of the studied project. It is important to highlight a set of parameters in the output that are suitable for characterizing several aspects of the organization of the studied project, which makes it possible to gain a lot of insight on how a group of developers is managing coordination and information flow within the project. In addition, it has been shown that the introduction of weights in the relationships gives more realistic information about the projects under study.

It has also been shown how our methodology is applied to some important and well known projects: KDE, Apache and GNOME. Although these studies are sketched just as case examples, some relevant results can also be extracted from them. For instance, it has been shown how all the networks that have been studied fulfill the requirements to be a small world. This has important consequences on their characterization, since small worlds have been comprehensively studied and are well understood in many respects. We have also not only found that the growth of the studied networks cannot be explained by random preference attachment (something that could be previously suspected). Moreover, it matches pretty well the pattern of preference attachment related to the weight (amount of shared effort) of links. Some other relevant results are the elitist behavior found in these projects with respect to the connectivity of modules and developers, which are indicators of an over-redundancy of links, and of a good structure for the flow of knowledge, and the absence of centers of power (in terms of information flux). All of this conclusions should be validated by studying more projects, and by analyzing with detail their micro implications before being raised to the category of characteristics of libre software projects, but that so far are good lines of further research.

There are some other studies applying similar techniques to libre software projects. For instance, (Crowston and Howison, 2003), which suggests that large projects are more modular than small ones. However, to our knowledge the kind of comprehensive analysis shown in this paper has never been proposed as a methodology for characterizing libre software projects and their coordination structure. In fact, after using it in the study of some projects, we believe that it has a great potential to explore informal organizational patterns, and uncovering non-obvious relationships and characteristics of their underlying structure of coordination.

References

Albert, R. and Barabasi, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97.

Albert, R., Barabási, A.-L., Jeong, H., and Bianconi, G. (2000). Power-law distribution of the world wide web. *Science*, 287.

Anthonisse, J. (1971). The rush in a directed graph. Technical report, Stichting Mathemastisch Centrum, Amsterdam, The Netherlands.

Conway, M. (1968). How do committees invent? *Datamation*, 14(4):28–31.

Crowston, K. and Howison, J. (2003). The social structure of open source software development teams. In *Proceedings of the ICIS*.

Crowston, K., Scozzi, B., and Buonocore, S. (2003). An explorative study of open source software development structure. In *Proceedings of the ECIS*, Naples, Italy.

Dinh-Trong, T. T. and Bieman, J. M. (2005). The FreeBSD project: A replication case study of Open Source development. *IEEE Transactions on Software Engineering*, 31(6):481–494.

Elliott, M. and Scacchi, W. (2004). Mobilization of software developers: The free software movement. Technical report. `http://opensource.mit.edu/papers/elliottscacchi2.pdf`.

Freeman, C. (1977). A set of measures of centrality based on betweenness. *Sociometry 40, 35-41*.

Fuggetta, A. (2003). Open source software - an evaluation. *Journal of Systems and Software*, 66(1):77–90.

German, D. (2004). An empirical study of fine-grained software modifications. In *International Conference in Software Maintenance*.

German, D. M. and Hindle, A. (2005). Visualizing the evolution of software using softChange. *Journal of Software Engineering and Knowledge Engineering*. Accepted for publication, under revisions.

Germán, D. (2004). The GNOME project: a case study of open source, global software development. *Journal of Software Process: Improvement and Practice*.

Ghosh, R. A. and Prakash, V. V. (2000). The orbiten free software survey. 5(7).

Gîrba, T., Kuhn, A., Seeberger, M., and Ducasse, S. (2005). How developers drive software evolution. In *Proceedings of the International Workshop on Principles in Software Evolution*, pages 113–122, Lisbon, Portugal.

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

González-Barahona, J. M., López-Fernández, L., and Robles, G. (2004). Community structure of modules in the Apache project.

Guimera, R., Danon, L., Diaz-Guilera, A., Giralt, F., and Arenas, A. (2003). Self-similar community structure in a network of human interactions. *Physical Review E 68, 065103(R)*.

Healy, K. and Schussman, A. (2003). The ecology of open-source software development. Technical report, University of Arizona, USA.

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

Krishnamurthy, S. (2002). Cave or community? an empirical investigation of 100 mature Open Source projects. *First Monday*, 7(6).

Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (2002). The web and social networks. *IEEE Computer*, 35(11):32–36.

Latora, V. and Marchiori, M. (2003). Economic small-world behavior in weighted networks. *Euro Physics Journal B 32, 249-263*.

Lopez, L., Gonzalez-Barahona, J. M., and Robles, G. (2004). Applying social network analysis to the information in cvs repositories. In *Proceedings of the International Workshop on Mining Software Repositories, 26th International Conference on Software Engineering*, Edinburg, Scotland, UK.

Madey, G., Freeh, V., and Tynan, R. (2002). The open source development phenomenon: An analysis based on social network theory. In *Americas Conference on Information Systems (AMCIS2002)*, pages 1806–1813, Dallas, TX, USA.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346.

Newman, M. E. J. (2001a). Scientific collaboration networks: I. network construction and fundamental results. *Phys. Rev. E 64, 016131*.

Newman, M. E. J. (2001b). Scientific collaboration networks: Ii. shortest paths, weighted networks, and centrality. *Phys. Rev. E 64, 016132*.

Raymond, E. S. (1997). The cathedral and the bazar. *First Monday*, 3(3).

Robles, G., Amor, J. J., Gonzalez-Barahona, J. M., and Herraiz, I. (2005a). Evolution and growth in large libre software projects. In *Proceedings of the International Workshop on Principles in Software Evolution*, pages 165–174, Lisbon, Portugal.

Robles, G., González-Barahona, J. M., and Michlmayr, M. (2005b). 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.

Robles, G., Koch, S., and Gonzalez-Barahona, J. M. (2004). 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), 26th International Conference on Software Engineering*, Edinburg, Scotland, UK.

Sabidussi, G. (1996). The centrality index of a graph. *Psychometrika 31, 581-606*.

Wagstrom, P. A., Herbsleb, J. D., and Carley, K. (2005). A social network approach to Free/Open Source Software simulation. In *Proceedings of the 1st International Conference on Open Source Systems*, pages 100–107, Genoa, Italy.

Watts, D. J. (2003). *Six Degrees*. W. W. Norton & Company, New York.

Watts, D. J. and Strogatz, S. (1998). Collective dynamics of small-world networks. *Nature 393, 440-442*.

Xu, J., Gao, Y., Christley, S., and Madey, G. (2005). A topological analysis of the Open Source Software development community. In *Proceedings of the38th Hawaii International Conference on System Sciences*, Hawaii, USA.

Zimmermann, T. and Weißgerber, P. (2004). Processing CVS data for fine-grained analysis. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 2–6, Edinburg, Scotland, UK.

Zimmermann, T., Weißgerber, P., Diehl, S., and Zeller, A. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445.