Chapter 5.6
Applying Social Network Analysis Techniques to Community-Driven Libre Software Projects

Luis López-Fernández
Universidad Rey Juan Carlos, Spain

Gregorio Robles
Universidad Rey Juan Carlos, Spain

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos, Spain

Israel Herraiz
Universidad Rey Juan Carlos, Spain

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 article, 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 shows some relevant results which open new paths in the understanding of the informal organization of libre software development communities.

INTRODUCTION

Software projects are usually the collective work of many developers. In most cases, and especially in the case of large projects, those

developers are formally 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 do 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 (1997) and later complemented by some more formal models of nonhierarchical coordination (Elliott & Scacchi, 2004; Healy & 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 (Healy & Schussman, 2003; Krishnamurthy, 2002), 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 superlinear growth even after reaching a size of several millions of lines of code (Godfrey & Tu, 2000; Robles, Amor, Gonzalez-Barahona, & Herraiz, 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 indicative of an efficient organizational structure.

On the other hand, the study of several large libre software projects has shown evidence about the unequal distribution of the contributions of developers (Dinh-Trong & Bieman, 2005; Koch & Schneider, 2002; Mockus, Fielding, & Herbsleb, 2002). 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 (Dinh-Trong & Bieman, 2005; Mockus et al., 2002). This is what has been called the *onion* structure of libre software projects (Crowston, Scozzi, & Buonocore, 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, and so forth. Modules are usually supposed to be built maintaining the interrelationships to a minimum, so that independent evolution is possible (Germán, 2004a).

In this article, a new approach is explored in order to study the informal structure and organization of the developers in large libre software