Applying Social Network Analysis to the Information in CVS Repositories

Luis Lopez-Fernandez, Gregorio Robles, Jesus M. Gonzalez-Barahona
GSyC, Universidad Rey Juan Carlos
{llopez,grex,jgb}@gsyc.escet.urjc.es

Abstract

The huge quantities of data available in the CVS repositories of large, long-lived libre (free, open source) software projects, and the many interrelationships among those data offer opportunities for extracting large amounts of valuable information about their structure, evolution and internal processes. Unfortunately, the sheer volume of that information renders it almost unusable without applying methodologies which highlight the relevant information for a given aspect of the project. In this paper, we propose the use of a well known set of methodologies (social network analysis) for characterizing libre software projects, their evolution over time and their internal structure. In addition, we show how we have applied such methodologies to real cases, and extract some preliminary conclusions from that experience.

Keywords: *source code repositories, visualization techniques, complex networks, libre software engineering*

1 Introduction

The study and characterization of complex systems is an active research area, with many interesting open problems. Special attention has been paid recently to techniques based on network analysis, thanks to their power to capture some important characteristics and relationships. Network characterization is widely used in many scientific and technological disciplines, ranging from neurobiology [14] to computer networks [1] [3] or linguistics [9] (to mention just some examples). In this paper we apply this kind of analysis to software projects, using as a base the data available in their source code versioning repository (usually CVS). Fortunately, most large (both in code size and number of developers) libre (free, open source) software projects maintain such repositories, and grant public access to them.

The information in the CVS repositories of libre software projects has been gathered and analyzed using several methodologies [12] [5], but still many other approaches are possible. Among them, we explore here how to apply some techniques already common in the traditional (social) network analysis. The proposed approach is based on considering either modules (usually CVS directories) or developers (commiters to the CVS) as vertices, and the number of common commits as the weight of the link between any two vertices (see section 3 for a more detailed definition). This way, we end up with a weighted graph which captures some relationships between developers or modules, in which characteristics as information flow or communities can be studied.

There have been some other works analyzing social networks in the libre software world. [7] hypothesizes that the organization of libre software projects can be modeled as self-organizing social networks and shows that this seems to be true at least when studying SourceForge projects. [6] proposes also a sort of network analysis for libre software projects, but considering source dependencies between modules. Our approach explores how to apply those network analysis techniques in a more comprehensive and complete way. To expose it, we will start by introducing some basic concepts of social network analysis which are used later (section 2), and the definition of the networks we consider 3. In section 4 we introduce the characterization we propose for those networks, and later, in section 5, we show some examples of the application of that characterization to Apache, GNOME and KDE. To finish, we offer some conclusions and discuss some future work.

2 Basic concepts on Social Network Analysis

The Theory of Complex Networks is based on representing complex systems as graphs. There are many examples in the literature where this approach has been successfully used in very different scientific and technological disciplines, identifying vertices and links as relevant for each specific domain. For example, in ecological networks each vertex may represent a particular specie, with a link between two species if one of them "eats" the other. When dealing with social networks, we may identify vertices with persons or groups of people, considering a link when there is some kind of relationship between them.

Among the different kinds of networks that can be considered, in this paper, we use affiliation networks. In affiliation networks there are two types of vertices: *actors* and *groups*. When we represent the network in terms of actors, each vertex is associated with a particular person and two vertices are linked together when they belong to the same group of people. When we represent the network in terms of groups, each vertex is associated with a group and two groups are linked through an edge when there is, at least, one person belonging to both at the same time.

Social networks can be directed (when the relationship between any two vertices is one way, like "is a boss of") or undirected (when it is bidirectional, like "live together"). In addition, they can be weighted (each edge has an associated numeric value) or unweighted (each edge exists or not).

3 Definition of the networks of developers and modules

In the approach we propose, for each project we build two networks using the commit information of the CVS system. Both correspond to the two sides of an affiliation network obtained when we consider commiters and modules in libre software projects. In both cases we consider weighted undirected networks as follows:

- **Commiter network**. Each vertex corresponds to a particular commiter (usually, a developer of the project). Two commiters are linked when they have contributed to at least one common module, being the weight of the corresponding edge the number of commits performed by both developers to all common modules.
- **Module network**. Vertices represent a software module of the project. Two modules are linked when there is at least one commiter who has contributed to both of them. Edges are weighted by the total number of commits performed by common commiters to both modules.

The definition of what is a module will be different from project to project, but usually will correspond to top level directories in the CVS repository. In the case of both networks, the weight of each edge (*degree of relationship*) reflects the *closeness* of two vertices. The higher it is, the stronger the relationship between the given two vertices. We may also define the *cost of relationship* between any two vertices as the inverse of the *degree of relationship*. That *cost of relationship* is a measure of the "distance" between them, in the sense that the higher this parameter the more difficult to reach one vertex from the other. For this reason we use the *cost of relationship* as the base for defining a distance in our networks. Given a pair of vertices i and j, we define the distance between them as $d_{ij} = \sum_{e \in P_{i,j}} c_e$, where P_i, j is the set of all the edges in the shortest path from i to j, and c_e is the *cost of relationship* of edge e of such path.

4 Characterization of the networks considered for each project

For our analysis, we have considered a number of parameters characterizing the topology of the networks. In particular, we use the following definitions (which are common in the analysis of affiliation networks):

- **Degree of a vertex** (k): number of edges connected to that vertex. In the case of commiter networks, for each commiter it represents the number of *companion* commiters, contributing to the same modules as the given one. In the case of module networks, it is the total number of modules with which the given one shares commiters.
- **Weighted degree of a vertex**: sum of the weights of all edges connected to that particular vertex. This can be interpreted as the degree of relationship of a given vertex with its direct neighborhood.
- **Distance centrality of a vertex** [13] (D_c): proximity to the rest of vertices in the network. It is also called *closeness centrality*: the higher its value, the closer that vertex is to the others (on average). Given a vertex v and a graph G, it can be defined as:

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

  where $d_G(v,t)$ is the minimum distance from vertex v to vertex t (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: the higher its value, the easiest it is for that vertex to spread information into that network. Let's 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. In that case the term $\sum_{t \in G} d_G(v,t)$ will be high, meaning that the vertex is not placed in a central position in the network, being its distance centrality low. This parameter can be used to identify modules or commiters which are *well related* in a project.
- **Betweenness centrality of a vertex** [4, 2]: The betweenness centrality of a vertex B_c 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:

  $$B_c(v) = \sum_{s \neq v \neq t / inG} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{2}$$

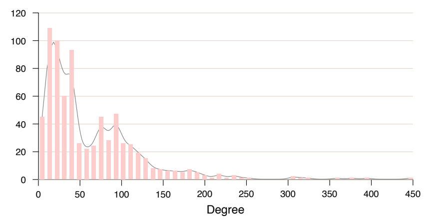

Figure 1. Distribution of the degrees of commiters in Apache, circa February 2004

where $\sigma_{st}(v)$ is the number of shortest paths from s to t going through v, and σ_{st} 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 importance of a vertex in a given graph, in the sense that vertices with a high value of this parameter are intermediate nodes for the communication of the rest. In the case of weighted networks, multiple shortest paths between any pair of vertices are highly improbable. So, the term $\frac{\sigma_{st}(v)}{\sigma_{st}}$ takes usually only two values: 1, if the shortest path between s and t goes through v, or 0 otherwise. Therefore, the betweenness centrality is just a measurement of the number of shortest paths traversing a given vertex.

- **Clustering coefficient of a vertex** [14]: The clustering coefficient c of a vertex measures the connectivity of its direct neighborhood. Given a vertex v in a graph G, it can be defined as the probability that any two neighbors of v be connected. Hence

$$c(v) = \frac{E(v)}{k_v(k_v - 1)}, \tag{3}$$

where k_v is the number of neighbors of v and $E(v)$ is the number of edges between those neighbors. A high clustering coefficient in a network indicates that this network has a tendency to form cliques. Observe that the clustering coefficient does not consider the weight of edges.

- **Weighted clustering coefficient of a vertex** [10]: The weighted clustering coefficient c_w of a vertex 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:

$$c_w(v) = \sum_{i \neq j \in N_G(v)} w_{ij} \frac{1}{k_v(k_v - 1)}, \tag{4}$$

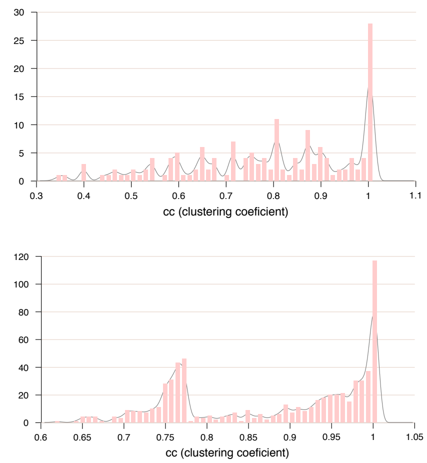

Figure 2. Clustering coefficient of modules in Apache (top) and GNOME (bottom), circa February 2004 (distribution)

where $N_G(v)$ is the neighborhood of v in G (the subgraph of all vertices connected to v), w_{ij} is the degree of relationship of the link between neighbor i and neighbor j ($w_{ij} = 0$ if there are no link), and k_v 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. For our networks, remark that the term $\sum_{i \neq j \in N_G(v)} w_{ij}$ can be seen as the total *degree of relationship* in the neighborhood of vertex v, while $\frac{1}{k_v(k_v-1)}$ is the total number of relationships that could exists in that neighborhood.

5 Case studies: Apache, GNOME and KDE modules

Apache, GNOME and KDE are all well known libre software projects, large in size (each well above the million lines of code), in which several subprojects (modules) can be identified. They have already been studied (for instance in [11] and [8]) from several points of view. We have used them to apply our methodology, and in this section some results of that application are shown (just an example of how a project can be characterized from several points of view).

In figure 1 the distribution of the degree of relationship for each commiter in the Apache project is shown as an example of how developers can be characterized by how they relate to each other. It is easy to appreciate how that distributions shows two peaks, one between 20-40 and other around 70-90. Only a handful of developers has direct relationship with more than 200 companions.

In figure 2 the distribution of the clustering coefficient of modules in Apache and GNOME is compared. Although in both cases there is a peak in 1 (meaning that in many cases the direct neighborhood of a module is completely linked together), there is an interesting peak in GNOME around 0.77, which should be studied but probably corresponds to a sparse-connected cluster.

Figure 3 shows how, despite differences in the distribution of the clustering coefficient, the distribution of the weighted clustering coefficient has more similar shapes, with a quick rise from zero to a maximum, and a slower, asymptotic decline later. This would mean than in the three projects most nodes (those near the peak) are in clusters with a similar interconnection structure.

As a final example, on the evolution of a project, figure 4 shows the distribution of the connection degree of four snapshots of the Apache project. It can be seen how there is a tremendous growth in the connection degree of the most connected module (from 34 in 2001 to more than 100 in 2004), while the shape of the distribution changes over time: from 2001 to 2002 a two-peak structure develops, which slowly changes into a one-peak distribution through 2003 and 2004.

For lack of space we do not offer it here, but the analysis of the top modules and developers for each parameter considered gives a lot of insight on which ones are helping to maintain the projects together, to deal with information flows, or are the aggregators of clusters.

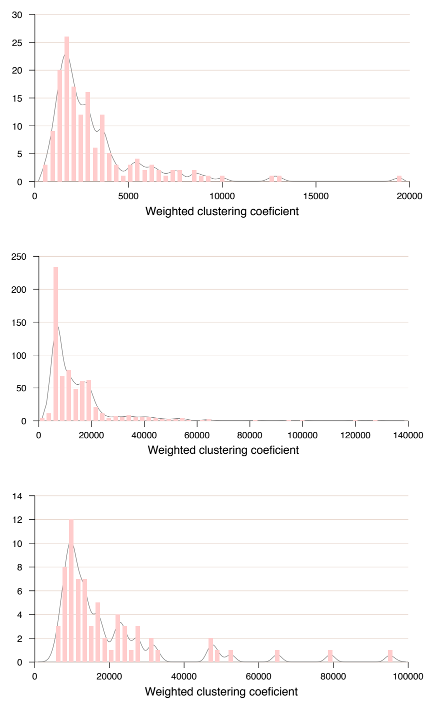

Figure 3. Weighted clustering coefficient of modules in Apache (top), GNOME (middle), and KDE (bottom), circa February 2004 (distribution)

6 Conclusions and further work

In this paper we have shown a methodology which applies affiliation network analysis to data gathered from CVS repositories. We also offer some examples of how it can be applied to characterize libre software projects. From a more general point of view, we have learned (demonstration not shown in this paper) that in the three analyzed cases (Apache, GNOME and KDE), both the commiters and the modules networks are small-world networks, which means that all the theory developed for them applies here.

Our group is still starting to explore the many paths open by this methodology. Currently, we are interested in analyzing a large number of projects, looking for correlations which can help us to make estimations and predictions of the future evolution of projects. We are also looking for characterizations of projects based on the parameters of the curves that interpolate the distributions of the parameters we are studying. And of course, applying other techniques usual in small-world and other social networks.

We feel that these research paths will allow for the more complete understanding of how libre software projects differentiate from each other, and also will help to identify common patterns and invariants.

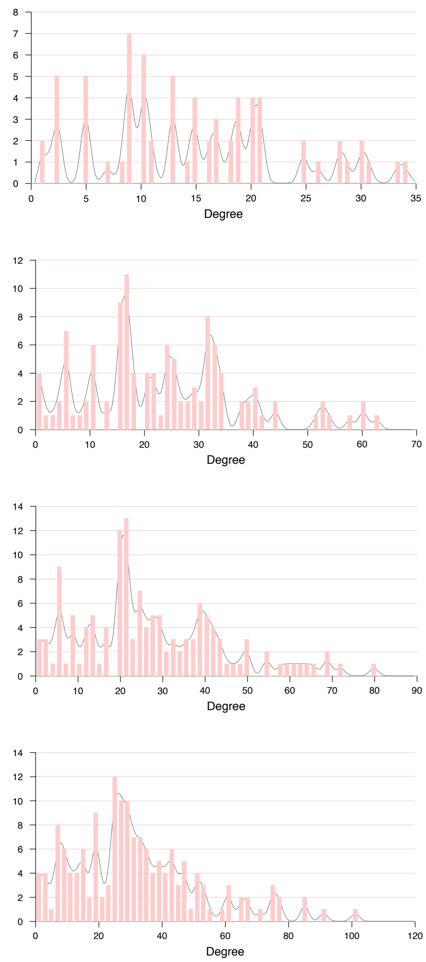

Figure 4. Connection degree of modules in Apache circa February from 2001 (top) to 2004 (bottom) (distribution)

References

[1] R. Albert, A. L. Barabsi, H. Jeong, and G. Bianconi. Power-law distribution of the world wide web. *Science*, 287, 2000.

[2] J. Anthonisse. The rush in a directed graph. Technical report, Stichting Mathemastisch Centrum, Amsterdam, The Netherlands, 1971.

[3] Cancho and R. Sole. The small world of human language. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 268:2261–2265, Nov. 2001.

[4] C. Freeman. A set of measures of centrality based on betweenness. *Sociometry 40, 35-41*, 1977.

[5] D. Germn and A. Mockus. Automating the measurement of open source projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering*, Portland, Oregon, 2003.

[6] R. A. Ghosh. Clustering and dependencies in free/open source software development: Methodology and tools. *First Monday*, 2003. `http://www.firstmonday.dk/issues/issue8_4/ghosh/index.html`.

[7] V. F. Greg Madey and R. Tynan. 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, 2002. `http://www.nd.edu/~oss/Papers/amcis_oss.pdf`.

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

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

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

[11] A. Mockus, R. Fielding, and J. Herbsleb. A case study of open source software development: The Apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 263–272, Limerick, Ireland, 2000.

[12] G. Robles-Martinez, J. M. Gonzalez-Barahona, J. Centeno-Gonzalez, V. Matellan-Olivera, and L. Rodero-Merino. Studying the evolution of libre software projects using publicly available data. In *Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering*, pages 111–115, Portland, Oregon, 2003.

[13] G. Sabidussi. The centrality index of a graph. *Psychometirka 31, 581-606*, 1996.

[14] D. Watts and S. Strogatz. Collective dynamics of small-world networks. *Nature 393, 440-442*, 1998.