Community structure of modules in the Apache project

Jesús M. González-Barahona, Luis López, Gregorio Robles
Grupo de Sistemas y Comunicaciones – Universidad Rey Juan Carlos
{jgb,llopez,grex}@gsyc.escet.urjc.es

Abstract

The relationships among modules in a software project of a certain size can give us much information about its internal organization and a way to control and monitor development activities and evolution of large libre software projects. In this paper, we show how information available in CVS repositories can be used to study the structure of the modules in a project when they are related by the people working in them, and how techniques taken from the social networks fields can be used to highlight the characteristics of that structure. As a case example, we also show some results of applying this methodology to the Apache project in several points in time. Among other facts, it is shown how the project evolves and is self-structuring, with developer communities of modules corresponding to semantically related families of modules.

1. Introduction

Large libre software[1] projects are usually organized as a set of modules. Each one of them can correspond to a given program, library, or any other unit identified by the project as distinct. It is common that developers work in several modules, according to their interests, skills, and constraints, and that those modules to which they contribute change over time. The relationships among modules due to the people working in them constitutes a sort of social structure of the project. In some sense, those human relationships are the glue that maintain the whole project together, and the chains that contribute to spread information and uses from one part of the project to others. In this paper, we explore how those developer connections contribute to the making of the community structure of the project, how it can be identified and visualized and how it evolves over time.

The study and characterization of complex systems is a very fruitful research area nowadays with may interesting open problems. Special attention has been paid recently to complex networks, where graph and network analysis plays and important role and is gaining great popularity due to its intrinsic power to reduce a particular system to its simple components and relationships. Thanks to this, network characterization is widely used in many scientific and technological disciplines as neurobiology [1], computer networks [2] [3], linguistics [4], etc.

Among complex networks, social networks appear in a quite natural way as a method for analyzing the structure and interactions of people and groups of people within complex organizations [5][6][7][8][9]. To understand the structure of those networks, we are interested in determining how the different nodes interact and form groups that, in turn, interact with each other giving rise to higher order groups. The set of groups obtained, as well as their relationships, is which we call the community structure of the network.

All the information we need for such an analysis is available in the CVS repository of the project. Using it, we construct the network of modules at a given time, considering a link when there is a common set of developers that have contributed to both modules. Later, we apply on it some techniques from the social networks field.

There have been some other works analyzing social networks in the libre software world. [10] 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. [11] proposes also a sort of network analysis for libre software projects, but taking into account this time a technical connection between modules as code dependency is.

How libre software projects organize themselves in subprojects is an issue that has been already discussed in literature. For instance, in [12] it is argued that projects that require a number of core developers that is larger than a given amount (10 to 15 persons) create in effect several related projects if no other means of code control are introduced. In the conclusions of this paper we will discuss the validity of this hypothesis.

In the rest of this paper we detail the methodology we are using to build the network of modules, and how we identify communities in that network. Later, we show some results of applying this methodology to the Apache

[1] Through this position paper we mean by libre software any program which is either open source software (according to the Open Source Initiative definition) or free software (according to the Free Software Foundation definition).

project, and present some conclussions and future lines of work.

2. Methodology

The network which we use to identify communities is built as follows. From the CVS repository we extract, for each module, the list of people committing changes to it, and the number of of those changes (commits) [13]. With this list we compute, for each pair of modules, the set of committers who have collaborated in both modules and the number of commits they have contributed to them. Using these data, we can build a network by considering modules as vertices, and establishing a link among any two vertices if the corresponding modules have common committers. We use wheighted edges, with the wheight being proportional (although not linearly) to the number of commits that common authors have contributed[2].

To obtain the community structure of the resulting network, we perform community analysis based on the Girvan-Newman (GN) algorithm [14] which has been proved to be one of the most accurate[3] (see Figure A-1), and has already been successfully used to obtain the informal collaboration network of organizations (showing that it is self-similar [17]). The GN algorithm produces as output a minimal binary expanding tree, in which we have used colored[4] vertices (modules) to show visually how the extracted communities follow a certain pattern. The size of a vertex is proportional to the size of the module in LOC (lines of code), so that bigger modules can be identified visually.

3. A case study: the Apache project

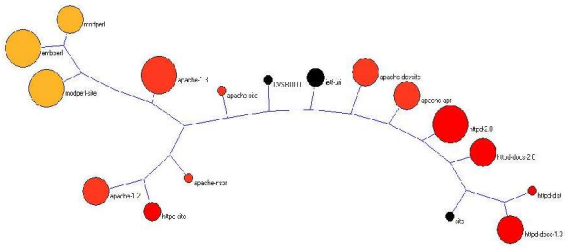

Illustration 1 Structure January 1st, 1999

We have applied the described methodology to the Apache project CVS repository. To color the nodes, we have followed in general the naming conventions of the project, which correspond to families of modules. That way, we have used red for those modules related to the HTTP server (apache/httpd), orange for those related to modperl, green for XML related modules, dark blue for Jakarta, light blue for Avalon, yellow for TCL-related modules and black for generic modules like CVSROOT (administrative files of the CVS repository) or the web site (site).

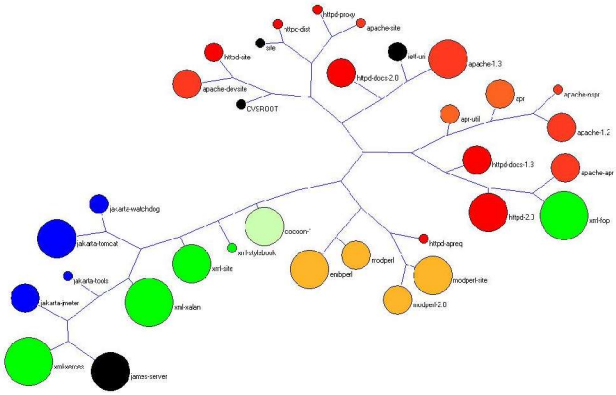

Illustration 2 Structure January 1st, 2000

Thanks to the coloring, it can be observed how the project self-structures itself. For instance, in January, 1999 (Figure 1), modules related to modperl are seen only in the upper-left branch, which means that those modules have already some differences in authorship with the rest of the project.

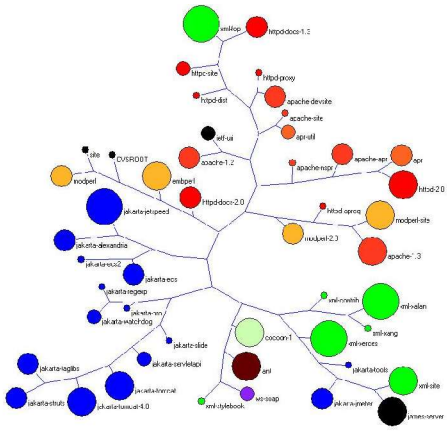

Illustration 3 Structure September 1st, 2000

Later, in January 2000 (Figure 2), the community of modperl modules is also clearly seen as a branch (in the center of the figure). Meanwhile, a branch for Jakarta and XML-related modules can be identified (left bottom). Nine months later (Figure 3) Jakarta has differentiated itself (two close branches at the bottom of the figure) from XML-related modules (branch at the bottom right).

[2] We have used other weights for the edges, for instance the number of aggregated LOC (lines of code) committed by the common developes instead of the number of commits. However, in the cases we have studied the resulting networks are very similar.

[3] Other different alternatives proposed in the literature for obtaining the community structure of a social network can be seen in [15] [16].

[4] There exists a colored on-line version of this paper at http://libresoft.dat.escet.urjc.es/html/downloads/woss-icse-2004.pdf

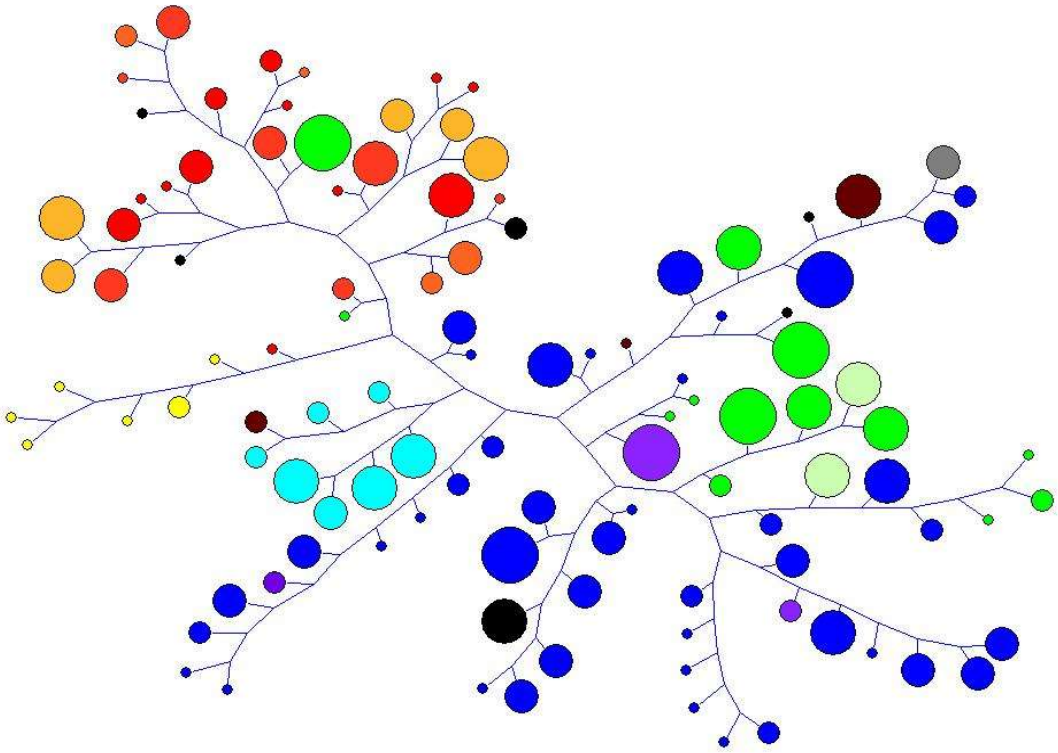

Illustration 4 Community structure of the Apache project. January 1st, 2002

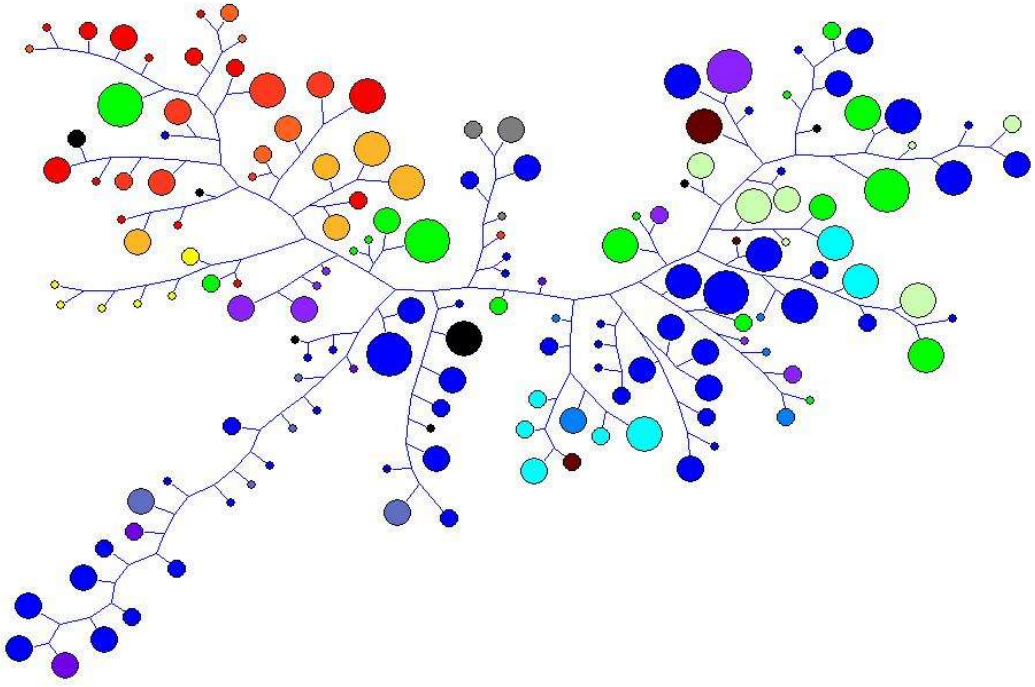

Illustration 5 Community structure of the Apache project. February 1st, 2004

On the other hand, modperl modules are now spread through several branches, mixed with the apache/httpd modules (top part of the figure).

The figures in early 2002 and 2004 (Figure 4 and Figure 5) show how modperl is still somewhat spread through the project, while there are clear communities (branches) for the apache/httpd (red, not too split in large branches), Avalon (light blue, clearly differentiated as a community), Jakarta (dark blue, present in several different communities, intermixed with green, XML-related modules) and Tcl related modules (yellow community).

4. Conclusions and further work

In the Apache project, the structure seen after several years of development is rather similar to that in early 1999, which seems to be the epoch in which it stabilized. It is interesting to see how, even when the project grows a lot during that time, the conservation of the structure means that the pairs of nodes related by the developers working on them remain more or less the same. In addition, it is clear that developers tend to work in the projects within its thematic community (apache/httpd, Jakarta, TCL, Avalon, etc.), but even within these communities there are clear differentiations (see for instance the many large branches of the Jakarta-XML modules).

One of the more impacting cases that come into view through the evolution of the structure of the Apache project is how apache/httpd-related modules (red) and Jakarta-related modules (blue) represent separated communities in the computed binary trees, in all dates during the life of the project. Of course, technological aspects are a key fact for understanding this (Jakarta is mainly based in Java, while apache/httpd uses basically C).

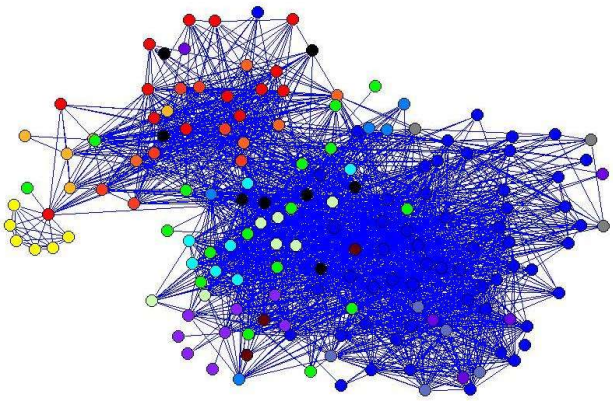

Illustration 6 "Classical" network analysis. Feb 1st 2004

Regarding our methodology, we believe this technique can be used to obtain and compare the community structure of modules in large libre software projects. From this structure, it can be inferred whether there are or not strong communities within the projects, the evolution of these communities, and their interrelationships. These implications can be used to estimate information flow within the project, collaboration patterns, and other important characteristics of the project.

In comparison to previous works, such as [10] this view is a step forward at least in the aspect of offering a way to visually identify an affiliation network. Such a classical network analysis would offer a graph as in Figure 6, where the structure can be hardly infered.

Inspite of the findings in [12] this analysis does not show an effective splitting of modules when the size of the main contributing group exceeds a certain number of persons. In fact, two other behaviours can be deduced: the emergence of new modules around consolidated ones and the specialization of developers in time (which can be can be derivated by the fact that a branch splits in two).

This paper offers also a first dynamic vision of the evolution of the structure, although future work on this issue is pending, specially regarding relationships between the different graphs in time.

A. The Girvan-Newman algorithm

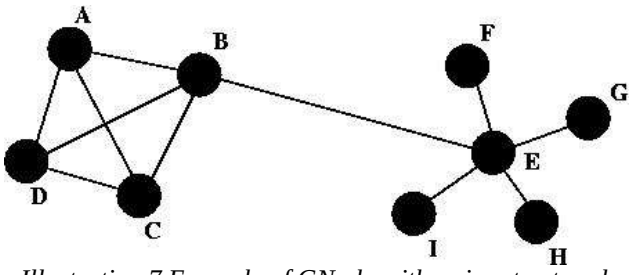

Illustration 7 Example of GN algorithm: input network

When we apply to the network in Figure 7 the GN algorithm, we get as output the tree in Figure 8, which represents its community structure. The GN algorithm proceeds by splitting the network recursively until single nodes are left. The information about the community structure of the original network can be deduced from the topology of the binary tree that represents this splitting procedure. Basically, each branch of this tree can be seen as a community of nodes of the original network.

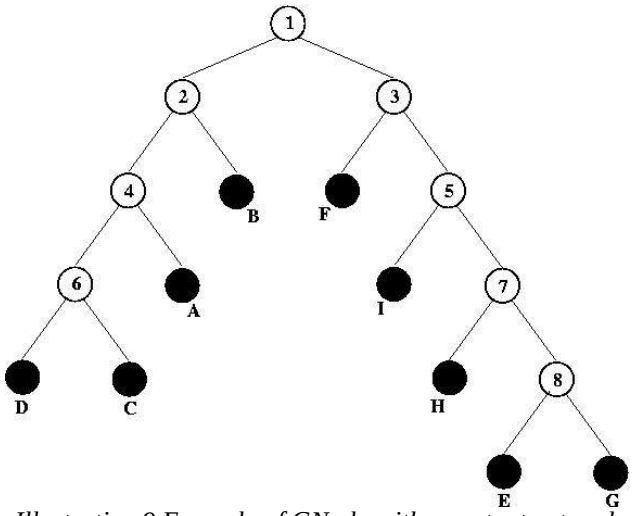

Illustration 9 Example of GN algorithm: output network

The GN algorithm is based on a parameter called the betweenness of edges, which measures the number of shortest paths connecting pairs of nodes which go through that edge [5]. Edges connecting high clustered communities must have higher betweenness. So, the GN algorithm proceeds by calculating the betweenness of all edges and eliminating the edge with the highest betweenness. These steps are repeated until the network is split into two connected components. After that, the process is recursively executed on each of the two separate components. To represent the splitting process, we use a binary tree which is built in the following way: each time a split is carried out, we add a virtual node in the tree (marked with numbers in Figure A-2) which is connected to the two novel components. When these components are single vertices of the original network, we add them as terminal nodes of the tree (marked with letters in Figure A-2).

10. References

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

[2] R. Albert, A.-L. Barabasi, H. Jeong, and G. Bianconi, Power-law distribution of the World Wide Web, Science 287 2115a (2000).

[3] R.F. Cancho and R. Sole, The small world of human languaje, Proc. R. Soc. 268, 2261-2265, 2001.

[4] R. Kunar, P. Raghavan, S. Rajagopalan, and A. Tomkins, The Web and social networks, IEEE Computer 35(11) 32-36, 2002.

[5] M.E.J. Newman, Scientific collaboration networks: I. Network construction and fundamental results, Phys. Rev. E 64, 016131 (2001).

[6] M.E.J. Newman, Scientific collaboration networks: II. Shortest paths, weighted networks, and centrality, Phys. Rev. E 64, 016132, 2001.

[7] R. Guimera, A. Daz-Aguilera, F. Vega-Redondo, A. Cabrales, and A. Arenas, Optimal network topologies for local search with congestion, Phys. Rev. Let. 89, 248701, 2002.

[8] L. Lopez and M.A.F. Sanjuan, Relation between structure and size in social networks, Phys. Rev. E. 65, 036107, 2002.

[9] L. Lopez, J.F. Mendes, and M.A.F. Sanjuan, Hierarchical social networks and information flow, Physica A, 316, 591-604, 2002.

[10] Greg Madey, V. Freeh, and R. Tynan, The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory, 2002, http://www.nd.edu/~oss/Papers/amcis_oss.pdf.

[11] Rishab Aiyer Ghosh, Clustering and dependencies in free/open source software development: Methodology and tools, Abril 2003, http://www.firstmonday.dk/issues/issue8_4/ghosh/index.html.

[12] Audris Mockus, Roy T. Fielding, and James D. Herbsleb, Two Case Studies of Open Source Software Development: Apache and Mozilla, http://www.research.avayalabs.com/techreport/ALR-2002-003-paper.pdf.

[13] Gregorio Robles, Jesús González-Barahona, José Centeno-González, Vicente Matellán-Olivera, and Luis Rodero-Merino. http://opensource.ucc.ie/icse2003/3rd-WS-on-OSS-Engineering.pdf.

[14] M. Girvan and M. E. J. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821-7826, 2002.

[15] M.E.J. Newman, Detecting community structure in networks., Eur. Phys. J. B, in press. http://www.santafe.edu/~mark/pubs.html.

[16] M.E.J. Newman and M. Girvan, Finding and evaluating community structure in networks., Phys. Rev. E, in press. http://www.santafe.edu/~mark/pubs.html.

[17] R. Guimera, L. Danon, A. Daz-Aguilera, F. Giralt, and A. Arenas, Self-similar community structure in organizations, http://arxiv.org/pdf/cond-mat/0211498.