Association for Information Systems
AIS Electronic Library (AISeL)

ECIS 2009 Proceedings

European Conference on Information Systems (ECIS)

2009

Panel: A call for action in tackling environmental sustainability through green information technologies and systems

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

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

THE RELATIONSHIP AMONG DEVELOPMENT SKILLS, DESIGN QUALITY, AND CENTRALITY IN OPEN SOURCE PROJECTS

Barbagallo, Donato, Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34/5, 20133 Milano, Italy, barbagallo@elet.polimi.it

Francalanci, Chiara, Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34/5, 20133 Milano, Italy, francala@elet.polimi.it

Abstract

In a previous paper, we have found empirical evidence supporting a positive relationship between network centrality and success. However, we have also found that more successful projects have a lower technical quality. A first, straightforward argument explaining previous findings is that more central contributors are also highly skilled developers who are well known for their ability to manage the complexity of code with a lower attention to the software structure. The consolidated metrics of software quality used by the authors in their previous research represent measures of code structure. This paper provides empirical evidence supporting the idea that the negative impact of success on quality is caused by the careless behaviour of skilled developers, who are also hubs within the social network. Research hypotheses are tested on a sample of 56 OS applications from the SourceForge.net repository, with a total of 378 developers. The sample includes some of the most successful and large OS projects, as well as a cross-section of less famous active projects evenly distributed among SourceForge.net's project categories.

Keywords: social networks, software quality, software design skills.

1 INTRODUCTION

Social networks represent social systems characterized by very large numbers of individuals and relationships among individuals. By empirically analyzing the evolution of OS networks modeled as sets of cooperation relationships among project contributors, previous research has tested the following principles:

• 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).

In the OS context, 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 Section 2). Grewal et al. (2006) have tested whether higher values of centrality metrics are positively correlated with the ranking measure of project success[1]. By testing correlation for 12 projects from *SourceForge.net* written in Perl, 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).

In a previous paper (Barbagallo et al. 2008), we have tested Grewal et al.'s relationship on a significantly larger sample and actually found empirical evidence supporting the relationship between centrality and success. 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. However, in our previous research, we have also found that more successful projects have a lower technical quality. This paper aims at understanding the reasons behind the negative relationship between success and quality. Is success in and of itself detrimental to quality or are there drivers of quality other than success that help explain quality degradation? If so, are these drivers related to centrality?

Answering these questions can help managers understand the implications of gaining a more central position in the network as a way to reach success. The software engineering literature explains that there exists a trade off between quality and costs. Since quality represents an investment in the long term, managers are often willing to accept a lower level of quality if it represents a way to have a marketable output more quickly (Tan and Mookerjee 2005). Explaining the relationship between success and quality can help managers understand whether an OS social network represents a governance structure that allows deliberate quality to cost decisions or it creates new and negative quality degradation effects.

The presentation is organized as follows. Section 2 reviews the literature on social networking centrality metrics and software design quality metrics. Section 3 discusses our research hypotheses, while Section 4 describes the operationalization of variables, the data sample used to verify our hypotheses, our statistical

[1] A definition of ranking in *SourceForge.com* is given in http://apps.sourceforge.net/trac/sourceforge/wiki/Project%20statistics

approach and reports the results of empirical testing. Finally, Section 5 provides a discussion of empirical findings and outlines possible directions for future research.

2 RELATED WORK

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

2.1 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.

The first metric of centrality, called *degree centrality*, discussed by Freeman (1979), 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 (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.

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 (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 (1995) has noted that eigenvector centrality is conceptually similar to degree centrality. However, it should be acknowledged that the *PageRank* metric proposed by Brin and Page (1998) is based on the notion of eigenvector centrality. Even earlier than Bonacich, Katz (1953) and 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.

2.2 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, Marinescu 2005). In turn, these external quality variables affect user satisfaction and can influence software adoption and actual usage (Bevan 1995). 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 (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 (Harrison et al. 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).

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 (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. (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 (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 (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. (2008) has empirically verified that quality and development effort are not correlated in OS projects, supporting the theoretical observations of Tan and Mookerjee (2005). In this paper, we consider software design quality and development effort as independent variables.

3 HYPOTESES

In a previous paper (Barbagallo et al. 2008), we have found empirical evidence supporting the following hypoteses:

- Projects involving contributors with a higher level of centrality are more successful.
- Projects involving contributors with a higher level of centrality are able to attract a greater development and maintenance effort.
- More successful projects have a lower software design quality.

Overall, these hypotheses indicate that involving contributors that play a more central role in the social network helps projects to attract investments and eventually reach success. However, it does not seem to help the quality of the software artifact. These findings are counterintuitive, since they show that more successful projects (in terms of *SourceForge.com* ranking) have lower technical quality. In the paper, we discuss the possible reasons behind this inverse relationship between success and quality. A first, straightforward argument explaining previous findings is that more central contributors are also highly skilled developers who are well known for their ability to manage the complexity of code with a lower attention to the software structure (Lakhani and Wolf 2003). The consolidated metrics of software quality used by the authors in their previous research represent measures of code structure. Structure is generally considered a proxy of quality, as it represents the main driver of software maintenance costs (Li and Henry 1996).

This paper aims at providing empirical evidence supporting the idea that the negative impact of success on quality is caused by the careless behaviour of skilled developers, who are also hubs within the social network. Our first research hypothesis addresses the relationship between centrality and developers' skills. In contexts different from OSS, 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 OSS 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 improve the communication with other network members, increase the number of cooperation opportunities within a variety of projects, help gain new skills, and ultimately develop a broader expertise. (Kidane and Gloor 2005) have found a correlation between group density and its performance and creativity in the case of Eclipse community. These benefits create a virtuous circle, contributing to the growth of developers as hubs of the network. This leads us to our first research hypothesis:

H1: *More skilled contributors are involved in projects with a higher level of centrality.*

Our second research hypothesis addresses the relationship between skills and quality. As noted before, skilled developers have the ability to solve problems quickly and efficiently, but are less willing to design code according to the software engineering principles ruling structured development (Lakhani and Wolf 2003). It has been demonstrated that organizations tend to apply their most skilled developers in complex tasks, where devising an efficient algorithm is more important than obtaining a well-structured and maintainable software (Faraj and Sproull 2000). These results are interesting in a OS context, since we know from Capra et al (2008) that the governance model of an OS project can resemble that of a closed source project, with a single sponsor company developing the bulk of code. Furthermore, hub nodes 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).

The rate of new change requests (new features, bug reports, etc.) grows accordingly. Hypotesis H1 posits that more skilled developers are involved in more central projects and, therefore, can benefit from all the advantages of their central position in terms of knowledge transfer. However, they are also burdened by a greater information load related to communicating and coordinating with other contributors in development activities and also in a number of non-development activities, including debugging, translation, advisory, and documentation. 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. Therefore, more central developers can be supposed to spend more time in coordination and less time in quality, which, in the short term, represents a time investment that they may not be able to afford given their information overhead. This leads to our second research hypothesis:

H2: *Projects involving more skilled contributors have a lower software design quality.*

4 METHODOLOGY AND RESULTS

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

4.1 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.

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 (Hossain et al 2006).

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 (1994) metrics' suite for the evaluation of quality at the class level. The four chosen metrics (MIF, NOC,

CBO, DIT) are the most preferable to provide measures of *inheritance* and *coupling*, which are two of the three sets of fundamental quality metrics, and also the most studied in literature (Capra et al. 2008).

Metrics of skills. Two metrics deal with skills in our model, *skill level* and *skill range*. The *skill level* metric SL_k of project k is defined as the mean value of the average per-skill level of the developers involved in the project:

$$SL_k = \sum_{i \in k} \frac{\sum_{s \in S(k)} \frac{SK(i,s)}{MAX\_VAL * ns(i)}}{nd(k)},$$

where *SK(i,s)* is the level of skill *s* of developer *i*, *S(k)* is developer *i*'s set of skills, *MAX_VAL* is a constant representing the maximum skill level (5 in *SourceForge.net*), *ns(i)* is the total number of skills of developer *i*, and *nd(k)* is the total number of developers working at project *k*.

The *skill range* metric SR_k of project *k* is defined as the ratio between the number of distinct skills owned by at least one developer in the project and the total number of skills in the project:

$SR_k = NDS_k / NS_k$,

where NDS_k and NS_k are the total number of distinct skills and the total number of skills in project *k*, respectively. The *skill range* measures the diversification of skills within a project. For example, if a project has two developers with the same skill set the metric will evaluate to 0.5, while if the skill sets of the two developers are disjoint the metric will be equal to 1.

4.2 Data Sample

The data set used for this study has been gathered by analyzing a sample of OS community applications from the *SourceForge.net* repository. Data on skills have been extracted by analyzing each developer's profile on *SourceForge.net*. 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:

- 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). As suggested by Marcoulides and Sounders (2006), confidence intervals at level $\alpha = 0.05$ have been computed for the variables involved in hypotheses testing in order to assess the adequacy of our sample size. Although the sample is not extremely wide, the confidence intervals are relatively narrow for all the considered 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 1 presents cardinalities of application samples AS1 and AS2.

Variable	Dataset AS1	Dataset AS2
Number of projects	56	29,836
Number of developers	378	57,142

Table 1. Cardinality of datasets AS1 and AS2

Social networking metrics have been derived from the analysis of the social network of the applications included in sample AS2. Social network data has been derived from the *SourceForge.net* data warehouse

(Gao et al. 2007) for all the applications in sample AS2, and processed by a tool developed ah-hoc. The computation of social networking metrics has been performed by analyzing the *SourceForge.net* social network with Pajek (Batagelj and Mrvar 1997), one of the most used and referenced tools for large networks analysis.

Software quality metrics have been evaluated by analyzing the source code of all available versions of each project in our dataset AS1, and considering average values for each metric. Source code has been analyzed with a tool developed ad-hoc. The tool provides data on all the software quality metrics described in Section 4.1, performing static analyses of 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.

4.3 Measurement Model

The measurement model has been defined to verify the assumption that social networking and software design quality metrics actually measure different aspects of the same phenomena. A principal component analysis (PCA) has been performed on both sets of metrics to verify the assumption. Results of such analysis are shown in Table 2.

Figure 1. Measurement model of design quality

Figure 1 presents the measurement model related to the design quality set of metrics. In this case only one factor has been extracted, and has been labelled as *Design Quality*.

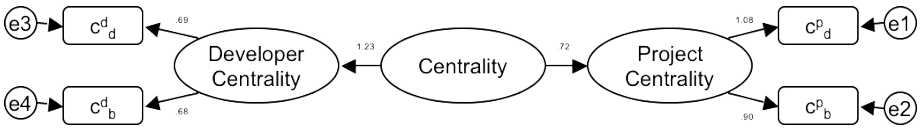

Figure 2. Measurement model of centrality

Figure 2 shows the measurement model related to the social networking set of metrics. As it can be noted, two different factorization variables have been identified, related to the centrality of developers and of projects, respectively. In turn, these two latent variables have been found to be different aspects of the same concept, that has been named *Centrality*.

Dependent Variable	Independent Variable	Standardized Regression Weight (b)	Standard Error	p-value	Composite Factor Reliability
C^d_d	Developer Centrality	0.688	1.311	<0.001	0.830
C^d_b	Developer Centrality	0.680	-	-	0.684
C^p_d	Project Centrality	1.079	0.049	<0.001	0.970
C^p_b	Project Centrality	0.896	-	<0.001	0.915
Developer Centrality	Centrality	1.232	-	-	-
Project Centrality	Centrality	0.722	1.649	<0.001	-
NOC	Design Quality	-0.983	-	-	0.911
CBO	Design Quality	-0.697	0.525	<0.001	0.740
DIT	Design Quality	-0.973	0.074	<0.001	0.900
MIF	Design Quality	-0.748	0.099	<0.001	0.894

Table 2. *PCA results and estimates of regression weights for the measurement models of Figure 1 and Figure 2*

Table 2 shows the results of the PCA performed on the different parts of the measurement model, along with the standardized regression weights of the relationships between latent and observed variables. Results show that all the factorizations should be considered acceptable, since all the composite factor reliability values are greater or very close to the threshold value of 0.700 as suggested by Bagozzi and Yi (1988). All the relationships considered between the set of observed variables (either referred to centrality or software design entropy) and the latent variables are significant with $p < 0.001$: this confirms that the factorizations in the measurement model were performed correctly.

4.4 Structural Model Testing

The estimation results of the research model in Figure 3 used to test the research hypotheses are shown in Table 3. Please note that, for sake of simplicity, the model in Figure 3 is a simplified version in which the factorizations related to the latent variables *Design Quality* and *Centrality* discussed in the previous section are not shown. Variables *Design Quality*, and *Centrality* have been controlled by project age *Age*, as suggested by Banker and Slaughter (2000), although the controlling variable *Age* is not shown to reduce the complexity of the model in Figure 3.

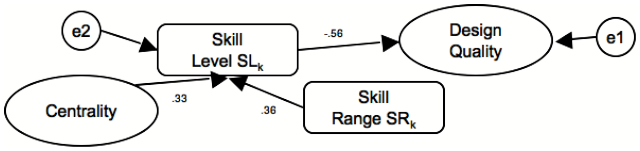

Figure 3. *Structural model for the verification of research hypoteses*

Dependent Variable	Independent Variable	Standardized Regression Weight (b)	Standard Error	p-value
Centrality	Age	0.526	0.000	<0.001
Skill Level	Centrality	0.329	737.241	0.045
Skill Level	Skill Range	0.356	0.260	0.050
Design Quality	Skill Level	-0.556	0.084	<0.001

Table 3. *Estimates of regression weights for the research model of Figure 3*

All the relationships hypothesized between model variables are significant with $p < 0.05$, that is, can be accepted at a significance level $\alpha = 95\%$. Table 4 shows that the overall model fit is satisfactory.

Index	Research Model	Desired Level	Reference
χ^2/d.f.	1.727	<3.0	(Carmines and McIver 1981)
p-value	0.003	<0.05	-
IFI	0.944	>0.90	(Bollen 1989)
TLI	0.903	>0.90	(Tucker and Lewis 1973)
CFI	0.941	>0.90	(Bentler 1990)

Table 4. *Goodness of fit indices for the research model in Figure 3.*

Research hypothesis H1 (more skilled contributors are involved in projects with a higher level of centrality) is represented within our model by the regression relationship between *Centrality* (independent variable) and *Skill Level* (dependent variable). Our analysis shows that the regression weight is positive and statistically significant ($b = 0.329$, $p = 0.045$). Consequently, research hypothesis H1 is verified.

Hypothesis H2 (projects involving more skilled contributors have a lower software design quality) is tested by the relationship between *Design Quality* and *Skill Level*, where the former is the dependent variable. Since the regression weight is negative (b = -0.556) and the estimation of this relation is significant (p = 0.050), hypothesis H2 is verified. Please note that *Skill Level* is controlled by *Skill Range* in order to consider not only the depth of skills but also the heterogeneity.

5 DISCUSSION AND CONCLUSIONS

Empirical results support hypothesis H1, thus confirming centrality as a significant driver of the expertise of developers. Therefore, knowledge seems to represent a first, fundamental benefit of being a hub node in an OS network. Previous to our work, the literature has described the role of social networking as enabler of knowledge transfer processes and related effectiveness. As noted in Section 1, our work is the first empirical contribution providing empirical evidence to support the relationship between network centrality and expertise. In particular, we have made an effort to overcome some of the limitations of previous research by measuring our centrality metrics on the largest network of Java projects that can be built from *SourceForge.net*, which includes all the 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, suggesting that more skilled developers will tend to produce a software with a lower design quality. Even if this seems counterintuitive, it should be noticed that hubs in OS communities are involved in several projects and incur a coordination overhead that reduces the share of time that they can devote to quality. The literature indicates that quality represents an investment that requires time and provides its pay-offs over time in terms of lower maintenance costs (Slaughter et al. 1998). The coordination overhead caused by a more central role may reduce the willingness of developers to invest in quality.

It should be noticed that in our model the relation between success and design quality remains significant, thus indicating that development skills are only one of the factors impacting on quality. Future research will investigate the possible influence of other factors that, overall, may have a mediation effect on success.

Our results show that centrality metrics are significant proxies of developers' skills that should be monitored from the perspective of a project administrator or team manager. However, they also prove that projects with more skilled team members 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) and represents an interesting subject for future research.

References

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

Bagozzi, R. P. and Yi, Y. (1988). "On the evaluation of structural equation models", Journal of the Academy of Marketing Science, 16 (1), 74-94.

Banker, R. D., and Slaughter, S. A. (2000). The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement. Information Systems Research, 11 (3), 219-240.

Barbagallo, D., Francalanci, C., and Merlo, F. (2008). The impact of Social Networking on Software Design Quality and Development Effort in Open Source Projects. Proc. of Intl. Conf. on Information Systems.

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

Bentler, P. M. (1990). Comparative fit indexes in structural models, Psychol. Bulletin, 107 (2), 238-246.

Bevan, N. (1995). Usability is Quality of Use. Proc. of Intl. Conf. on Human Computer Interaction.

Boehm, B., Brown, J. R., and Lipow, M. (1976). Quantitative evaluation of software quality, Proc. of Intl. Conf. on Software Engineering, 592–605.
Boehm, B., Brown, A. W., Madacy, R., and Yang, Y. (2004). A software product line life cycle cost estimation model, Proc. of Intl. Symposium on Empirical Software Engineering, 156–164.
Bollen, K.A. (1989). A new incremental fit index for general structural equation models, Sociological Methods and Research, 17, 303–316.
Bonacich, P. (1972). Factoring and weighting approaches to status scores and clique identification, Journal of Mathematical Sociology. 2, 113-120.
Borgatti, S. P. (1995). Centrality and AIDS, Connections, 18 (1), 112-115.
Borgatti, S. P., Carley, K. M. and Krackhardt, D. (2006). On the robustness of centrality measures under conditions of imperfect data, Social Networks, 28 (2), 124-136.
Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine, Comput Networks ISDN, 30(1-7), 107-117.
Brito e Abreu, F. (1995). The MOOD metrics set, Proc. of ECOOP Workshop on Metrics.
Brito e Abreu, F., Melo, W. (1996). Evaluating the Impact of Object-Oriented Design on Software Quality. Proc. of METRICS, 90.
Capra, E., Francalanci, C., and Merlo, F. (2007). The economics of open source software: an empirical analysis of maintenance costs. Proc. of Intl. Conf. on Software Maintenance, 395-404.
Capra, E., Francalanci, C., and Merlo, F. (2008). An empirical study on the relationship between software design quality, development, and governance in Open Source projects. IEEE Trans. on Soft. Eng., 34 (6), 765-782.
Carmines, E. G. and McIver, J. P. (1981). 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, 65-115.
Chen, J. Y. and Lu, J. F. (1993). A new metric for object-oriented design. J Inform Systems Soft Tech, 35 (4), 232-240.
Chidamber, S. and Kemerer, C. (1994). A metrics suite for object oriented design. IEEE Trans on Soft Eng, 20 (6), 476–493.
Choi, B., Raghu, T. S. and Vinze, A. (2006). An Empirical Study of Standards Development for E-Businesses: A Social Network Perspective. Proc. Ann. Hawaii Intl. Conf. on System Sciences, 139-148.
Emerson, T. J. (1984). A discriminant metric for module comprehension. Proc. Int. Conf. Soft Eng, 294-431.
Faraj, S. and Sproull, L. (2000). Coordinating Expertise in Software Development Teams. Management Science, 46 (12), 1554-1568.
Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (2001). Refactoring: improving the design of existing code, Addison Wesley.
Freeman, L. C. (1979). Centrality in Social Networks: conceptual clarification. Soc Networks, 1(3), 215-239.
Gao, Y. and Madey, G. (2007). Network analysis of the SourceForge.net community. In Open Source Development, Adoption and Innovation, Springer Boston.
Gao, Y., Madey, G. and Freeh, V. (2005). Modeling and Simulation of the Open Source Software Community. Agent-Directed Simulation Symp.
Gao, Y., Van Antwerp, M., Christley, S., and Madey, G. (2007). A Research Collaboratory for Open Source Software Research. Proc. of Intl. Workshop on Emerging Trends in FLOSS R&D.
Grewal, R., Lilien, G. L., and Mallapragada, G. (2006). Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems. Manage Sci, 52 (7), 1043-1056.
Halstead, M. H. (1977). Elements of software science, Elsevier Computer Science Library.
Harrison, R., Counsell, S., and Nithi, R. (1998). An evaluation of the MOOD set of object-oriented software metrics. IEEE Trans on Soft Eng, 24 (6), 491-496.
Henry, S. and Kafura, D. (1981). Software structure metrics based on information flow. IEEE Trans on Soft Eng, 7 (5), 510–518.
Herbsleb, J.D. and Mockus, A., (2003). An empirical study of speed and communication in globally distributed software development. IEEE Transactions on Software Engineering, 29 (6), 481-494.
Hinds, P. and McGrath, C. (2006). Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In Proc. of Conf. on Comp Supp Coop Work, NY, 343-352.
Hossain, L., Wu A. and Chung, K. K. S. (2006). Actor Centrality Correlates to Project Based Coordination. Proc. of the Conf. Comp Supp Coop Work Conference, 363-372.

Howison, J. and Crowston, K. (2004). The perils and pitfalls of mining SourceForge. Proc. of Intl. Workshop on Mining Software Repositories, 7–12.
Hubbell, C. H. (1965). An Input-Output Approach to Clique Identification. Sociometry, 28 (4), 377-399.
Jamali, M. and Abolhassani, H. (2006). Different Aspects of Social Network Analysis, IEEE/WIC/ACM Intl. Conf. on Web Intelligence, 66-72.
Katz, L. (1953). A new status index derived from sociometric analysis, Psychometrika, 18 (1), 39-43.
Kidane, Y. Gloor, P. (2005). Correlating Temporal Communication Patterns of the Eclipse Open Source Community with Performance and Creativity, NAACSOS Conference.
Kline, R. B. (2004). Principles and practice of structural equation modeling, Second Ed., Guilford Press, NY.
Koch, S. (2004). Agile principles and Open Source software development: a theoretical and empirical discussion. Extreme Programming and Agile Processes in Software Engineering, Springer, Berlin, 85-93.
Lakhani K. R. and Wolf R. G. (2003). Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects, Working Paper 4425-03, MIT Sloan School of Management.
Lanza, M. and Marinescu, R. (2006). Object-Oriented metrics in practice - Using software metrics to characterize, evaluate, and improve the design of Object-Oriented systems, Springer.
Li, W. and Henry, S. (1993). Maintenance metrics for the object oriented paradigm. Proc. of IEEE Intl. Software Metrics Symp.
Marcoulides, G. A. and Saunders C. (2006). PLS: A silver bullet?. MIS Quart., 30 (2), 3-4.
Marinescu, R. (2005). Measurement and Quality in Object-Oriented Design. Proc. Intl. Conf. on Soft Maint.
McCabe, T. J. (1976). A complexity measure. Proc. of Intl. Conf. on Software Engineering, 407.
Mehra, A., Kilduff. M. and J. Brass, D. J. (2001). The Social Networks of High and Low Self-Monitors: Implications for Workplace Performance. Admin. Science Quarterly, 46 (1), 121-146.
Newman, M. E. J. (2005). A measure of betweenness centrality based on random walks. Soc Networks, 27 (1), 39-54.
Newman, M., Barabàsi, A. L. and Watts, D. J. (2006). The Structure and Dynamics of Networks, Princeton University Press.
Ohira, M., Ohoka, T., Kakimoto, T., Ohsugi, N., and Matsumoto, K. (2005). Supporting knowledge collaboration using social networks in a large-scale online community of software development projects. Proc. of Asia-Pacific Software Engineering Conf., 835-840.
Reagans, R., and McEvily, B. (2003). Network Structure and Knowledge Transfer: The Effects of Cohesion and Range. Administrative Science Quarterly, 48 (2), 240-267.
Sharble, R. C. and Cohen, S. S. (1993). The Object-Oriented brewery: a comparison of two Object-Oriented development methods. ACM SIGSOFT Software Engineering Notes, 18 (2), 60-73.
Slaughter, S. A., Harter, D. E. and Krishnan, M. S. (1998). Evaluating the cost of software quality. Comm of the ACM, 41 (8), 67-73.
Sparrowe, R. T., Liden, R. C., Wayne, S. J., and Kraimer, M. L. (2001). Social Networks and the Performance of Individuals and Groups. Acad of Management Journal, 44 (2), 316-325.
Stephenson, K. A. and Zelen, M. (1989). Rethinking centrality: methods and examples. Soc Networks, 11, 1-37.
Symons, C. R. (1988). Function Point analysis: difficulties and improvements, IEEE Trans on Soft Eng, 14 (1), 2-11.
Tan, Y., and Mookerjee, V. (2005). Comparing uniform and flexible policies for software maintenance and replacement. IEEE Trans on Soft Eng, 31 (3), 238–256.
Troy, D. and Zweben, S. (1993). Measuring the quality of structured design, McGraw-Hill International Series in Software Engineering, 214-226.
Tsai, W. (2001). Knowledge Transfer in Intraorganizational Networks: Effects of Network Position and Absorptive Capacity on Business Unit Innovation and Performance. Acad Manage J, 44 (5), 996-1004.
Tucker, L. R. and Lewis, C. (1973). A reliability coefficient for maximum likelihood factor analysis, Psychometrika, 38 (1), 1–10.
Wasserman, S. and Faust, K. (1994). Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge.
Xu, J., Christley, S., and Madey, G. (2006). Application of Social Network Analysis to the Study of Open Source Software. In The Economics of Open Source Software Development, Elsevier B.V.
Ye, Y., Nakakoji, K., Yamamoto, Y. and Kishida, K. (2004). The Co-Evolution of Systems and Communities in Free and Open Source Software Development. In Free/Open Source Software Development, Idea Group.