Association for Information Systems
AIS Electronic Library (AISeL)

ICIS 2008 Proceedings

International Conference on Information Systems (ICIS)

2008

The Impact of Social Netowrking on Software Design Quality and Development Effort in Open Source Projects

Donato Barbagallo
Politecinco di Milano, barbagallo@elet.polimi.it

Chlara Francalenei
Politecnico di Milano, francala@elet.polimi.it

Francesco Merlo
Politecnico di Milano, merlo@elet.polimi.it

THE IMPACT OF SOCIAL NETWORKING ON SOFTWARE DESIGN QUALITY AND DEVELOPMENT EFFORT IN OPEN SOURCE PROJECTS

L'impact des réseaux sociaux sur la qualité de la conception logicielle et l'effort de développement dans les projets de logiciels libres

Completed research paper

Donato Barbagallo
Dipartimento di Elettronica e Informazione
Politecnico di Milano
barbagallo@elet.polimi.it

Chiara Francalanci
Dipartimento di Elettronica e Informazione
Politecnico di Milano
francala@elet.polimi.it

Francesco Merlo
Dipartimento di Elettronica e Informazione
Politecnico di Milano
merlo@elet.polimi.it

Abstract

This paper focuses on Open Source (OS) social networks. The literature indicates that OS networks have a few nodes with a number of relationships significantly higher than the network's average, called hubs. It also provides numerous metrics that help verify whether a node is a hub, called centrality metrics. This paper posits that higher values of centrality metrics are positively correlated with project success. Second, it posits that higher values of centrality metrics are positively correlated with the ability of a project to attract new contributions. Third, it posits that projects with greater success have a lower software design quality. Hypotheses are tested on a sample of 56 applications written in Java from the SourceForge.net online OS repository. The corresponding social network is built by considering all the contributors, both developers and administrators, of our application sample and all contributors directly or indirectly connected with them within SourceForge.net, with a total of 57,142 nodes. Empirical results support our hypotheses, indicating that centrality metrics are significant drivers of project success that should be monitored from the perspective of a project administrator or team manager. However, they also prove that successful projects tend to have a significantly lower design quality of software. This has a number of consequences that could be visible to users and cause negative feedback effects over time.

Keywords: Open source, social networks, network centrality metrics, software development effort, software design quality.

Résumé

Cette recherche se focalise sur l'impact des réseaux sociaux dans les communautés de logiciels libres. Les résultats empiriques prouvent que les indicateurs de centralité des réseaux sociaux sont des mesures significatives du succès des projets de logiciels libres. Toutefois, les résultats prouvent aussi que les projets les plus réussis ont une faible qualité de conception logicielle.

Abstract in Native Language

La ricerca si focalizza sull'impatto delle reti sociali nel contesto delle comunità Open Source (OS). I risultati empirici indicano che le metriche di centralità sono indicatori significativi del successo di un progetto nell'ambito di una comunità OS. Tuttavia, i risultati provano anche che i progetti di maggior successo sono caratterizzati da una minor qualità di progettazione interna.

THE IMPACT OF SOCIAL NETWORKING ON SOFTWARE DESIGN QUALITY AND DEVELOPMENT EFFORT IN OPEN SOURCE PROJECTS

Introduction

Social networks represent social systems characterized by very large numbers of individuals and relationships among individuals. Human variables, such as the behavior of individuals and the consequent evolution of their mutual relationships, are inherently subject to uncertainty and, therefore, difficult to predict. Due to their size and to the difficulty of modeling and forecasting their evolution, social networks are considered *complex systems* (Wasserman and Faust 1994; Albert and Barabàsi 1999). However, their large number of individuals suggests that even if the network is difficult to model in the small, in the large it might follow a probabilistic evolution.

Most research has focused on the empirical analysis of the evolution of social networks in order to gather the laws of their probabilistic behavior. These empirical laws vary with the purpose and nature of the specific social community (Albert and Barabàsi 1999). This paper focuses on Open Source (OS) software communities. In this context, the individuals constituting the network are the contributors of OS projects and a relationship ties two individuals if they take part in the same project. Contributors can play different roles in the project, typically developer or administrator, and can provide a more or less significant contribution depending on their commitment to the project. These simple characteristics of OS communities are consistently traced by the online repositories of OS software, such as *SourceForge.net* or *Tigris.org*, and represent the most widely studied variables of OS social networks. By empirically analyzing the evolution of OS networks modeled as sets of cooperation relationships among project contributors, previous research has verified the following general laws:

- The probability with which a new relationship connects to a contributor is exponentially proportional to the number of existing relationships involving the contributor (Xu et al. 2006; Gao and Madey 2007). This law applies to a variety of social networks and is often referred to as the *rich-get-richer* evolution principle (Albert and Barabàsi 1999).
- As a consequence of the *rich-get-richer* principle, OS networks have a few nodes with a number of relationships significantly higher than the network's average, called *hubs*. The alternative random evolution principle has been found not to apply to OS networks, i.e. relationships are not uniformly distributed across nodes (Xu et al 2006).
- Hubs have been found to follow a life cycle. Hubs appear, grow, and ultimately stop growing and quickly disappear (Gao et al. 2005).
- The creation of hubs is enabled by the ability of some nodes to evolve considerably faster than the average evolution rate of the network (Gao et al. 2005).

Note that in contexts different from OS communities, in particular within companies, the general rules of social networking can be difficult to verify, as the natural evolution of communities is biased by the structure, regulations, and social norms of the organization (Hossain et al. 2006). On the contrary, OS social networks are freely accessible with almost no entry barriers (Xu et al. 2006). They are broad and easy to study in the large. Their members contribute to the network by performing knowledge-intensive activities, such as software development, and, as a consequence, the value of an OS network is in the human capital that it aggregates.

In the OS context, Grewal et al. (Grewal et al. 2006) have hypothesized that hub contributors have a positive impact on the success of the projects they are involved in. Their claim is that the *rich-get-richer* principle suggests that hub contributors have the ability to attract further contributions and, thus, positively influence the evolution of their projects. The literature provides numerous metrics that help verify whether a node is a hub, called *centrality* metrics (see the "Related Work" section). Grewal et al. (Grewal et al. 2006) have tested whether higher values of centrality metrics are positively correlated with the *ranking* measure of project success. By testing correlation for 12 projects from *SourceForge.net* written in Perl, Grewal et al. (Grewal et al. 2006) have found mixed results that only partially support their hypotheses. A possible problem with their approach to testing is the size of the social network that they

have considered, which is limited to the contributors of a few projects and their direct connections to other contributors in the *SourceForge.net* community, while the literature clearly indicates that the laws governing a social network can be observed only if the network is analyzed in the large (cf. Newman et al. 2006).

Despite their non-conclusive results, Grewal et al. (Grewal et al. 2006) have put forward a correlation hypothesis that is consistent with interesting previous findings on other types of social networks. Reagans and McEvily (Reagans and McEvily 2003) have explained how higher values of centrality metrics affect people's motivation to invest time and energy in sharing knowledge. In general, centrality metrics have been found to act as enablers of knowledge transfer processes (Tsai 2005) and related effectiveness. These results seem to confirm the role of centrality in determining the success of knowledge intensive processes.

The relationship between centrality and project success has clear implications for managers. A company that is interested in making business through OS could aim at having its developers become hubs as a way to enhance success. As for any strategy to success, the cost-to-quality ratio of centrality should be assessed first. The goal of this paper is to help the assessment of this ratio by empirically investigating the cost and quality advantages of centrality and by reconsidering the hypothesis of Grewal et al. (Grewal et al. 2006) tying centrality to project success. In particular, the following research questions are addressed in this paper:

- Is centrality related to project success?
- What is the relationship between centrality and cost?
- Do projects that reach success by leveraging centrality also show higher quality?

By addressing these questions, a fundamental contribution of this paper is to search for new laws governing the evolution of OS networks. Answering this paper's research questions does not help understand how a contributor can become a hub, that is, how it can grow faster than other contributors. Understanding the drivers behind the evolution of a few hub nodes seems more suitable for a direct inspection of their business characteristics rather than overall social network analyses. However, the paper is a first step towards understanding the real benefits of being a hub. This represents an important input to a cost-benefit analysis that should precede the operationalization of any business strategy.

The presentation is organized as follows. The "Related Work" section reviews the literature on social networking centrality metrics, software design quality metrics, and development effort models. The "Research Hypotheses" section discusses our research hypotheses, while the "Variables and Data Sample" section describes the operationalization of variables and the data sample used to verify our hypotheses. The "Empirical Results" section explains our statistical approach and reports the results of empirical testing, while the "Discussion and Conclusions" section provides a discussion of empirical findings and outlines possible directions for future research.

Related Work

This section reviews the literature focusing on the concepts of centrality in social networks, software design quality, and development and maintenance effort.

Centrality in Social Networks

The concept of centrality has been defined as the importance of an individual within a network (Freeman 1979). Centrality has attracted a considerable attention as it clearly recalls notions like social power, influence, and prestige. Over time, several metrics have been introduced to formalize and then measure centrality from different points of view.

In 1979, Freeman has introduced the first metric of centrality, called *degree centrality* (Freeman 1979). This metric is defined as the number of links of a node normalized to the total number of links in the network. Degree centrality still represents the simplest and most widely used indicator of centrality, as it is intuitive and easy to calculate (Choi et al. 2006). A node that is directly connected to a high number of other nodes is obviously central to the network and likely to play an important role (Sparrowe et al. 2001). A node with a high degree centrality has been found to be more actively involved in the network's activities (Hossain et al. 2006).

Freeman has also introduced the metric of *betweenness centrality* (Freeman 1979). This metric is defined as the average frequency with which a node is crossed by the shortest path connecting two generic nodes of the network.

This metric is widely used in the literature, as it represents the simplest way to measure the ability of a node to reach other nodes in the network and act as an intermediary of the interactions between them. Over time, several refinements of the original Freeman's metric have been proposed. For example, Newman (Newman 2003) has posited that a random walk among all possible paths should be considered as opposed to the shortest path. Although the opposite claim could be put forward too, Newman's metric has the advantage of lowering the complexity of the algorithm to calculate betweenness centrality. Note that a recent work by Burt (Burt 1997) extends the generality of betweenness centrality by suggesting a conceptual closeness between structural holes and betweenness centrality. Burt (Burt 1997) observes that the concept of structural holes is strongly based upon betweenness centrality.

Freeman has also proposed the metric of *closeness centrality*, which is meant to extend the concept of betweenness centrality by measuring how far an actor is from all other actors in the network along the overall shortest path (Freeman 1979). This metric is less intuitive and more difficult to calculate than the previous two and has obtained a more limited success. Freeman notes that closeness centrality can be associated with the idea of independence of a node, since high values of closeness involve a lower need to depend on other nodes in order to communicate with other parts of the network. However, the metric becomes meaningless if applied to disconnected networks, as it cannot be calculated for non-reachable nodes. A more recent metric proposed by Stephenson and Zelen (Stephenson and Zelen 1989), named *harmonic centrality*, represents a measure of closeness centrality that considers harmonic distance in place of shortest path distance.

Previous to Freeman, Bonacich (Bonacich 1972) introduced the metric of *eigenvector centrality*, which measures centrality as the principal eigenvector of the whole network's adjacency matrix. The disadvantage of this approach is that the eigenvector of the adjacency matrix must be calculated iteratively and convergence can be very slow. This metric has had limited success, especially due to its mathematical and computational complexity. Furthermore, Borgatti has noted that eigenvector centrality is conceptually similar to degree centrality (Borgatti 1995). However, it should be acknowledged that the *PageRank* metric proposed by Brin and Page (Brin and Page 1998) is based on the notion of eigenvector centrality. Even earlier than Bonacich, Katz (Katz 1953) and Hubbell (Hubbell 1965) introduced two centrality metrics, which, similar to eigenvector centrality, consider a node important if it is connected to other important nodes. However, both indices have series convergence issues that have limited their use in practice.

This paper focuses on degree and betweenness centrality, according to their original definition provided by (Freeman 1979). Degree and betweenness centrality represent the most intuitive and widely used metrics of centrality. We acknowledge that closeness and eigenvector centrality are also theoretically important metrics of centrality in the field of social networks, as discussed in (Borgatti et al. 2006). However, their greater conceptual and computational complexity makes them more difficult to use in empirical research on large social networks.

Software Design Quality

This paper focuses on the quality of software design, i.e. on the *internal* quality of software. Previous literature suggests that higher values of software design quality metrics represent drivers of a number of *external* quality variables, such as testability, correctness, and reliability (Boehm 1976; Brito e Abreu and Melo 1996; Barbey et al. 1998; Marinescu 2005). In turn, these external quality variables affect user satisfaction and can influence software adoption and actual usage (Bevan 1995, ISO/IEC 2004). However, the direct analysis of external quality variables, i.e. of software effectiveness variables, is outside of the scope of the present paper.

Software design quality can be measured by analyzing the design properties of source code. There exists a consolidated body of literature focusing on code-based design quality metrics. Traditionally, the measurement of code design quality is based upon *i)* complexity and *ii)* design quality metrics. The first research contributions were aimed at providing operating definitions and metrics of software complexity, focusing on the analysis of the code's information flow. Cyclomatic Complexity (McCabe 1976), Software Science (Halstead 1977), and Information Flow Complexity (Henry and Kafura 1981) represent the most widely used metrics from this early research.

Over time, design quality has become of increasing importance to cope with the continuously growing size of software systems. Research has started to distinguish between the complexity due to poor design quality and the inherent complexity of software due to requirements (Oviedo 1980; Troy and Zweben 1993). The main contribution of these studies has been to show that design quality is necessary to handle the complexity caused by challenging requirements.

With the advent of the object-oriented programming paradigm, coupling, cohesion, inheritance, and information hiding have been identified as the basic properties of software design quality (Emerson 1984; Symons 1988; Chen and Lu 1993; Sharble and Cohen 1993). Based on these four basic properties, a number of metrics have been proposed to evaluate the design quality of object-oriented software. The most widely known metrics have been first proposed by (Chidamber and Kemerer 1994) (WMC, NOC, DIT, RFC, LCOM, and CBO) and by (Brito e Abreu 1995) (COF, PF, AIF, MIF, AHF, and MHF). These milestone contributions have started a lively debate within the software engineering community on the consistency and generality of such metrics (Basili et al. 1996b; Chidamber et al. 1998; Harrison et al. 1998; Hitz and Montazeri 1996; Rosenberg 1998). As a matter of fact, metrics such as CBO, NOC, MIF, and DIT represent a standard and are included in most development environments, such as Eclipse and Visual Studio.NET. This paper focuses on these standard metrics.

CBO, NOC, and DIT have been found to impact on software maintainability and, hence, on maintenance effort and costs (Li and Henry 1993). Increasing software design quality is viewed as a costly activity that pays back in the long term by reducing the cost of subsequent maintenance interventions (Slaughter et al. 1998). With proprietary software, companies usually take a short-term perspective and tend to develop code faster at the expense of quality, which, in turn, tends to decrease over time (Tan and Mookerjee 2005). As observed by (Tan and Mookerjee 2005), the deterioration of quality over time leads to a break-even time when a short-term perspective becomes economically inefficient and companies should invest in quality. This can be obtained either by replacing old software with new code of higher quality or by launching a maintenance initiative aimed at increasing quality without necessarily developing new functionalities, commonly referred to as *refactoring* (Fowler et al. 2001; Mens and Tourwé 2004).

In OS applications these phenomena are difficult to observe. Some projects become inactive when they reach the end of their lifecycle and, until then, they are continuously maintained. However, projects reach their end for a number of reasons that may not be related to quality deterioration. For example, solo projects, i.e. projects launched and maintained by individual programmers, are often active for a very short period of time and come to an end due to lack of interest from the OS community.

The most successful projects, such as Linux and PostgreSQL, are still active although they are considered mature. Koch (Koch 2004) has noted that in OS projects refactoring tends to be a continuous process and developers allocate time and effort to quality improvements when needed. A previous work by Capra et al. (Capra et al. 2007) has studied the refactoring process of a sample of 95 OS applications (1251 versions) from *SourceForge.net*. Empirical analyses have showed that the number of versions between two subsequent refactorings is highly variable. On average, a significant quality improvement can be observed in 40% of the total number of versions, while Tan and Mookerjee (Tan and Mookerjee 2005) have found that in a sample of closed source applications refactorings occur in about 10% of an application's versions.

Previous literature indicates that the cost benefits of quality improvements are reaped over time. However, it provides only partial evidence to demonstrate that quality investments have a positive balance (Slaughter et al. 1998). From a theoretical standpoint, Tan and Mookerjee (Tan and Mookerjee 2005) suggest that quality investments typically represent a zero-sum game. However, the only clear empirical result is that quality involves an investment and, in the short term, it represents a cost. OS projects challenge also this result, since continuous refactoring practices should release similarly continuous cost benefits. A previous work by Capra et al. (Capra et al. 2008) has empirically verified that quality and development effort are not correlated in OS projects, supporting the theoretical observations of Tan and Mookerjee (Tan and Mookerjee 2005). In this paper, we consider software design quality and development effort as independent variables.

Development and Maintenance Effort Models

The literature provides several methodologies and models for software cost estimation (Boehm 1981; Symons 1988; Basili et al. 1996a; Boehm 2000; Ahn et al. 2003) and benchmarking (Kemerer 1987; Fenton and Pfleeger 1997). The typical goal of most of these models is to predict the effort required to develop or to maintain a software system. Prediction is based on measures of various software properties that have been found to influence effort, such as size, age, and complexity (Boehm 1981; Banker et al. 1991; Banker and Slaughter 1996; Banker and Slaughter 2000). Other approaches focus on the estimation of productivity drivers and resources based on the analysis of data gathered from change management systems, such as CVS or SVN systems (Mockus and Graves 2001; Ramil 2003).

From a social networking perspective, the influence of network structure on development and maintenance effort has been studied in globally distributed development groups of both OS and proprietary applications (Herbsleb and Mockus 2003; Gomes and Joglekar 2004; Monga 2004). These studies show that in highly geographically distributed groups social networks tend to become less effective, involving a greater development effort due to communication issues and delays. A typical consequence is the creation of sub-groups corresponding to smaller and, thus, more efficient social networks. In particular, Hinds and McGrath (Hinds and McGrath 2006) have empirically found that highly connected social networks (that is, networks with a high average value of degree centrality) cannot be considered as an effective support to distributed project development. Findings prove that the coordination overhead caused by the extension of the network increases the effort required for software development and maintenance. However, as noted by Ohira et al. (Ohira et al. 2005), a minimum degree of centrality in the social network is required to gather new contributions to the project. Communities that do not succeed in building an effective network experience difficulties in finding contributors, such as bug reporters and even new developers. From a managerial perspective, it is essential to find a trade-off between *i)* the need of gathering contributions to the project, and *ii)* the coordination and communication overhead associated with a broad social network.

Research Hypotheses

Our first research hypothesis aims at reconsidering the relationship between centrality and project success put forward by Grewal et al. (Grewal et al. 2006). While Grewal et al. (Grewal et al. 2006) have found mixed results, in contexts different from OS the literature acknowledges that social networks are a significant enabler of knowledge transfer processes and related effectiveness (Reagans and McEvily 2003; Tsai 2005). In particular, higher levels of betweenness centrality in the network have been found to be correlated to a greater timeliness in reaching new information, while higher levels of degree centrality have been found to influence the ability to gain access to a broader information base (Mehra et al. 2001).

In the OS context, a more central position in the social network surrounding a project can foster the growth of the community by means of the *rich-get-richer* effect (Gao and Madey 2007). In turn, a more central position can help gain new skills, gather new ideas on how to improve the project, solicit feedback from users, and even recruit new team members. Moreover, a more central role can help marketing the project through word of mouth, as discussed by (Krishnamurthy 2003), and increase the number of downloads and users. According to (Crowston et al. 2003), all these factors can be considered indices of success for an OS project. This leads us to our first research hypothesis:

H1: *Projects involving contributors with a higher level of centrality are more successful.*

As noted in the previous section, the literature has identified a trade-off between a positive effect of centrality on the ability to attract new contributions to a project and a negative effect of centrality on effort due to a more cumbersome coordination overhead. Both effects tend to increase the *overall* development and maintenance effort associated with a project. The more cumbersome coordination overhead caused by centrality should also increase the *unit* development and maintenance effort associated with a project. Contributors who must face a more cumbersome coordination can be supposed to devote a smaller time share to development and maintenance and, thus, develop or maintain a lower number of functionalities within the same time frame, with a consequently higher unit effort. In turn, this higher unit effort increases the overall effort.

The software engineering literature has showed that managing a project with a broad and far-reaching social network may or may not involve a higher unit effort depending on the governance practices adopted to manage the project (Slaughter 1998; Capra et al. 2008). For example, in OS projects developers may never meet in person and interact entirely by e-mail, forums, and IRC channels. This makes the decision-making process slower and possibly troublesome, as discussion is rarely synchronous. Community members can reside in different geographical locations, with significant time-zone differences that can delay decision making and cause reworks. Certain OS projects tend to be very informal, without roadmaps, deadlines, and tight schedules. Whereas this may lead to higher design quality, it may also translate into lower efficiency due to a lack of planning (Monga 2004). However, the literature observes that more formal governance practices can help mitigate these phenomena and, thus, help reduce unit effort (Goldman and Gabriel 2005). The governance practices of OS projects are diverse and there exist a number of projects, referred to as commercial OS, that apply traditional forms of formal project governance (Goldman and Gabriel 2005). Therefore, the negative effect of centrality on unit effort can be difficult to observe without a model of governance that helps discriminate between different types of projects.

We posit that the positive effect of centrality on the ability of a project to attract new contribution is instead visible and phrase H2 accordingly. In the "Empirical Results" section we also verify the correlation between centrality and unit development and maintenance effort in order to test whether data support the exclusion of the negative effect of centrality on effort. Central contributors have numerous and far reaching links to other individuals. This can be a valuable way to advertise the project through word of mouth marketing (Krishnamurthy 2003). As a consequence, the project can attract many potential community participants and even team members (Ohira et al. 2005). Each user that comes into contact with the project can interact in different ways with the community. The most trivial form of participation is the simple usage of the application, which increases the number of downloads. However, by using the application, a user is more likely to take part in the community by interacting with forums, mailing lists, or discussion groups. Furthermore, he or she can start reporting bugs or making requests for new functionalities. In some cases, the user might also become part of the project team, thus contributing to the actual development of the project (Howison and Crowston 2005). All these contributions represent an effort devoted to the project which positively contributes to the growth of the application and of the community. On the other hand, they also represent an opportunity to share development and maintenance costs within a broader community. These considerations are summarized by our second research hypothesis:

H2: *Projects involving contributors with a higher level of centrality are able to attract a greater development and maintenance effort.*

Note that a larger project is not necessarily more successful. Viceversa, a more successful project is not necessarily larger. However, hypotheses H1 and H2 claim that more central projects are both more successful and larger. In order to verify that hypotheses H1 and H2 are in fact independent, in the "Empirical Results" Section we test the correlation between the variables that operationalize success and effort. Clearly, more successful projects have a broader user base. As previously discussed, this leads to more frequent requests for new functionalities, a greater number of suggestions on how to improve the application, more bug reports that must be taken into account (Ohira et al. 2005). Previous literature indicates that project administrators do not only focus on the software product itself, but also "strive to create an environment and culture that fosters the sense of belongingness to the community" (Nakakoji et al. 2002). Satisfying the requirements of an evolving community of users requires a restructuring of the project team and a continuous change in how the development process is managed (Ye et al. 2004). These consequences of success may lead project administrators to overburdening themselves in order to preserve the cohesion among community members and remain attractive to potential new members.

Among successful projects, more central projects are likely to include hub nodes which tend to experience an exponential growth, as the probability to gain new relationships has been found to be exponentially proportional to the number of existing relationships (Xu et al. 2006; Gao and Madey 2007). As a consequence, the rate of new change requests (new features, bug reports, etc.) grows accordingly. This difference between growth rates of requests and project contributors raises further doubts as to whether administrators can adequately fulfill the burden of requests and suggestions induced by greater success. On the other hand, it is well known that ignoring the feedback from the community would be perceived as an indication of a scarce interest and support from the project team, with a negative impact on success (Crowston et al. 2003). We posit that the time needed to keep the community alive by taking care of requests and manage a growing project is spent to the detriment of the time devoted to development and/or bug fixing activities. The literature indicates that performing development and maintenance activities in haste typically leads to a decrease in the quality of source code (Austin 2001, Capra et al. 2007). It has also been noticed that when the project team is under pressure due to the number of requests from clients, quality tends to decrease (Tan and Mookerjee 2005). Tan and Mookerjee note that this decrease is systematic in proprietary software projects. Our claim is that project success adds pressure on the project team, increasing the number of requests that contributors receive and should fulfill. In turn, we posit that success is detrimental to software design quality. This leads to our third research hypothesis:

H3: *More successful projects have a lower software design quality.*

Variables and Data Sample

This section presents the operationalization of the variables involved in our testing and the data sample used for empirical verifications.

Variable Definition and Operationalization

Network model. We model OS social networks as two-mode undirected affiliation networks (Wasserman and Faust 1994) with two types of nodes: developers and projects. A node representing a developer, say *d*, is associated with another node representing a project, say *p*, when *d* is a member of *p*'s team of contributors. Two distinct one-mode networks can be derived from a two-mode network by considering either developers or projects only:

- *Developers network.* All nodes represent developers. Two nodes are linked when both developers are members of the same project team.
- *Projects network.* All nodes represent projects. Two nodes are linked when corresponding projects have at least one developer in common.

Figure 1 shows a sample two-mode network, along with the two corresponding one-mode networks.

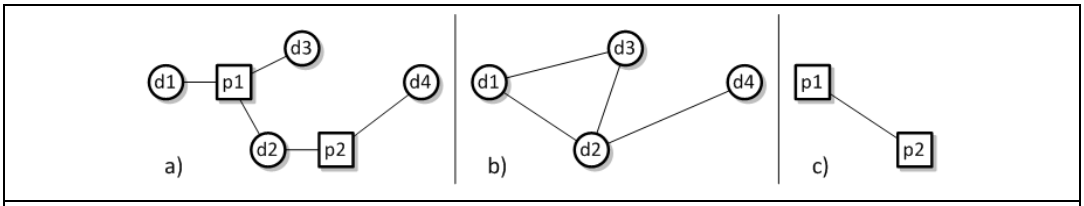

Figure 1. Network models: *a)* two-mode network (circles – developers, squares – projects); *b)* one-mode developers network; *c)* one-mode projects network.

Metrics of centrality. The *degree centrality* (Freeman 1979) $c_d(n_i)$ of node n_i is defined as the ratio of the number of edges involving node n_i, $\rho(n_i)$, to the total number of nodes in the network excluding node n_i:

$c_d(n_i) = \rho(n_i) / (N-1)$.

The *betweenness centrality* (Freeman 1979) $c_b(n_i)$ of node n_i is defined as the average frequency with which a generic node n_j crosses node n_i to reach a different node n_k trough a shortest path:

$$c_b(n_i) = \frac{2}{(N-1)(N-2)} \sum_{j<k} \frac{g_{jk}(n_i)}{g_{jk}},$$

where g_{jk} represents the total number of shortest paths from n_j to n_k and $g_{jk}(n_i)$ represents the number of shortest paths between nodes n_j and n_k crossing n_i. The metric is normalized to the maximum number of shortest paths crossing n_i in an undirected network with *N* nodes. Betweenness centrality is a measure of the ability of a node to control the information flows in the network. A node with a high betweenness centrality can be considered as an important information broker for the network, as it is likely to receive and convey many information flows.

Degree and betweenness centrality for developers' and projects' networks are indicated with apex *d* and *p*, respectively. For the sake of simplicity, we refer to the degree and betweenness centrality of nodes in the developers' networks as developer degree centrality (c^d_d) and developer betweenness centrality (c^d_b), respectively. Similarly, we refer to the degree and betweenness centrality of nodes in the projects' networks as project degree centrality (c^p_d) and project betweenness centrality (c^p_b), respectively.

Metrics of software design quality. Two of the most referenced suites of object-oriented design metrics have been included in our metrics' set, as suggested by (Harrison et al. 1998): the MOOD metrics' set for the evaluation of quality at the software system level (Brito e Abreu 1995), and the Chidamber and Kemerer metrics' suite for the evaluation of quality at the class level (Chidamber and Kemerer 1994). Table 1 reports the definition of the metrics in our set.

Table 1. Metrics of software design quality.

Metric	Definition	Reference
CBO	Distinct classes used by a given class (excludes inheritance relationships)	(CK 1994)
DIT	Maximum path length from a class to the root of an inheritance tree	(CK 1994)

MIF	Percentage of inherited methods	(Brito e Abreu 1995)
NOC	Number of immediate subclasses of a class	(CK 1994)

Metric of success. The success S_k of application k is measured as the ranking index of the application given by the *SourceForge.net* repository. Low ranking values are an index of high success, while high ranking indicates low success. As suggested by (Crowston et al. 2003; Howison and Crowston 2004; Crowston et al. 2006), ranking is operationalized as a function of several indices, namely the number of downloads, the project page views, and the development activity of team members:

$S_k = f(downloads) + g(page\ views) + h(development\ activity).$

The component *f(downloads)* evaluates the raw popularity of a project. The component *g(page views)* quantitatively evaluates how effectively a project is communicating by considering, for example, the activity of the bug tracking system and the number of posts in the mailing lists and forums of the project. Finally, the component *h(development activity)* measures the amount of work that a project team has performed in a given time frame, including the number of commits to the versioning system, the age of the latest released files and the frequency of login of the project's administrators. The complete expression of the ranking index is documented on the *SourceForge.net* web site[1].

Metric of age. The age A_k of application k is measured in days and is defined as the difference between the current date and the date of subscription of each project to the *SourceForge.net* repository.

Metric of size. The size of version i of application k is defined as the total number of methods $M_k(i)$ (public, private, protected, and package-only) of all the classes of application version i (external components and third-party libraries included in distribution packages are excluded).

Metric of development effort. Development effort is measured in man-days. The development time $t_k(i)$ of version i of application k is defined as the number of days elapsed between the release of version i and previous version $i-1$. Development effort $E_k(i)$ measured in man-days is obtained by multiplying development time $t_k(i)$ by the following correction factors:

- $n_k(i)$, the number of project administrators and developers;
- α_k, the average fraction of time each contributor (administrator or developer) devotes to the project;
- β_k, the percentage of active members of the project team.

Each correction factor is related either to project administrators and developers (thus leading to a total of six factors to be considered), in order to account for the different amount of effort spent by the two classes of project team members. The complete expression of development effort is:

$E_k(i) = t_k(i) \cdot [\alpha_k^{admin} \cdot \beta_k^{admin} \cdot n_k^{admin}(i) + \alpha_k^{devel} \cdot \beta_k^{devel} \cdot n_k^{devel}(i)].$

The unit development effort $e_k(i)$ for version i of application k is defined as the ratio between the development effort $E_k(i)$ and the total number of methods $M^+_k(i)$ added to version i with respect to the previous version *i-1*:

$e_k(i) = E_k(i) / M^+_k(i),$

where $M^+_k(i)$ is the number of new methods added to version i with respect to previous version *i-1* of application k and is defined as the set theoretical difference between the sets of method signatures of versions i and *i-1*. An *ad-hoc* developed tool was used to analyze the Java source code, extract the complete method signatures of each version, and compute the set theoretical difference.

Data Sample

The data set used for this study has been derived by analyzing a sample of OS community applications taken from the *SourceForge.net* repository. Since mining on line repositories (such as *SourceForge.net*) can lead to controversial results because of the varying quality of available data (Howison and Crowston 2004), a first sample of applications (AS1) has been selected according to the following criteria:

[1] http://alexandria.wiki.sourceforge.net/Statistics

- Project maturity: active and beta status or higher; inactive and less mature applications have been excluded because of their instability and low significance;
- Version history: at least 5 versions released;
- Programming language: Java;
- Domain: selected applications are uniformly distributed across the *SourceForge.net* domain hierarchy.

A second sample of applications (AS2) has been considered to allow the correct evaluation of the social networking metrics described in the previous section. Applications belonging to sample AS2 have been selected by relaxing some of the criteria used to select applications of sample AS1. AS2 includes all the active projects of *SourceForge.net* written in Java. This sample has been used to build a *SourceForge.net* social network as wide as possible, with the aim of overcoming network size limitations of previous research (Newman et al. 2006). Table 2 presents the summary statistics of application samples AS1 and AS2. As suggested by (Marcoulides and Sounders 2006), confidence intervals with $\alpha = 0.05$ have been computed for the variables involved in our hypotheses in order to assess our sample size. The confidence intervals have been verified to be in the $\alpha = 0.05$ range for all variables. Data on all the applications of samples AS1 and AS2 refer to June, 30th 2007 to guarantee the temporal consistency of the data sets.

Table 2. Descriptive statistics of application samples AS1 and AS2.

	Dataset	AS1		AS2
Variable	Symbol	Value	Conf. Int. ($\alpha/2 = .025$)	Value
Number of projects	N	56	-	29,836
Average number of versions per project		15.4	-	-
Total number of administrators	n^{admin}	90	-	36,948
Total number of team members	n^{devel}	378	-	57,142
Average Project age (years)	A	3.91	± 0.55	3.50
Average Project ranking	S	20,011	± 6,504	43,502
Average number of administrators per project		1.6	-	1.2
Average number of team members per project		6.8	-	1.9
Average degree of nodes in the projects' network	c^p_d	$1.60 \cdot 10^{-4}$	$\pm 1.45 \cdot 10^{-4}$	$5.01 \cdot 10^{-5}$
Average betweenness of nodes in the projects' network	c^p_b	$7.46 \cdot 10^{-4}$	$\pm 2.23 \cdot 10^{-4}$	$1.27 \cdot 10^{-4}$
Average degree of nodes in the developers' network	c^d_d	$2.05 \cdot 10^{-4}$	$\pm 1.53 \cdot 10^{-4}$	$7.75 \cdot 10^{-5}$
Average betweenness of nodes in the developers' network	c^d_b	$2.62 \cdot 10^{-5}$	$\pm 2.15 \cdot 10^{-5}$	$1.75 \cdot 10^{-5}$
Average Coupling Between Objects	*CBO*	2.55	± 0.28	-
Average Number of Children	*NOC*	0.20	± 0.05	-
Average Depth of Inheritance Tree	*DIT*	0.27	± 0.08	-
Average Methods Inheritance Factor	*MIF*	0.21	± 0.06	-
Average Size	$M_k(i)$	1,517	± 481	-
Average Development Effort	$E_k(i)$	74.30	± 34.26	-
Average Unit Effort	$e_k(i)$	0.10	± 0.03	-

The number of project administrators and developers that we have used is the one officially listed by *SourceForge.net*. Factors α_k and β_k of our effort metric have been empirically estimated by surveying 570 OS project administrators and developers of *SourceForge.net*, including all the 378 team members involved in the projects of our sample AS1. The questionnaire has been customized to individual contributors in order to gather the actual time devoted by each respondent to each project he or she was involved in. We have received a total of 134 answers, with a global response ratio of 23.5%.

The value of the α factor has been calculated as the ratio between the total amount of time spent by each developer on a project and a full-time employment week of 40 hours. The value of the β factor has been calculated as the ratio between the number of respondents who declared to be actively involved in a project and the total number of respondents to our survey multiplied by the total number of projects they were involved in. Table 3 presents a summary of the overall results of the survey, along with the average values of correction factors.

Table 3. Summary statistics from the survey on development and maintenance effort.

Team member class	n total	n active	Average (hr/week)	St. Dev.	Conf. Int. (α/2 = .025)	α	β
Administrator	32	29	8.09	±9.65	±4.24	0.20	0.91
Developer	102	56	8.62	±12.55	±3.86	0.22	0.55
Global	134	87	8.35	±11.02	±2.70	0.21	0.65

The resulting evaluation of development effort can be clarified by means of a numerical example. Let us consider two versions of the JEdit project (4.2pre6 and 4.2pre7), released on Oct 26th, 2003 and Dec 1st, 2003, respectively. From the release dates, it is clear that the development activities of version 4.2pre7 lasted 35 days. The project's homepage on SourceForge.net reports that the project team is composed by 9 administrators and 110 developers, thus $n^{admin}_{JEdit} = 9$ and $n^{devel}_{Jedit} = 110$. From the results of our survey, $\beta^{admin} = 0.91$ and $\beta^{devel} = 0.55$. The average weekly effort spent in development activities by administrators is 8.09 hours, while by developers is 8.62 hours, that is $\alpha^{admin} = 0.20$ and $\alpha^{devel} = 0.22$. Based on these data, the total development effort for version 4.2pre7 is:

$E_{JEdit}(4.2pre7) = 35 \cdot [0.20 \cdot 0.91 \cdot 9 + 0.22 \cdot 0.55 \cdot 110] = 523.18$ man-days

Considering that the number of new methods of version 4.2pre7 is $M_{JEdit}(4.2pre7) = 5037$, the unit development effort is:

$e_{JEdit}(4.2pre7) = 523.18 / 5037 = 0.104$ man-days/method (~ 50 minutes).

Tools for Data Analyses

The metrics of social networking have been evaluated on the social network including the projects of sample AS2 and related team members. The information necessary to build this social network has been gathered from the *SourceForge.net* data warehouse (Gao et al. 2007) for all applications in sample AS2 and processed with an *ad-hoc* developed software tool. The tool converts the raw data collected from the *SourceForge.net* data warehouse into the correct format to be supplied to the tool used to compute the metrics of social networking, i.e. Pajek (Batagelj and Mrvar 1997), one of the most used and referenced tools for the analysis of large networks.

The metrics of software quality have been evaluated by analyzing the source code of all available versions of each project in dataset AS1. Source code has been analyzed with an *ad-hoc* developed tool. The tool provides data on all the software quality metrics reported in Table 1, by performing static analyses of the Java source code. The static analysis engine is based on the Spoon compiler (Pawlak 2005), which provides the user a representation of the Java abstract syntax tree in a meta-model that can be used for program processing.

Statistical analyses and structural equation model testing have been performed with SPSS and AMOS, respectively.

Empirical Results

Measurement model

The measurement model has been defined to verify the assumption that social networking and software design quality metrics represent two coherent aggregate properties of the same phenomenon. A principal component analysis (PCA) has been performed on both sets of metrics to verify this assumption.

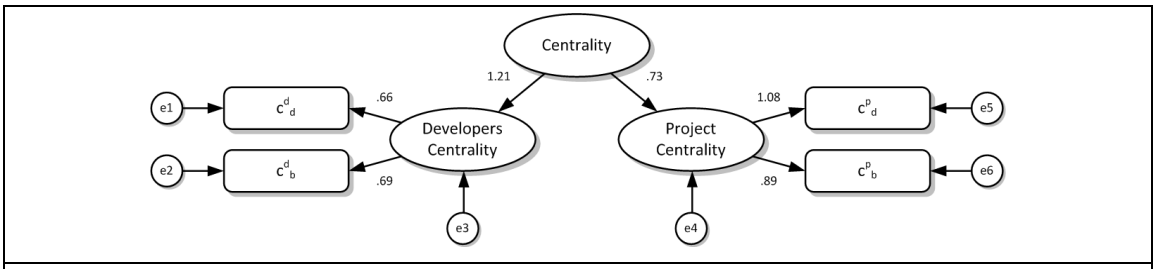

Figure 2. Measurement model of centrality.

Figure 2 shows the measurement model of the metrics of social networking. As it can be noted, two latent variables have been identified, related to the centrality of developers and of projects, respectively. These two latent variables have been found to represent different aspects of a same concept that we have called *Centrality*.

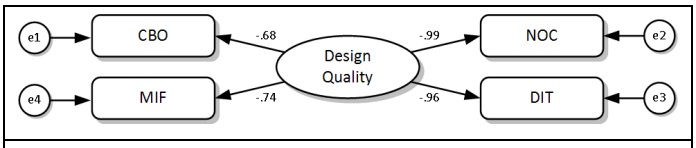

Figure 3. Measurement model of design quality.

Figure 3 presents the measurement model of the metrics of design quality. One factor has been extracted and labeled as *Design Quality*.

Table 4. PCA results and estimates of regression weights for the measurement models of Figure 2 and Figure 3.

Dependent Variable	Independent Variable	Standardized Regression Weight (b)	Standard Error	*p*-value	Composite Factor Reliability
c^d_b	Developer Centrality	0.689	-	< 0.001	0.684
c^d_d	Developer Centrality	0.660	1.289	< 0.001	0.830
c^p_b	Project Centrality	0.895	-	< 0.001	0.915
c^p_d	Project Centrality	1.081	0.049	< 0.001	0.970
Developers Centrality	Centrality	1.206	-	< 0.001	-
Project Centrality	Centrality	0.733	1.692	< 0.001	-
NOC	Design Quality	-0.988	-	< 0.001	0.911
CBO	Design Quality	-0.680	0.543	< 0.001	0.740
DIT	Design Quality	-0.964	0.078	< 0.001	0.900
MIF	Design Quality	-0.738	0.101	< 0.001	0.894

Table 4 shows the results of PCA along with the standardized regression weights of the relationships between latent and observed variables. Results show that all the factorizations can be accepted, since all the values of the composite factor reliability are greater than the threshold value of 0.70, as suggested by (Bagozzi and Yi 1988; Fornell and Larcker 1981). All the relationships considered between observed and latent variables are significant with $p < 0.001$. This confirms that the factorizations in the measurement model were performed correctly.

Structural model testing

The research model of Figure 4 is used to test our research hypotheses. The estimation results of the research model of Figure 4 are reported in Table 5. For sake of simplicity, the model in Figure 4 does not show the factorizations leading to the latent variables *Design Quality* and *Centrality* discussed in the previous section. *Development Effort*, *Size*, *Design Quality*, and *Centrality* are controlled by project age A_k, as suggested by (Banker and Slaughter 2000).

Although Figure 4 does not show the controlling variable *Age* (A_k), it has been considered in the model, as reported in Table 5. Since the *Development Effort* metric is an absolute measure of the maintenance and development effort spent for the project, it has been controlled with variables *Unit Effort* and *Size*, in order to compare the data related to different applications, as suggested by (Banker and Slaughter 2000). By controlling for *Unit Effort* we also isolate the positive effect of centrality on the ability to attract new contributions to a project from the negative effect of centrality caused by an increase in unit development and maintenance effort (see the "Research Hypotheses" section).

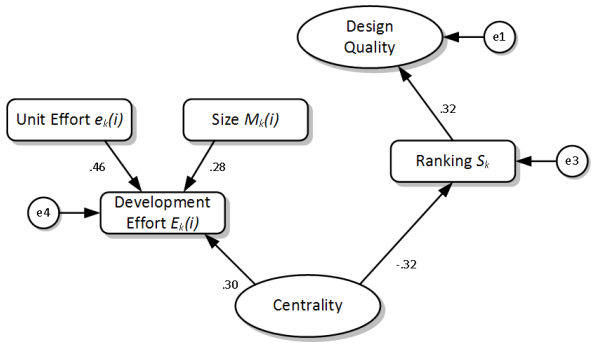

Figure 4. Structural model for the verification of research hypotheses.

Table 5. Estimates of regression weights for the research model of Figure 4.

Dependent Variable	Independent Variable	Standardized Regression Weight (b)	Standard Error	*p*-value
Development Effort	Centrality	0.366	2.290E05	0.002
Development Effort	Size	0.357	0.008	0.002
Development Effort	Unit Effort	0.382	106.887	<0.001
Ranking	Centrality	-0.322	4.983E07	0.019
Design Quality	Ranking	-0.316	0.000	0.010
Development Effort	Age	0.099	0.000	<0.001
Size	Age	0.437	0.000	<0.001
Design Quality	Age	0.296	0.000	0.016
Centrality	Age	0.561	0.000	<0.001

All the relationships hypothesized between model variables are significant with $p < 0.05$, that is, can be accepted at a significance level $\alpha = 95\%$. To assess the overall fit of the model several indices have been considered, as suggested by (Kline 2004). Table 6 shows that the overall model fit is satisfactory. All goodness-of-fit indices of the SEM estimation are above the recommended threshold values, except for TLI, which, nevertheless, is very close to the acceptance threshold. Table 6 reports the reference justifying the threshold value of the corresponding goodness-of-fit index.

Table 6. Goodness-of-fit indices for the research model of Figure 4.

Index	Research Model	Desired Level	Reference
χ^2	108.672	-	-
d.f.	58	-	-
χ^2/d.f.	1.874	< 3.0	(Carmines and McIver 1981)

p-value	< 0.001	< 0.01	-
IFI	0.914	> 0.90	(Bollen 1989)
TLI	0.880	> 0.90	(Tucker and Lewis 1973)
CFI	0.911	> 0.90	(Bentler 1990)

Research hypothesis H1 (projects involving contributors with a higher level of centrality are more successful) is represented within our model by the regression relationship between *Centrality* (independent variable) and *Ranking* (dependent variable). Our analysis shows that the regression weight is negative and statistically significant ($b = -0.322$, $p = 0.019$). Consequently, research hypothesis H1 is verified. Note that a decrease of *Ranking* corresponds to a higher level of success of the application.

Hypothesis H2 (projects involving contributors with a higher level of centrality are able to attract a greater development and maintenance effort) is tested by the relationship between *Development Effort* and *Centrality*, where the former is the dependent variable. Since the regression weight is positive ($b = 0.366$) and the estimation of this relation is significant ($p = 0.002$), hypothesis H2 is verified. Note that the *Development Effort* variable addressed by hypothesis H2 is not correlated to the *Ranking* variable. The Pearson's correlation index between the two variables is $\rho = 0.26$, suggesting that no evident relation can be supposed to persist between the two variables.

Finally, hypothesis H3 (more successful projects have a lower software design quality) is tested by the relationship between *Design Quality* (dependent variable) and *Ranking* (independent variable). As the regression weight is positive and significant ($b = 0.316$, $p = 0.010$), H3 is verified. Note that, since a decrease of *Ranking* corresponds to a higher level of success of the application, better ranking leads to lower design quality.

Table 7 summarizes the main results of our testing, by specifying the metrics involved in verifying different hypotheses.

Table 7. Summary overview of research hypotheses and results.

Hypothesis	Metrics	Supported results
H1	- Centrality - Success	Higher levels of centrality lead to greater success.
H2	- Centrality - Development Effort	Higher levels of centrality are able to attract a higher total development effort.
H3	- Success - Design Quality	Greater success leads to a lower design quality.

Discussion and Conclusions

Empirical results support hypothesis H1, thus confirming centrality as a significant driver of project success. Therefore, success seems to represent a first, fundamental benefit of being a hub node in an OS network. Previous to our work, Grewal et al. have hypothesized the causal relationship between centrality and project success, but have found mixed results that they have not considered conclusive to support their hypothesis (Grewal et al. 2006). As noted in the "Introduction" section, a possible problem with their approach to testing is the size of the social network that they have considered, which is limited to the contributors of 12 projects written in Perl and their direct connections to other contributors in the *SourceForge.net* community. We measure our centrality metrics on the largest network of Java projects that can be built from *SourceForge.net*, which includes all relationships among contributors, both direct and indirect. It should be noted that the metrics of centrality have been originally defined to account for both types of relationships (Freeman 1979).

Results also support hypothesis H2, thus confirming centrality as a driver of the ability of a project to attract more effort and, therefore, more contributions. Therefore, a greater ability to share costs within a larger community seems a second important benefit of being a hub node in an OS network. Previous literature has highlighted both a positive and a negative effect of centrality on effort. Centrality has a positive effect as it helps conveying new contributions to the project, while it has a negative effect as it involves a more cumbersome coordination effort that can reduce the team's productivity (Herbsleb and Mockus 2003; Gomes and Joglekar 2004; Monga 2004; Hinds and McGrath

2006). In the "Research Hypotheses" section, we have discussed how the negative effect of centrality primarily impacts on unit effort, i.e. the effort devoted to the development or to the maintenance of a unit of code (e.g. a module). By correcting for unit effort, our research model (Figure 4) provides an estimate of the positive effect of centrality on total effort. Therefore, the significance of the relationship between centrality and total effort in our model can be interpreted as a positive influence of centrality on the ability of a project to attract more contributions. Note that the direct relationship between centrality and unit costs has been found to be non significant. This seems to indicate that centrality *per se* does not increase unit costs. As noted in previous literature, the governance model adopted by the specific OS project can help mitigate the negative effects of a more cumbersome coordination and, therefore, also the negative impact of centrality put forward in previous literature (Goldman and Gabriel 2005).

Findings also support hypothesis H3, thus indicating a negative effect of success on software design quality. We have posited that successful projects have a more chaotic development process. As they strive to preserve their leading position, they are required to focus on a number of collateral activities, such as answering to the forum or mailing list posts, evaluating and integrating new contributions from the community, or providing additional value-added services such as e-learning courses. These activities are performed to the detriment of the time spent in development tasks, which are more likely to be completed with a focus on output, rather than quality. A whole body of literature agrees upon the fact that a low design quality of software negatively influences a number of external quality variables that can be directly perceived by users:

- Reliability, bug-proneness, and testability, which have been proved to negatively impact on user satisfaction, usability, and software effectiveness (Boehm 1976; Brito e Abreu and Melo 1996; Barbey et al. 1998; Subramanyam and Krishnan 2003; Marinescu 2005).
- Source code understandability and, as a consequence, software maintainability, tailorability, and evolvability (Capra et al. 2007);
- Capability of attracting and integrating new team members (Von Krogh et al. 2003).

This suggests that low levels of software quality may end up limiting the success of a project with a negative indirect feedback loop effect. Intuitively, if an application becomes less reliable, usable, and maintainable, the number of users, downloads, and, ultimately, contributors is likely to decrease. In turn, this would negatively influence all the components of success as defined in the "Variables and Data Sample" section. Future work will address the empirical assessment of such negative feedback loop effect in order to understand whether being a hub has both a positive effect in the short term, but also a negative effect on success in the long term. However, we know from previous literature that performing interventions aimed at restoring the quality of source code, such as refactoring interventions, can rejuvenate a project and help further growth. For example, as discussed by (MacCormack 2006), the Netscape Navigator web browser underwent a heavy refactoring intervention before being released as Mozilla. The refactoring intervention had several objectives, but it was mainly aimed at making the source code more attractive to contributors as the project moved from the thousands to the hundreds of thousands of downloads. Similarly, the Eclipse IDE platform has been deeply restructured when the OSGI middleware has been included to provide a more modular support to plug-in development (Gruber et al. 2005). It has been observed that the costs of large refactoring interventions is very high (Advani et al. 2006) and it is preferable to perform small and simple changes, planned early in the project timeline and aimed at preserving good quality levels (Slaughter 1998). Our results indicate that successful OS projects do not seem to follow this good management approach.

Overall, our results show that centrality metrics are significant drivers of project success that should be monitored from the perspective of a project administrator or team manager. However, they also prove that successful projects tend to have a lower design quality of software. This has a number of potential consequences that might be visible to users and could cause negative effects over time. From previous case studies such as Mozilla and Eclipse, it is clear that in order for a social network of a project to become a global success, a refactoring intervention is needed, with a consequent infusion of large investments. The natural behaviour of the social network does not seem to be able to cope with these levels of growth. While it can help a project to start growing and reach a significant level of success, above a certain level it may represent a weak management lever. This is consistent with previous literature positing that excessively large social networks have a lower effectiveness (Herbsleb and Mockus 2003; Gomes and Joglekar 2004; Monga 2004) and represents an interesting subject for future research.

References

Advani, D., Hassoun, Y., and Counsell, S. "Extracting refactoring trends from open-source software and a possible solution to the 'related refactoring' conundrum", *Proceedings of the ACM Symposium on Applied Computing*, 2006.

Ahn, Y., Suh, J., Kim, S., and Kim, H. "A software maintenance project effort estimation model based on function points", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 2, pp. 71-85, 2003.

Albert, R. and Barabàsi, A. L. "Emergence of scaling in random networks", *Science*, vol. 286, no. 5439, pp. 509-512, 1999.

Austin, R. D. "The effects of time pressure on quality in software development: an agency model", *Information Systems Research*, vol. 12, no. 2, pp. 195-207, 2001.

Bagozzi, R. P. and Yi, Y. "On the evaluation of structural equation models", *Journal of the Academy of Marketing Science*, vol. 16, no. 1, pp. 74-94, 1988.

Banker, R., Datar, S., Kemerer, C. "A Model to Evaluate Variables Impacting Productivity on Software Maintenance Projects", *Management Science*, vol. 37, no. 1, pp. 1-18, 1991.

Banker, R., Datar, S., Kemerer, C., and Zweig, D. "Software complexity and maintenance costs", *Communications of the ACM*, vol. 36, no. 11, pp. 81–94, 1993.

Banker, R. D. and Slaughter, S. A. "A study on the effects of software development practices on software maintenance effort", *Proceedings of International Conference on Software Maintenance*, pp. 197-205, 1996.

Banker, R. D., and Slaughter, S. A. "The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement", *Information Systems Research*, vol. 11, no. 3, pp. 219-240, 2000.

Barbey, S., Buchs, D., and Peraire, C. "Modeling the production cell case study using the fusion method", *Technical Report 98/208*, EPFL-DI.

Basili, V. R., Briand, L., Condon, S., Kim, Y. M., Melo, W. L. and Valett, J. D. "Understanding and predicting the process of software maintenance releases", *Proceedings of International Conference on Software Engineering*, pp. 464-474, 1996a.

Basili, V. R., Briand, L. C. and Melo, W. L. "A validation of object-oriented design metrics as quality indicators", *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996b.

Batagelj, V., Mrvar, A. "Pajek - Analysis and Visualization of Large Networks", in Jünger, M., Mutzel, P. (eds.) *Graph Drawing Software*, pp. 77-103, Springer, Berlin, 2003.

Bentler, P. M. "Comparative fit indexes in structural models", *Psychological Bulletin*, vol. 107, no. 2, pp. 238-246, 1990.

Bevan, N. "Usability is Quality of Use", Proceedings of International Conference on Human Computer Interaction, 1995.

Bianchi, A., Caivano, D., Lanubile, F., and Visaggio, G. "Evaluating software degradation through quality", *Proceedings of International Software Metrics Symposium*, pp. 210–219, 2001.

Binkley, A. B. and Scatch, S. R. "Validation of the coupling dependency metric as a predictor of run time failures and maintenance measures", *Proceedings of International Conference of Software Engineering*, pp. 452-455, 1998.

Boehm, B., Brown, J. R., and Lipow, M. "Quantitative evaluation of software quality", *Proceedings of International Conference on Software Engineering*, pp. 592–605, 1976.

Boehm, B. *Software engineering economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

Boehm, B. *Software cost estimation with COCOMO II*, Prentice-Hall, Englewood Cliffs, NJ, 2000.

Boehm, B., Brown, A. W., Madacy, R., and Yang, Y. "A software product line life cycle cost estimation model", *Proceedings of International Symposium on Empirical Software Engineering*, pp. 156–164, 2004.

Bollen, K.A. "A new incremental fit index for general structural equation models", *Sociological Methods and Research*, vol. 17, pp. 303–316, 1989.

Bonacich, P. "Factoring and weighting approaches to status scores and clique identification", *Journal of Mathematical Sociology*. vol. 2, pp. 113-120, 1972.

Borgatti, S. P., "Centrality and AIDS", *Connections*, vol. 18, no. 1, pp. 112-115, 1995.

Borgatti, S. P., Carley, K. M. and Krackhardt, D. "On the robustness of centrality measures under conditions of imperfect data", *Social Networks*, vol. 28, no. 2, pp. 124-136, 2006.

Brin, S. and Page, L. "The anatomy of a large-scale hypertextual Web search engine", *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107-117, 1998.

Brito e Abreu, F. “The MOOD metrics set”, *Proceedings of ECOOP Workshop on Metrics*, 1995.

Brito e Abreu, F., Melo, W. "Evaluating the Impact of Object-Oriented Design on Software Quality," *Proceedings of METRICS*, p. 90, 1996.

Capra, E., Francalanci, C., and Merlo, F. “The economics of open source software: an empirical analysis of maintenance costs”, *Proceedings of International Conference on Software Maintenance*, pp. 395-404, 2007.

Capra, E., Francalanci, C., and Merlo, F. “An empirical study on the relationship between software design quality, development, and governance in Open Source projects”, accepted for publication on the IEEE Transactions on Software Engineering, July 2008.

Carmines, E. G. and McIver, J. P. “Analyzing models with unobserved variables: analysis of covariance structures”, In *Social Measurement: Current Issues*, Bohrnstedt, G. W., and Borgatta, E. F. (eds.), Sage Publications, Beverly Hills, CA, pp. 65-115, 1981.

Chan, T., Chung, S., and Ho, T. “An econometric model to estimate software rewriting and replacement times”, *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 580-598, 1996.

Chapin, N., Hale, J. E., Khan, K. Md., Ramil, J. F., and Tan, W-G. “Types of software evolution and software maintenance”, *Journal of Software Maintenance and Evolution: Research and Practice*, no. 13, pp. 3-30, 2001.

Chen, J. Y. and Lu, J. F. “A new metric for object-oriented design”, *Journal of Information Systems and Software Technology*, vol. 35, no. 4, pp. 232-240, 1993.

Chidamber, S. and Kemerer, C. “A metrics suite for object oriented design”, *IEEE Transactions on Software Engineering*, vol. 20, no.6, pp. 476–493, 1994.

Chidamber, S. R., Darcy, D. P., and Kemerer, C.F. “Managerial use of metrics for object-oriented software: An exploratory analysis”, *IEEE Transactions on Software Engineering*, vol. 24, no.8, pp. 629–639, 1998.

Choi, B., Raghu, T. S. and Vinze, A. “An Empirical Study of Standards Development for E-Businesses: A Social Network Perspective”, *Proceedings Annual Hawaii International Conference on System Sciences*, pp. 139-148, 2006.

Crowston, K., Howison, J. and Annabi, H. “Information systems success in free and open source software development: Theory and measures”, *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 123-148, 2006.

Crowston, K., Annabi, H., and Howison, J. “Defining Open Source Software Project Success”, *Proceedings of International Conference on Information Systems*, Seattle, 2003.

Darcy, D. P., Kemerer, C. F., Slaughter, S. A. and Tomayko, J. E. “The structural complexity of software: an experimental test”, *IEEE Transactions on Software Engineering*, vol. 31, no. 11, pp. 982-995, 2005.

Emerson, T. J. “A discriminant metric for module comprehension”, *Proceedings of International Conference on Software Engineering*, pp. 294-431, 1984.

Fenton, N. and Pfleeger, S. L. *Software metrics: a rigorous and practical approach*, International Thomson Computer Press, London, UK, 1997.

Fornell, C. and Larcker, D. F. “Evaluating structural equation models with unobservable variables and measurement errors: Algebra and statistics”, *Journal of Marketing Research*, vol. 18, no. 3, pp. 383-388, 1981.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. *Refactoring: improving the design of existing code*, Addison Wesley, 2001.

Freeman, L. C. “Centrality in Social Networks: Conceptual clarification”, *Social Networks*, vol. 1, no. 3, pp. 215–239, 1979.

Gao, Y. and Madey, G. “Network analysis of the SourceForge.net community”. In *Open Source Development, Adoption and Innovation*, Springer Boston, 2007.

Gao, Y., Madey, G. and Freeh, V. “Modeling and Simulation of the Open Source Software Community”, *Agent-Directed Simulation Symposium*, 2005.

Gao, Y., Van Antwerp, M., Christley, S., and Madey, G. "A Research Collaboratory for Open Source Software Research", *Proceedings of International Workshop on Emerging Trends in FLOSS Research and Development*, 2007.

Goldman, R. and Gabriel, R. P. *Innovation happens elsewhere: Open Source as business strategy*, Morgan Kauffmann, 2005.

Gomes, P., and Joglekar, N. “The costs of coordinating distributed software development tasks”, *Technical report*, Boston University School of Management, Boston, 2004.

Grewal, R., Lilien, G. L., and Mallapragada, G. "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems", *Management Science*, vol. 52, no. 7, pp. 1043-1056, 2006.

Gruber, O., Hargrave, B. J., McAffer, J., Rapicault, P. and Watson, T. “The Eclipse 3.0 platform: adopting OSGi technology”, *IBM Systems Journal*, vol. 44, no. 2, pp. 289-299, 2005.

Halstead, M. H. *Elements of software science*, Elsevier Computer Science Library, 1977.
Harrison, R., Counsell, S., and Nithi, R. "An evaluation of the MOOD set of object-oriented software metrics", *IEEE Transactions on Software Engineering*, vol. 24, no. 6, pp. 491-496, 1998.
Harter, D. E., Krishnan, M. S., and Slaughter, S. A. "Effects of process maturity on quality, cycle time and effort in software product development", *Management Science*, vol. 46, no. 4, pp. 451-466, 2000.
Henry, S. and Kafura, D. "Software structure metrics based on information flow", *IEEE Transactions on Software Engineering*, vol. 7, no. 5, pp. 510–518, 1981.
Herbsleb, J.D. and Mockus, A., "An empirical study of speed and communication in globally distributed software development", *IEEE Transactions on Software Engineering*, vol.29, no.6, pp. 481-494, 2003.
Hinds, P. and McGrath, C. "Structures that work: social structure, work structure and coordination ease in geographically distributed teams", In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, NY, pp. 343-352.
Hitz, M. and Montazeri, B. "Chidamber and Kemerer's metrics suite: A measurement theory perspective", *IEEE Transactions on Software Engineering*, vol. 22, no. 4, pp. 267–271, 1996.
Hossain, L., Wu A. and Chung, K. K. S. "Actor Centrality Correlates to Project Based Coordination", *Proceedings of the Computer Supported Cooperative Work Conference*, pp. 363-372, 2006.
Howison, J. and Crowston, K. "The perils and pitfalls of mining SourceForge", *Proceedings of International Workshop on Mining Software Repositories*, pp. 7–12, 2004.
Howison, J. and Crowston, K. "The social structure of free and open source software development", *First Monday*, vol. 10, no. 2, 2005.
Hubbell, C. H. "An Input-Output Approach to Clique Identification", *Sociometry*, vol. 28, no. 4, pp. 377-399, 1965.
International Organization for Standardization (ISO/IEC), *ISO Standard Series 9126-1*, "Software Engineering - Product Quality - Part 1: Quality Model", Geneva, 2004.
Jamali, M. and Abolhassani, H. "Different Aspects of Social Network Analysis", IEEE/WIC/ACM International Conference on Web Intelligence 2006, pp. 66-72, 2006.
Katz, L. "A new status index derived from sociometric analysis", *Psychometrika*, vol. 18, no. 1, pp. 39-43, 1953.
Kemerer, C. F. "An empirical validation of software cost estimation models", *Communications of the ACM*, vol. 30, no. 5, pp. 416–430, 1987.
Kline, R. B. *Principles and practice of structural equation modeling*, Second Edition, Guilford Press, NY, 2004.
Koch, S. "Agile principles and Open Source software development: a theoretical and empirical discussion", *Extreme Programming and Agile Processes in Software Engineering*, Springer, Berlin, pp. 85-93, 2004.
Krishnamurthy, S. "A managerial overview of Open Source Software", *Business Horizons*, vol. 46, no. 5, pp. 47-56, 2003.
Lanza, M. and Marinescu, R. Object-Oriented metrics in practice - Using software metrics to characterize, evaluate, and improve the design of Object-Oriented systems, Springer, 2006.
Li, W. and Henry, S. "Maintenance metrics for the object oriented paradigm", *Proceedings of IEEE International Software Metrics Symposium*, 1993.
Lientz, B. and Swanson, B. *Software maintenance management*, Addison-Wesley, 1981.
MacCormack, A., Rusnak, J., and Baldwin, C. Y. "Exploring the structure of complex software designs: an empirical study of Open Source and proprietary code", *Management Science*, no. 52, pp. 1015-1030, July 2006.
Marcoulides, G. A. and Saunders C. "PLS: A silver bullet?", *MIS Quarterly*, vol. 30, no. 2, pp. iii-ix, June 2006.
Marinescu, R. "Measurement and Quality in Object-Oriented Design", *Proceedings of International Conference on Software Maintenance*, 2005.
McCabe, T. J. "A complexity measure", Proceedings of International Conference on Software Engineering, p. 407, 1976.
Mehra, A., Kilduff. M. and J. Brass, D. J. "The Social Networks of High and Low Self-Monitors: Implications for Workplace Performance", *Administrative Science Quarterly*, vol. 46, no. 1, pp. 121-146, 2001.
Mens, T., and Tourwé, T. "A survey of software refactoring", *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126-139, 2004.
Mockus, A. and Graves, T. L. "Identifying productivity drivers by modeling work units using partial data", *Technometrics*, vol. 42, no. 2, pp. 168-179, 2001.
Monga, M. "From Bazaar to Kibbutz: how freedom deals with coherence in the Debian project", *Proceedings of Workshop on Open Source Software Engineering*, pp. 71-75, 2004.
Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K. and Ye, Y. "Evolution patterns of open-source software systems and communities", *Proceedings of the International Workshop on Principles of Software Evolution*, pp. 76-85, 2002.

Newman, M. E. J. “A measure of betweenness centrality based on random walks”, *Social Networks*, vol. 27, no. 1, pp. 39-54, 2005.

Newman, M., Barabàsi, A. L. and Watts, D. J. *The Structure and Dynamics of Networks*, Princeton University Press, 2006.

Ohira, M., Ohoka, T., Kakimoto, T., Ohsugi, N., and Matsumoto, K. “Supporting knowledge collaboration using social networks in a large-scale online community of software development projects”, *Proceedings of Asia-Pacific Software Engineering Conference*, pp. 835-840, 2005.

Oviedo, E. I. “Control flow, data flow and programmers complexity”, *Proceedings COMPSAC80*, pp. 146–152, 1980.

Pawlak, R. “Spoon: Annotation-driven program transformation - the AOP case”, *Proceedings of Workshop on Aspect-Orientation for Middleware Development*, 2005.

Ramil, J. F. “Continual resource estimation for evolving software”, *Proceedings of International Conference on Software Maintenance*, 2003.

Reagans, R., and McEvily, B. "Network Structure and Knowledge Transfer: The Effects of Cohesion and Range" *Administrative Science Quarterly*, vol. 48, no. 2, pp. 240-267, 2003.

Rosenberg, L. H. *Applying and interpreting object oriented metrics*, Technical Report, Software Assurance Technology Center NASA SATC, April 1998.

Sharble, R. C. and Cohen, S. S. “The Object-Oriented brewery: a comparison of two Object-Oriented development methods”, *ACM SIGSOFT Software Engineering Notes*, vol. 18, no. 2, pp. 60-73, 1993.

Slaughter, S. A., Harter, D. E. and Krishnan, M. S. “Evaluating the cost of software quality”, *Communications of the ACM*, vol. 41, no. 8, pp. 67-73, August 1998.

Sparrowe, R. T., Liden, R. C., Wayne, S. J., and Kraimer, M. L., “Social Networks and the Performance of Individuals and Groups”, *The Academy of Management Journal*, vol. 44, no. 2., pp. 316-325, 2001.

Stephenson, K. A. and Zelen, M., “Rethinking centrality: Methods and examples”, *Social Networks*, vol. 11, 1–37, 1989.

Subramanyam, R. and Krishnan, M. S. “Empirical analysis of CK metrics for Object Oriented design complexity: implications for software defects”, *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297-310, 2003.

Symons, C. R. “Function Point analysis: difficulties and improvements”, *IEEE Transactions on Software Engineering*, vol. 14, no. 1, pp. 2-11, 1988.

Tan, Y., and Mookerjee, V. “Comparing uniform and flexible policies for software maintenance and replacement”, *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 238–256, 2005.

Troy, D. and Zweben, S. *Measuring the quality of structured design*, McGraw-Hill International Series in Software Engineering, pp. 214-226, 1993.

Tsai, W. “Knowledge Transfer in Intraorganizational Networks: Effects of Network Position and Absorptive Capacity on Business Unit Innovation and Performance”, *The Academy of Management Journal*, vol. 44, no. 5, pp. 996-1004, 2001.

Tucker, L. R. and Lewis, C. “A reliability coefficient for maximum likelihood factor analysis”, *Psychometrika*, vol. 38, no. 1, pp. 1–10, 1973.

Von Krogh, G., Spaeth, S., and Lakhani, K. R. “Community, joining and specialization in open source software innovation: a case study”, *Research Policy*, vol. 32, no. 7, pp. 1217-1241, 2003.

Wasserman, S. and Faust, K. “Social Network Analysis: Methods and Applications”, Cambridge University Press, Cambridge, 1994.

Xu, J., Christley, S., and Madey, G., “Application of Social Network Analysis to the Study of Open Source Software” in *The Economics of Open Source Software Development*, Bitzer, J. and Schröder, P. J. H. (eds.), Elsevier B.V., 2006.

Ye, Y., Nakakoji, K., Yamamoto, Y. and Kishida, K. “The Co-Evolution of Systems and Communities in Free and Open Source Software Development”, in *Free/Open Source Software Development*, Koch, S. (ed.), Idea Group, 2004.