

**THE CO-EVOLUTION OF ORGANIZATIONAL STRUCTURES AND SOFTWARE
STRUCTURES IN OPEN SOURCE AND CLOSED SOURCE PROJECTS**

FRANCESCO MERLO
Dipartimento di Elettronica e Informazione, Politecnico Di Milano
Via Ponzio 34/5, 20133 Milano, Italy
merlo@elet.polimi.it

SANDRA A. SLAUGHTER**
College of Management,
Georgia Institute of Technology,
Atlanta, Georgia, U.S.A.
Sandra.slaughter@mgt.gatech.edu

CHIARA FRANCALANCI
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
francalanci@elet.polimi.it

Abstract

This study draws on the product development literature to hypothesize how organizational structures and product design structures co-evolve in open and closed source software projects. We further hypothesize how these structures affect project performance. Findings suggest that organizational structures and software structures become mutually interdependent but also reveal significant differences in the patterns of relationships in Open Source and Closed Source projects. In Open Source, the software structure reflects the organizational structure of its development team whereas in Closed Source the software structure also impacts the organizational structure of the team. Further, the organizational structure impacts productivity in Closed Source projects, while the software structure impacts productivity in Open Source

projects. Intriguingly, over time the organizational structure in Closed Source projects evolves to resemble that of Open Source. Our results have important implications for the management and coordination of product development projects.

Keywords: Software Design Structure, Organizational Structure, Social Network, Open Source, Closed Source, Software Evolution; Product Development; Product Design; Organizational Design.

*** contact author for this paper. Please do not cite, quote or distribute without permission of the authors. An earlier version of this paper was presented at the Academy of Management meeting in Chicago, Illinois, and an extended abstract published in the Best Paper Proceedings.*

© Merlo, Slaughter, Francalanci, 2009.

THE CO-EVOLUTION OF ORGANIZATIONAL STRUCTURES AND SOFTWARE STRUCTURES IN OPEN SOURCE AND CLOSED SOURCE PROJECTS

Researchers in innovation and product development are keenly interested in the possibility of a relationship between the design of an organization and the design of the products it creates (e.g., Sosa, 2004; MacCormack et al. 2006; Henderson and Clark, 1990). An organizational design allocates members to different tasks, roles and units while a product design allocates functionality into different components. Members of an organization interact and communicate to accomplish their work; similarly, components in a product design interrelate to implement the functionality of the product. Because developers must communicate more intensely when their tasks are interdependent (Allen, 1977) and because components of a product can be interdependent, a natural proposition is that the partitions and connections between members of the organizational structure will be aligned with or “mirror” those in the product design structure (Simon, 1962; Henderson and Clark, 1990; Baldwin and Clark, 2000).

Although a mirroring hypothesis seems straight-forward, empirical evidence of it to date is mixed. For example, Sosa and colleagues (Sosa, et al., 2004, 2003) have examined how design interfaces in an aircraft engine corresponded to communication patterns in the team that developed the engine. Although the study found some evidence of a match between team communications and product design, the data also suggested several “mis-matches” in which the product design and team communications were not in sync. This prompted the researchers to invoke the bounded capacity of human cognition (Simon, 1962) as one potential reason for “mis-alignments” between product design and organizational structure.

Compared with other classical fields of engineering, the development of software applications is sometimes considered more complex since it is more human-centered (Brooks,

1995). Furthermore, the output of the development process, i.e. the software, is often wrongly perceived as easily modifiable thanks to its malleability (Ghezzi, 2003). Software engineering has the aim of providing rules of guidance on how to structure the development process and the development team in order to effectively build large and complex software architectures. Given the malleability of software and the human-centered nature of the development process, the relationships between the team of designers and their software product is tight and multi-faceted. Previous studies have pointed out that, when addressing the design of a complex system, the structure of the system influences the organizational structure of its design team (e.g., Sosa, 2004). In the software engineering domain, the reverse relationship may hold too. Given the tight relationship between developers and software, the organizational structure can be posited to have an impact on the product structure. In fact, the notion of a potential duality between the software structure and organizational structure has already been articulated more than forty years ago by Conway (1968), who noted that large organizations will always leave a “print” of their organizational communication structure in the design of their software products.

Conway’s conclusions were drawn when software products were primarily designed ad hoc and developed within firms in a classic Closed Source context. Nowadays, the Open Source development paradigm is challenging the classic way of developing software. Thousands of geographically dispersed developers work together, producing high-quality software products which in many cases succeed in gaining market share to the detriment of established closed-source competitors. Open Source development teams reflect innovative organizational structures. Open Source communities are dynamic, developers can freely join and leave the community, the assignment of tasks is self-determined, and, in some cases, projects split into distinct parallel versions to address the needs of a specific sub-group of users. In this context, the relationships

between the software product and the organizational structure can be even closer than in classical Closed Source development.

The aim of this paper is to investigate the co-evolution of the structure of software and the organizational structure of the team of its developers, in Open and Closed Source projects. The study analyzes a unique and detailed dataset on the evolution of a set of Closed Source enterprise software applications and their development teams over more than twenty years and compares these applications with Open Source counterparts. In particular, the paper investigates the structural characteristics of the social network of designers, as a fundamental component of the organizational structure of modern development teams. The paper takes an evolutionary perspective and analyzes how software design structure and social network structure are related over time. This evolutionary perspective represents an imperative, since the mutual causal relationship between variables cannot be understood with a static analysis. Finally, the paper evaluates the relative influence of the organizational structure versus the software structure on the performance of the projects, comparing productivity in Open Source to Closed Source.

The paper is structured as follows. Section 2 defines the key structural dimensions of software designs and social networks and presents our research hypotheses of their co-evolution, providing theoretical background and motivations. Section 3 describes the research methodology, focusing on the data sample and discussing the operationalization of the variables we have used in this study. Section 4 presents the empirical results of our testing, while Section 5 draws out conclusions and implications of our findings.

THEORY AND HYPOTHESES

In this section we start by defining the key structural dimensions of software design and then define the key structural dimensions of social networks. We continue by considering how the

dimensions can be aligned. Finally, we theorize how the software and organizational structures will be aligned in the context of Open Source and Closed Source projects and whether the software or organizational structure will be more important for development performance.

Dimensions of Software Structures

The essence of software development involves abstracting a design and coding that design (Card and Glass, 1990). Inherent in a given design is a certain level of complexity (Brooks, 1995). To manage that complexity, a fundamental approach to designing software is functional decomposition (Dijkstra, 1968; Parnas, 1972) which involves splitting the design implementation into parts or modules. In turn, modules consist of data structures and one or more functions.

Two primary structural dimensions of a design are coupling and cohesion. *Coupling* has been defined as “the measure of the strength of association established by a connection from one module to another” (Stevens et al., 1974, p. 117). *Cohesion* was originally described as binding - “Binding is the measure of the cohesiveness of a module” (Stevens et al., 1974, p. 121). Cohesion subsequently replaced binding as the term used for this dimension of software design (Bieman and Ott, 1994). The original definition and subsequent treatments of this structural design property rely on the notion of ‘togetherness’ of processing elements within a module to define cohesion. It is important to note that coupling and cohesion are not independent dimensions of software design but rather are involved in mutual influence relationships (Darcy et al., 2005). Since a goal of this paper is to analyze the evolution of the structure of software systems, is it important to understand and clarify how the dimensions of software structure, namely coupling and cohesion, relate to each other and what are the effects of different combinations of values of the two dimensions on the global software design structure. FIGURE 1 provides a conceptual overview of such effects by means of four examples. Although each

dimension is a continuous variable, for the sake of simplicity we have considered only “good” or “bad” values of variables, leading to four combinations of coupling and cohesion. Generally, a software design is considered to have a good level of coupling whenever the number of ties between its modules is not too high and, as a consequence, the code is easier to maintain. Conversely, it is also desirable that all the modules of a software design show a high degree of cohesion, in order to avoid the distribution and duplication of functionalities across different modules (Ghezzi, 2003).

Insert Figure 1 about here

Cases ① to ④ in FIGURE 1 show different combinations of “good” and “bad” values of coupling and cohesion. For example, Case ④ in FIGURE 1 corresponds to bad coupling and bad cohesion: the number of ties between modules is high (as in Case ②) and there also exist ties between non-related parts of modules (e.g. ties between modules M_2 and M_4) and between different parts of the same module (e.g. modules M_2 and M_3).

Dimensions of Organizational Structures

We conceptualize organizational structures in terms of social networks. A social network consists of the interconnections between two or more actors or nodes (Knock and Yang, 2007). One of the most important structural aspects of a social network is the degree of *centrality*. 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 invokes notions like social power, influence, and prestige. Over time, several dimensions have been introduced to formalize and 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 (2005) 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 (1997) extends the generality of betweenness centrality by suggesting a conceptual closeness between structural holes and betweenness centrality. 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.

In this paper, we focus on normalized degree/closeness and betweenness centrality, according to the definitions provided by Wasserman and Faust (1994) and Beauchamp (1965). Degree/closeness and betweenness centrality represent the most intuitive and widely used metrics of centrality. We acknowledge that eigenvector centrality is also a theoretically important metric of centrality in the field of social networks, as discussed by Borgatti et al. (2006). However, its greater conceptual and computational complexity makes it more difficult to use in empirical research on large social networks.

FIGURE 2 provides a conceptual representation of network topologies with different levels of betweenness and degree/closeness centrality (“high” or “low”). Case ① in FIGURE 2 considers both dimensions to have their lowest value, i.e. zero. No links exist between the nodes of the network and no information is exchanged, since nodes operate in complete isolation. Case ④ represents the other extreme of the continuum, where the network is a fully connected graph (each node is connected to all other nodes). In this case, the average value of degree/closeness is at a maximum, since each node has the maximum number of connections and, as a consequence, is “as close as possible” to other nodes. To the contrary, the average value of betweenness is lowest, since there are minimum connection paths of unitary length between all nodes.

Insert Figure 2 about here

Case ③ describes the opposite situation of “low degree/closeness, high betweenness”. In this case, each node is connected only to its two neighbors, thus, betweenness is constant regardless

the number of nodes in the network. Finally, Case ② depicts the situation of “high degree/closeness, high betweenness”. This topology can be considered a more efficient way to provide good information sharing among network nodes without limiting the scalability of the network. However, a few “gatekeepers” are in charge of knowledge exchanges, which may be single points of failure.

Mutual influence between dimensions of software and organizational structures

As noted earlier, the literatures on organizations and product design suggest that organizational and product designs are inter-related (Henderson and Clark, 1990). Basically, the connections between the product and organizational design are driven by the desire to manage complexity via partitioning the task (Simon, 1962). When a complex task such as a software design is functionally decomposed into parts or modules, the labor to complete the task may also be decomposed such that each module is accomplished by a different set of developers. If the modules are independent or loosely coupled (such as in a modular design), the respective developers do not need to interact. However, if the modules are strongly inter-connected or coupled (such as in a monolithic design), the developers must communicate and coordinate intensely in order to manage the inter-dependencies.

The “mirroring” hypothesis proposed in the product development literature suggests that the social structure of the organization should match the design structure of the product to optimize communication and coordination in the design task (Henderson and Clark, 1990; Baldwin and Clark, 2000). For example, a modular product is developed by a modular organization while a “monolithic” product is developed by a “monolithic” organization. One of the most referenced works in this field is the analysis conducted by Henderson and Clark (1990). The authors provide evidence that firms often experience difficulties when adapting to the architectural innovation of

products, since their internal organizational structure tends to embed the characteristics of the product. More recently, Sosa et al. (2004) have provided an empirical analysis of the alignment of product and organizational structure design in the production of large commercial aircraft engine. Their results show that product design and team interactions tend to be matched throughout the network such that components with strong connections also have development teams who communicate intensely.

Given the particular characteristics of software, the properties of the interdependency between product and organizational structure need to be reconsidered. DeSouza et al. (2005) presented a visual approach to the analysis of the interdependencies between software and network structure. They show that both the software and the network are important intermediaries of information and knowledge sharing in software development communities, since they are both used to achieve agreements on socio-technical issues. While the structure of the product is known to influence organizational structure, there is no evidence of the reverse relationship in traditional studies. Based on the results of DeSouza et al. (2005), both software and organizational structure represent important intermediaries of information. There is also evidence that organizational structure tends to embed the characteristics of software (Henderson and Clark, 1990). These results suggest that, unlike the relationship between a physical product and the related organizational structure, the dependency between software and organizational structures might be mutual. This mutual relationship suggests a far more complex product and organizational structure design process.

As we have noted, the classic product management literature indicates that in typical engineering products the structure of the product affects the structure of the organization (Henderson and Clark, 1990; Sosa, 2004, DeSouza, 2005). However, the software product has

specific characteristics that make it different from any classical engineering product (Conway, 1968; Ghezzi, 2003; DeSouza et al., 2005). As described by Brooks (1995), software is almost infinitely malleable and not bound by typical physical or material constraints. Thus, the traditional causal relationship between product and organizational structure cannot be assumed to hold. Given the critical role of human labor in software design and the adaptability of software, the opposite relationship may hold too. These considerations lead us to propose that:

Proposition 1: *In software development environments, the product structure and the organizational structure will mutually influence each other.*

As a consequence of Proposition 1, it is informative to analyze the relationship between the software structures of FIGURE 1 and the network topologies of FIGURE 2. This can help us understand whether distinct software structures relate to corresponding distinct organizational structures. In the following, we propose a mapping between the different structures shown in Figures 1 and 2.

Case ① – *Software structure: low coupling, high cohesion; Network structure: low betweenness, high degree/closeness.* If the network structure exhibits a fully connected topology, the software is likely to have low coupling and high cohesion. Given the ease with which knowledge can be exchanged and information can be shared, each developer is more likely to be aware of the structure and the functionalities of software modules developed by the other team members. In this situation, software modules are likely to be highly cohesive, since no functionalities are duplicated and tied with a low coupling within a common architecture shared among all designers.

Case ② – *Software structure: high coupling, high cohesion; Network structure: high betweenness, high degree/closeness.* In this case, the network is composed of multiple teams

working in parallel connected to each other through a hub node. This kind of network structure is similar to that of classical closed source software development, in which development teams (each separate group of connected nodes) are linked by means of hierarchical levels of management (the connection hubs). Typically, the overall software design is planned centrally and shared among the development groups, leading to good cohesion. However, nodes communicate to each other through central hub nodes that do not allow direct knowledge exchanges and, thus, can create inefficiencies. The level of knowledge of other teams' modules is lower and the integration of modules produced by different teams is more difficult, increasing the level of coupling of the resulting software.

Case ③ – *Software structure: low coupling, low cohesion; Network structure: high betweenness, low degree/closeness.* This case represents good inter-group information sharing, since different working groups can easily exchange information (connections among groups are provided by “entry points” which promote knowledge sharing with other group members). This results in low coupling, since interfaces in the product design can benefit from the interfaces between working groups in the development team. However, group members can communicate only with a few neighbors and can have many degrees of separation from non-neighboring nodes, even if they belong to the same working group. This property of the network is likely to translate into a low level of cohesion of software modules, since functionalities may be duplicated because of low intra-group information sharing.

Case ④ – *Software structure: high coupling, low cohesion; Network structure: low betweenness, low degree/closeness.* In this network topology, each developer works in complete autonomy and has no knowledge exchange with any other team member. The only coordination mechanism is the formal design documentation (if any). As a consequence, the software product

is likely to be characterized by a low level of cohesion, since each developer is likely to repeat the needed functionalities in his/her modules, as well as a high degree of coupling, since there is limited or no information sharing on module interfaces, leading to unnecessary dependencies.

Based on these considerations, it is straightforward to argue that the network configuration of Case ② in FIGURE 2 corresponds to the Closed Source software development context. As previously discussed, classical software development projects are typically structured with many development teams coordinated and linked together by means of several management layers. Different management configurations can be represented by means of different connection topologies among the central connection hub nodes.

The Open Source software development context is not easy to directly associate with any of the four network topologies in FIGURE 2. As largely documented in the literature (Mockus et al., 2000; Koch and Schneider, 2002), the typical Open Source network topology presents an onion-like or *core-periphery* structure, of which FIGURE 3 provides a sample layout. The core group of members is constituted by the leading developers, and is typically in charge of executing important processes, such as the development of a shared understanding of user needs, the design of the system architecture, the resolution of integration conflicts, and so on (Crowston et al., 2006). The periphery is instead composed of a larger group of developers, whose typical activities are less critical for the project, but still important, as they include bug reporting, bug fixing, development of patches and plug-ins, etc.

The core-periphery topology can be considered a hybrid between Cases ① and ④ of FIGURE 2. The core part of the network topology represented in FIGURE 3 can be clearly mapped to the completely connected network topology of Case ①, while the loosely connected nodes of the periphery have the layout of Case ④ (without the central core). As noted in previous

literature on Open Source (VonKrogh et al., 2003; Howison et al., 2006), network nodes are typically distributed with an 80/20 proportion between periphery and core. This network structure is more efficient in managing larger groups with respect to network topology of Case ①, without introducing the potential inefficiencies of the Closed Source topology of Case ②.

Insert Figure 3 about here

The literature and software practice widely recognize that the Open Source development model is considerably different from the Closed Source one. The former is generally perceived as more informal, more dynamic and open to innovations, and having less tight schedules. It is worth noting that such differences are directly related to the organizational structure, whose peculiar characteristics influence the software production process. In particular, it should be noted that the network of developers in Open Source projects (especially community projects such as those retrieved from online repositories like SourceForge.net) shows an extremely high degree of dynamicity, since new members continuously join the community (as noted by the literature on scale-free networks, cf. Xu et al., 2006) and the turnover of community members is high compared to the Closed Source context (as noted by Howison et al., 2006, member changes happen frequently in the periphery, but also in the core). Consistent with these considerations, we adapt Proposition 1 to take into account the different characteristics of Closed and Open Source. This leads to our first set of research hypotheses:

***H1A:** In Closed Source projects the structure of the organization and the structure of the software mutually influence each other.*

***H1B:** In Open Source projects the structure of the organization affects the structure of the software, but the structure of software does not affect the structure of the organization.*

Team coordination and management is a predominant activity in the software development process (Kraut and Streeter, 1995), and has been widely addressed in the information systems literature (Zmud 1984; Kim 1988, 1993; Malone and Crowston, 1994). Coordination has been defined by Van de Ven et al. (1976) as "the linking together [of] different parts of an organization to accomplish a collective set of tasks". Coordination mechanisms are fundamental levers to address the management of task-planning activities and decision-making processes. As a consequence, the productivity of development teams is directly affected by the coordination mechanisms adopted in software development processes (Chiang and Mookerjee, 2004).

In the classic software development context, the organizational structure is used as coordination mechanism, since decisions taken at the higher hierarchical levels are addressed and structured through the middle levels and implemented by the low-level development teams (this kind of coordination is typically referred to as *vertical coordination*). Conversely, in the Open Source context, no explicit and formal organizational structure is present. As previously discussed, the network is highly dynamic and team members are likely to change, even within the core. As a consequence, the only coordination mechanism that can be effectively exploited is the software product itself, which becomes the only way to share design decisions over time and coordinate tasks among the community members (Tervonen and Kerola, 1998). Based on these considerations, we posit our second set of research hypotheses:

H2A: *In Closed Source projects the organizational structure will affect productivity more than the software structure as coordination and information sharing are done using the organizational structure.*

H2B: *In Open Source projects the software structure will affect productivity more than the organizational structure as coordination and information sharing are done using the product.*

RESEARCH METHOD

Our hypotheses are evaluated empirically, comparing data on enterprise software applications in Closed Source and Open Source projects. This section describes the data sample used to evaluate our hypotheses and presents the operationalization of the variables involved.

Sample

Closed Source Projects. The Closed Source data include a sample of 23 enterprise software applications from one of the major retailing companies in the U.S. Selected applications are all written in COBOL and are components of the Fiscal, Human Resources, Merchandising, and Operations subsystems of the company's enterprise system. For each project, the complete change history at the individual program level is available, in the form of change logs (Kemerer and Slaughter 1999). Each change log is related to a specific software module, and includes information on its creation date, author, and function, as well as information about the change that is performed. Data are related to a time period of more than twenty years from the 1970's to 1990's; during this time period, more than 28,000 change logs were created by more than 300 distinct authors. Software design structure metrics were computed by submitting each software module to a commercial source code static analysis tool.

Open Source Projects. The Open Source data set is comprised of a sample of 37 enterprise software applications from the *SourceForge.net* online repository. Since the analysis of applications extracted from online repositories can lead to controversial results because of the varying quality of available data (Howison and Crowston, 2004), we have enforced the following selection 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: all application have been selected from the Office/Business domain, and have been further filtered by considering the E-Commerce/Shopping, Enterprise, Financial, Project Management, Scheduling, and Time Tracking sub-domains of the *SourceForge.net* domain hierarchy.

For each application, all available releases on the SourceForge.net repository have been downloaded, leading to a total of 693 versions. The release history of selected applications ranges on a time span of about seven years, from December 2000 to November 2007. TABLE 1 provides an overview of the summary statistics of the two data sets.

 Insert Table 1 about here

Since the focus of this study is to analyze the evolution of the relationships between organizational and software design structures over time, we have computed all the metrics described earlier in each time period for both samples. We define a “version” or *timeframe* i for each *application* k as the basic unit over which our analyses are conducted. For the Closed Source sample, each timeframe is defined as a four month period. Eighty-three timeframes are thus identified over the twenty years of development. In order to obtain a comparable number of data points, the timeframe for the Open Source sample is defined as a one month period, leading to eighty-three timeframes between December 2000 and November 2007.

Variable Definition and Operationalization

Software Structure Dimensions

As discussed earlier, this paper focuses on coupling and cohesion as fundamental dimensions of software structure. Since the Closed Source application is written in a procedural language, the coupling COU^{CS} of a software module (in this case, of a program) is measured as the number of calls to external modules per function point (Banker and Slaughter, 2000):

$$COU^{CS} = \frac{calls}{FP}. \quad (1)$$

The average coupling $COU^{CS}_k(i)$ of version i of application k is defined as the average coupling of its modules, while the average coupling COU^{CS}_k of application k is defined as the average coupling of its versions. The cohesion COH^{CS} of a software module in the Closed Source application is defined as the Halstead's number of operands (Halstead, 1977) per function point:

$$COH^{CS} = \frac{n_2}{FP}. \quad (2)$$

The average cohesion $COH^{CS}_k(i)$ of version i of application k is defined as the average cohesion of its modules, while the average cohesion COH^{CS}_k of application k is defined as the average cohesion of its versions.

The Open Source applications are written in Java, an object-oriented programming language. Coupling and cohesion have been computed on the basis of two of the metrics included in the Chidamber and Kemerer's metrics suite for object-oriented systems (Chidamber and Kemerer, 1994). The coupling COU^{OS} of a software module (in this case, of a class) is defined as the Coupling Between Objects (CBO) metric normalized to the number of function points:

$$COU^{OS} = \frac{CBO}{FP}. \quad (3)$$

The average coupling $COU^{OS}_{k(i)}$ of version i of application k is defined as the average coupling of its modules, while the average coupling COU^{OS}_k of application k is defined as the average coupling of its versions. The cohesion COH^{OS} of a software module is defined as the Lack of Cohesion in Methods (LCOM) metric normalized to the number of function points:

$$COH^{OS} = -\frac{LCOM}{FP}. \quad (4)$$

Since the LCOM metric measures lack of cohesion, a minus sign has been added to obtain values consistent with those defined for the Closed Source context. The average cohesion $COH^{OS}_{k(i)}$ of version i of application k is defined as the average cohesion of its modules, while the average cohesion COH^{OS}_k of application k is defined as the average cohesion of its versions. The metrics were computed by analyzing the Java bytecode of all the versions of the selected applications by means of a tool developed by an author of this paper. The tool performs static analyses of the Java bytecode and is based on the Apache BCEL bytecode engineering library.

Organizational Structure Dimensions

Social network model. Social networks can be modeled as two-mode undirected affiliation networks (Wasserman and Faust, 1994), where development team members represent the actors and software projects represent the events. A node d representing a team member is thus associated with another node p representing a software project when d is part of p 's team of contributors. In order to perform social network analyses to measure the dimensions of the organizational structure, we have derived single-mode social networks from the two-mode affiliation networks considering team members only. In the single-mode social networks all nodes represent team members, and two nodes are linked when both team members are working together on the same software project. As a consequence, two nodes d_1 and d_2 are directly

connected if and only if they were connected to the same software project node p in the two-mode network. We refer only to the single-mode team members networks.

In the Closed Source context, the relationships between team members are provided by the change event logs. Each log is related to a change performed on the system and reports the software module description, the date of the change, a textual description of the change, and the developer who has performed the change. Two developers are considered part of the same development team if they have performed changes to modules belonging to the same application. In the Open Source context, social networks have been derived by querying the *SourceForge.net* data warehouse (Gao et al., 2007), which provides access to temporal snapshots of the actual *SourceForge.net* operational database of contents and metadata. Two developers are considered part of the same development team when at least one of the following conditions holds:

- they are listed as members of the same project;
- they are listed as members of two different projects, but are connected by a third developer who is listed as member of both projects.

The second condition allows to include in the network also the one-degree developers associated to those directly connected to the projects of our sample, in order to take into account the more dynamic context of Open Source communities with respect to the more static Closed Source development teams.

Organizational structure dimensions. Since the relationships between actors in our social network model are undirected, social network analyses have been based on the three classic centrality metrics as suggested by Knoke and Yang (2008). All the metrics have been computed using *Pajek* (Batagelj and Mrvar, 1998). The *normalized actor degree centrality* (Wasserman

and Faust, 1994) $C_D(N_i)$ 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 :

$$DEG = C_D(N_i) = \rho(N_i) / (N-1). \quad (5)$$

Actor degree centrality is a measure of the proportion of the network members to which a node is directly connected. A node with a high degree centrality can be supposed to be greatly involved in the activities of the social network, as it is likely to be able to reach (and be reached from) a high number of other nodes.

The *normalized actor closeness centrality* (Beauchamp, 1965) $C_C(N_i)$ of node N_i is defined as the ratio of the total number of nodes in the network excluding node N_i , to the sum of the geodesic distances between node N_i and the other $(N-1)$ nodes in the network:

$$CLO = C_C(N_i) = \frac{(N-1)}{\sum_{j=1}^N d(N_i, N_j)} \quad (\text{with } i \neq j), \quad (6)$$

where $d(N_i, N_j)$ represents the length of the geodesic path between nodes N_i and N_j . As suggested by Sabidussi (1966), the closeness centrality of actors is a measure of how near a node is to the other nodes in the network. A node with high closeness centrality is supposed to be able to interact with other nodes more quickly, as very few intermediaries are involved in the communication.

The *normalized actor betweenness centrality* (Wasserman and Faust, 1994) $C_B(N_i)$ of node N_i is defined as the average frequency with which node N_i is involved in a geodesic path between two generic nodes N_j and N_k :

$$BET = C_B(N_i) = \frac{2}{(N-1)(N-2)} \sum_{j < k} \frac{g_{jk}(N_i)}{g_{jk}} \quad (7)$$

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 geodesic paths between nodes N_j and N_k crossing N_i . The metric is normalized to the maximum number of geodesic paths crossing N_i in an undirected network with N nodes. Actor 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.

Project Performance

We measure project performance or productivity $P_k(i)$ related to version i of application k as function points per person-day by the following expression:

$$PROD = P_k(i) = \frac{\Delta FP_k(i)}{t_k(i) \cdot d_k(i)}, \quad (8)$$

where $\Delta FP_k(i)$ is the number of delivered function points, defined as $\Delta FP_k(i) = FP_k(i) - FP_k(i-1)$ with $FP_k(i)$ indicating the number of total function points of version i ; $t_k(i)$ is the development time, defined as the number of days elapsed between the release of version i and previous version $i-1$, and $d_k(i)$ is the number of members in the development team. Components of Expression (8) must be properly defined in order to take into account the differences between the Closed Source and the Open Source contexts. In the Closed Source context, the number of members of the development team $d_k(i)$ is defined as the number of unique full-time developers working on version i of application k :

$$d_k(i) = \sum_{j=1}^{n_k(i)} \frac{1}{p_j}, \quad (9)$$

where $n_k(i)$ is the total number of developers who have worked on version i of application k , and p_j is the number of different applications on which each developer was working on concurrently.

Expression (9) let us take into account that a developer can work on more than one application at a time and, thus, should not be considered as a full-time developer. In the Open Source context, the number of members of the development team $d_k(i)$ of Expression (8) is defined as the number of unique full-time active developers working on version i of application k :

$$d_k(i) = \alpha_k^{admin} \cdot \beta_k^{admin} \cdot n_k^{admin}(i) + \alpha_k^{devel} \cdot \beta_k^{devel} \cdot n_k^{devel}(i), \quad (10)$$

where $n_k^{admin}(i)$ is the number of administrators and $n_k^{devel}(i)$ is the number of developers of version i of application k as listed by the *SourceForge.net* project's data, thus leading to $n_k(i) = n_k^{admin}(i) + n_k^{devel}(i)$. To take into account that in the Open Source context each team member is not typically working full-time on the project and that there could be inactive team members among a project's members, the following correction factors have been introduced:

- α_k , the average fraction of time each team member (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 or developers, in order to account for the differences among the two classes of project team members.

The number of project administrators and developers that we have used is the one officially listed by *SourceForge.net*. Values of correction factors α_k and β_k of the productivity metric have been derived from an empirical survey previously conducted on 2,564 open source project administrators and developers of *SourceForge.net*, including all the 424 team members involved in the projects of our sample (Capra, et al. 2008). The questionnaire was focused on gathering the actual time each respondent devotes to each project in which he or she is involved. Data are based on a total of 653 answers (of which 424 from actually active team members), with a global response ratio of 25.5%. 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 are actively involved in a project and the total number of respondents to our survey multiplied by the total number of projects in which they were involved. TABLE 2 presents a summary of the overall results of the survey, along with the average values of correction factors.

Insert Table 2 about here

EMPIRICAL RESULTS

Before evaluating our hypotheses, we first used vector autoregression analysis (Sims, 1980) to examine whether each software and network structure dimension independently shows a significant degree of autocorrelation or *persistence*. Considering the software and the network structures as dynamic systems, the autocorrelation function is a proxy of the “memory” that each dimension has of its own past history. If the value of a dimension at time t heavily depends on its value at previous time instants ($t - \tau$), the evolution of the variable is more likely to follow a steady and regular path, without being subject to large variations (for a detailed discussion of the autocorrelation function, please refer to Papoulis and Pillai, 2001). Persistence or autocorrelation is an important property for the social and software structures, since it is a measure of their self-dependence over time. By considering the evolutionary nature of both social network and software structures, it is clear that, at time t , dimensions describing each structure must be determined by a “carryover” from the prior time period.

The vector autoregression models are used to estimate four equations each for Closed Source and Open Source in which each equation predicts the current value of a structural dimension

based on a lagged value of that dimension. TABLE 3 summarizes the results of the vector autoregression models computed with time lags $\tau = 1$. Social network metrics of degree centrality (*DEG*) and closeness centrality (*CLO*) have been considered together since they were highly correlated and thus a unique factor *DEGCLO* has been addressed. As shown in Table 3, the estimated coefficients of each dimension are significant, indicating that the lag of each variable significantly predicts the value of the current variable. These results provide evidence of autocorrelation in the evolution of software structures both in Closed Source and Open Source, consistent with Lehman’s Laws of Software Evolution (1980). Similarly, the evidence of autocorrelation in the network structure dimensions provides support to the findings of Kossinets and Watts (2006), who suggested that “average network measures remain stable over time”.

 Insert Table 3 about here

Hypotheses H1A and H1B were evaluated by performing a vector autoregression analysis with time lag $\tau = 1$ on the software and network structure dimensions for both Closed and Open Source contexts, to capture the mutual effects among the four dimensions over time. In this analysis, vector autoregression allows us to specify a model in which all the time series variables are jointly determined, capturing their dynamic and interdependent relationships. The estimation model used for vector autoregression analysis consists of the following system of equations¹:

$$\begin{aligned}
 COU_t &= c_{COU} + \beta_{1,COU}COU_{t-1} + \beta_{2,COU}COH_{t-1} + \beta_{3,COU}DEGCLO_{t-1} + \beta_{4,COU}BET_{t-1} + \varepsilon_{COU} \\
 COH_t &= c_{COH} + \beta_{1,COH}COU_{t-1} + \beta_{2,COH}COH_{t-1} + \beta_{3,COH}DEGCLO_{t-1} + \beta_{4,COH}BET_{t-1} + \varepsilon_{COH} \\
 DEGCLO_t &= c_{DC} + \beta_{1,DC}COU_{t-1} + \beta_{2,DC}COH_{t-1} + \beta_{3,DC}DEGCLO_{t-1} + \beta_{4,DC}BET_{t-1} + \varepsilon_{DC} \\
 BET_t &= c_{BET} + \beta_{1,BET}COU_{t-1} + \beta_{2,BET}COH_{t-1} + \beta_{3,BET}DEGCLO_{t-1} + \beta_{4,BET}BET_{t-1} + \varepsilon_{BET}
 \end{aligned}$$

¹ The same set of equations has been used to estimate coefficients in both the Closed and the Open Source samples.

TABLE 4 presents the results of the vector autoregression analysis. To test H1A about the mutual relationship between social networks and software structure in Closed Source, we must check whether organizational structure dimensions affect software structure dimensions, and vice versa. The coefficient and significance columns of the left panel of TABLE 4 show that organizational structure dimensions do affect software structure dimensions. Significant effects are those of betweenness on coupling ($BET_{t-1} \rightarrow COU_t$, $b = -0.078$, $p < 0.05$) and degree/closeness on cohesion ($DEGCLO_{t-1} \rightarrow COH_t$, $b = 0.095$, $p < 0.10$). Results also show the influence of software structure dimensions on organizational structure dimensions. In particular, a statistically significant coefficient is found for the relationship between coupling and betweenness ($COU_{t-1} \rightarrow BET_t$, $b = -0.178$, $p < 0.05$). As a consequence, there is some empirical support for H1A.

 Insert Table 4 about here

Considering the Open Source context (shown in the right panel of TABLE 4), the results also provide some empirical support for H1B. That is, there is evidence of organizational structure dimensions affecting software structure dimensions, but not vice versa. Considering only cross-relationships between organizational structure and software structure dimensions, the only statistically significant effects are those of degree/closeness on coupling ($DEGCLO_{t-1} \rightarrow COU_t$, $b = -0.219$, $p = 0.000$), betweenness on coupling ($BET_{t-1} \rightarrow COU_t$, $b = -0.445$, $p = 0.000$), degree/closeness on cohesion ($DEGCLO_{t-1} \rightarrow COH_t$, $b = 0.210$, $p = 0.037$), and betweenness on cohesion ($BET_{t-1} \rightarrow COH_t$, $b = 0.308$, $p = 0.076$). No statistically significant evidence of the effect of software structure dimensions on organizational structure dimensions is found.

To verify the relationships involving software and organizational structure dimensions, we performed a Granger causality test (Granger and Newbold, 1977) of the causality effects between the variables in the vector autoregression model. TABLE 5 presents the results of the Granger causality test and confirms that all the significant relationships identified by means of the vector autoregression model are confirmed.

Insert Table 5 about here

FIGURE 4 illustrates the significant relationships found among software and organizational structure dimensions for both Closed Source (case *a*) of FIGURE 4) and Open Source (case *b*) of FIGURE 4). It is worth noting that some relationships have been found to be significant in both Closed and Open Source contexts. This should be interpreted as evidence of consistent results, since in both contexts degree/closeness (*DEGCLO*) is found to affect cohesion (*COH*) with a positive sign, and betweenness (*BET*) is found to affect coupling (*COU*) with a negative sign. Specifically, these results provide empirical support for Conway's law (1968), since both Open and Closed source contexts show significant evidence of the impact of the organizational structure dimensions on the software structure dimensions.

Insert Figure 4 about here

Recall that H2A posited that the organizational structure would have a greater influence on productivity in Closed Source, while H2B posited that the software structure would have a greater influence on productivity in Open Source. To test H2A and H2B, we pooled the data and performed a panel data regression analysis using generalized least squares to estimate the effects of software and network structure dimensions for each application *i* at each timeframe *t*, with

Productivity as the dependent variable. To distinguish between Closed Source and Open Source we added a dummy variable OC_i ($OC_i = 1$ if application i is Open Source, $OC_i = 0$ otherwise). This dummy variable is interacted with each of the software structure and social network variables to identify the differences in the effects of these variables on productivity for Open Source. Variable $TIME$ has been introduced to identify the time dimension of the panel, and the variable $TIME^2$ has been introduced with the aim of modeling nonlinear effects over time. The generalized least squares regression model is thus:

$$\begin{aligned}
 PROD_{it} = & c + \beta_1 COU_{it} + \beta_2 COH_{it} + \beta_3 DEG CLO_{it} + \beta_4 BET_{it} + \beta_5 TIME + \beta_6 TIME^2 + \\
 & + \beta_7 OC_i + \beta_8 OC_i \times COU_{it} + \beta_9 OC_i \times COH_{it} + \beta_{10} OC_i \times DEG CLO_{it} + \\
 & + \beta_{11} OC_i \times BET_{it} + \beta_{12} OC_i \times TIME + \beta_{13} OC_i \times TIME^2 + \varepsilon
 \end{aligned}$$

TABLE 6 presents the overall results of the model estimation. In the Closed Source context, the only statistically significant factor affecting productivity is the betweenness of the social network ($BET_{it} \rightarrow PROD_{it}$, $b = 2.712$, $p < 0.001$). Note also that productivity improves over time in the Closed Source context ($TIME \rightarrow PROD_{it}$, $b = 0.360$, $p < 0.01$) albeit in a non-linear fashion ($TIME^2 \rightarrow PROD_{it}$, $b = -0.002$, $p < 0.05$). In the Open Source context, the coefficient on OC_i is significant at $p < 0.05$, suggesting that productivity levels in Open Source are higher than in Closed Source ($OC_i \rightarrow PROD_{it}$, $b = 9.881$, $p < 0.05$) although this effect diminishes over time as ($OC \times TIME \rightarrow PROD_{it}$, $b = -0.380$, $p < 0.05$; $OC \times TIME^2 \rightarrow PROD_{it}$, $b = 0.003$, $p < 0.10$). For the social network and software structure variables in Open Source, the only significant effect on productivity is that of cohesion, a software structure dimension ($OC_i \times COH_{it} \rightarrow PROD_{it}$, $b = 1.819$, $p < 0.05$).

 Insert Table 6 about here

TABLE 7 presents results of the tests we performed to verify H2A and H2B. To verify H2A we tested whether the coefficients of social network structure dimensions are significantly greater than the coefficients of software structure dimensions. This requires us to test all four combinations deriving from the two pairs of dimensions considered. As shown in the last column of TABLE 7, H2A is partially confirmed, since tests are supported for the *BET* variable, but not for *DEGCLO*. To verify H2B, we tested whether the coefficients of software structure dimensions are significantly greater than the coefficients of social network structure dimensions. As shown in the last column of TABLE 7, H2B is partially confirmed.

Insert Table 7 about here

DISCUSSION AND CONCLUSIONS

In this study, we examined the mutual influence between design structure and organizational structure in the Open and Closed source software development contexts. We empirically investigated whether the structural properties of software design have an influence on centrality measures of the social networks of developers, and vice-versa. Overall, our results suggest that software design structure and social network structure are tightly interconnected. However, our findings also show that significant differences exist between the Open and Closed Source contexts. We found support for Conway’s law (Conway, 1968), suggesting that the software design structure typically reflects the organizational structure of its development team. However, in Closed Source (H1A), we also found the reverse relationship, since the structure of the software was found to significantly impact the structure of the organization (consistent with the “mirroring hypothesis” in the classic product development literature, cf. Sosa, 2004). In contrast, in Open Source (H1B) empirical results supported only the effect of the social structure on

software design structure. Such differences may be motivated by the extreme dynamicity of the Open Source organizational structure, which cannot be used as a coordination mechanism, different from the Closed Source context.

Finally, we found some support for the hypothesis that, given their organizational and managerial differences, the Open and Closed Source contexts also differ in the drivers of productivity. In Closed Source, the organizational structure is the main driver of productivity (H2A), while in Open Source the main driver is the software structure (H2B). To illustrate the differences in productivity drivers in Open and Closed Source projects, we graphed the significant relationships using the estimated coefficients (see Figures 5 and 6). As can be seen in Figure 5, higher levels of software cohesion are especially beneficial in Open Source projects, in terms of positively impacting productivity levels. On the other hand, as is evident in Figure 6, higher levels of betweenness increase productivity more in Closed Source projects.

Insert Figures 5 and 6 about here

From a managerial standpoint, our results suggest that productivity has different drivers in Open and Closed Source projects. In Closed Source projects, changes in the organizational structure show a direct impact on productivity implying that managers should focus on organizational design to drive productivity improvements. Conversely, in Open Source projects, managers should focus on software structure variables to experience a direct effect on productivity, since the effects of organizational structure on performance in Open Source projects has been found to be indirect and not entirely consistent. Consistent with this paper's results, MacCormack (2006) also observes how in Open Source projects software structure

represents a fundamental coordination lever to facilitate the division of labor in geographically dispersed development teams.

We conclude with some general observations on the evolution of software systems and their related organizational structures. FIGURE 7 illustrates the evolution of the organizational and software structures in Closed and Open Source projects at selected points in time (each network layout has been generated using the Kamada-Kawai free energy layout algorithm in *Pajek*). The diagrams for the software structures are drawn based on the coupling properties of the applications in the Closed Source and Open Source projects, respectively, at various timepoints throughout the lifecycle of the projects. The diagrams for the organizational structures are drawn based on the degree/closeness and betweenness properties of the social network of developers for the Closed Source and Open Source projects, respectively, in similar timeframes as the software structures.

Insert Figure 7 about here

Although we can draw only qualitative observations on the shape of the networks over time, analyzing their temporal evolution can provide some interesting insights. Starting from the Closed Source (left column in FIGURE 7), it can be clearly noted that the first configuration of the network for the organizational structure corresponds to a completely connected graph. As we discussed in Section 3, such a network layout corresponds to Case ① of FIGURE 2, which allows ease of information flow and knowledge sharing. As the network evolves over time, the organizational structure tends to retain this network topology at least inside the development workgroup of the team. According to the managers of the proprietary firm, the actual organizational structure of their department defined three main workgroups that were assigned to

different tasks at each time. By looking at FIGURE , the three workgroups can be easily identified at time points 20 and 40, and also 60 (although less clearly); it can also be recognized that the intra-group network structure is that of Case ① of FIGURE 2, while the different groups are connected together by means of a connection component as in Case ② of FIGURE 2, which we have mapped to the Closed Source context. Based on these considerations, we can conclude that, at least in our Closed Source context, the organizational structure is planned in order to facilitate the flow of information and knowledge sharing, but also to keep the traditional control structures that can be found in the classic pyramidal organizational structure of commercial firms.

Considering organizational structures in the Open Source context (right column of FIGURE), it is worth noting that the initial social network structure starts as that of Case ② of FIGURE 2, in which different workgroups are easily identifiable and are connected by means of few nodes that act as central connection points. However, by looking at the temporal evolution of the Open Source social network, it is clear that the social network structure tends to quickly shift to the topology depicted in FIGURE 3, where one (or more) central components are surrounded by a large number of peripheral nodes. These social network configurations begin to be recognizable in the Open Source context even from the very first periods of its evolution, indicating that traditional organizational structures (and, thus, underlying coordination mechanisms) in Closed Source contexts are not used given the peculiar characteristics of the Open Source software development model.

When considering the diagrams of the software structures together with the organizational structures, the following observations can be made. First, in Open Source projects, the relationship is almost straightforward: consistent with Conway's law, the right column of Figure

7 suggests that the evolution of the software structure follows the evolution of the organizational structure. Since the only significant effect we found in Open Source projects is that of the social network on software structure, we observe that the latter evolves to mirror the shape of the community of developers that is working on it.

On the other hand, in Closed Source projects, the relationship between software and organizational structure is more complex. While we found a significant effect of the organizational structure on the software structure in these projects, we also found an effect of the software structure on the organizational structure. The left column of Figure 7 suggests three distinct phases in the evolution. In the first phase of the Closed Source projects ($t=1$), the project commences. The organizational and software structures are aligned and well organized (in particular, a model of everybody-to-everything is adopted, since the dimensions of the social network allow a complete sharing of knowledge among developers and the corresponding software). In the second phase ($t=20$, $t=40$), the project grows, and the organization tends to be more structured into workgroups to manage the complexity of the software structure (which is no longer completely connected). In this phase, both organizational and software structures are “under control”. Finally, in the last phase ($t=60$, $t=83$), due to the particular environmental characteristics of the Closed Source developers (who work together in the same offices, communicate easily, work under pressure for deadlines, etc.), the structure of the social network tends to become more entropic and less organized. This, in turn, causes the degradation of the software structure (following Conway’s law), which becomes more and more intricate. At this point, it is worth noting that a vicious circle is triggered, since in the Closed Source projects, the organizational and software structures are mutually interdependent.

A final observation that can be made from Figure 7, is that although the two software development contexts reflect significant differences in their social network structures over time, it cannot be denied that in the last time period ($t = 83$) the network topology of the Closed Source context begins to resemble the core-periphery structure of FIGURE 3, typical of Open Source development teams. This suggests that after a sufficiently long period of evolution (over 20 years), the classical network layout such as that of Case ② of FIGURE 2 may no longer be suited to handle the complexity of interactions and the information flows between the team members in Closed Source projects, since a core-periphery network topology starts to appear even in a proprietary software development environment.

Our research makes a new contribution by revealing how organizational and software structures co-evolve in different product development contexts (namely Open Source and Closed Source projects) and by identifying the relative influence of these different structures on project performance. An important implication of our study is that in order to preserve the efficacy and the efficiency of the software development process, managers should focus their attention on the organizational and software structures as coordination mechanisms. Our empirical results, in fact, highlight that in the Open Source context the major coordination mechanism is the software product, more than the organizational structure. And, even more notably, the software product is used as a coordination mechanism in order to let the network be structured as the core-periphery topology. This suggests that in Closed Source software development environments the adoption of a core-periphery network structure should correspond to a careful reorganization of the coordination mechanisms, otherwise inefficiencies could be introduced with a loss of process performance. Future research to determine the optimal social network and software structures in both contexts would be instructive.

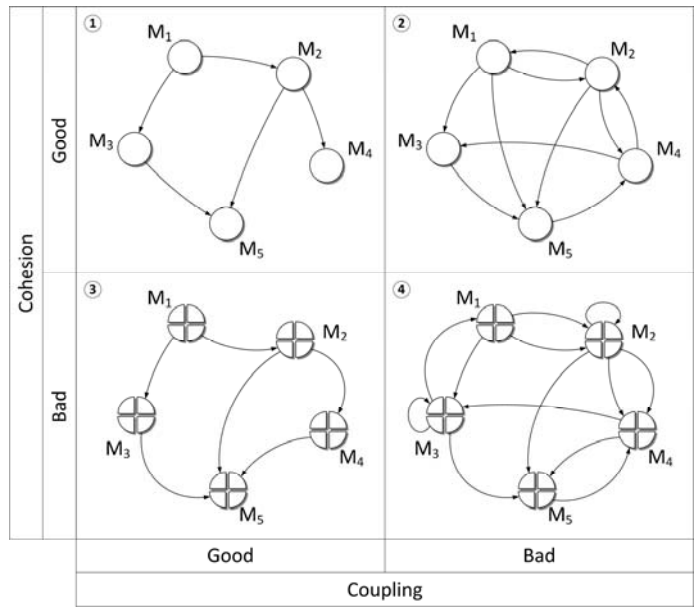


FIGURE 1. Combinations of coupling and cohesion in software structures (M=module)

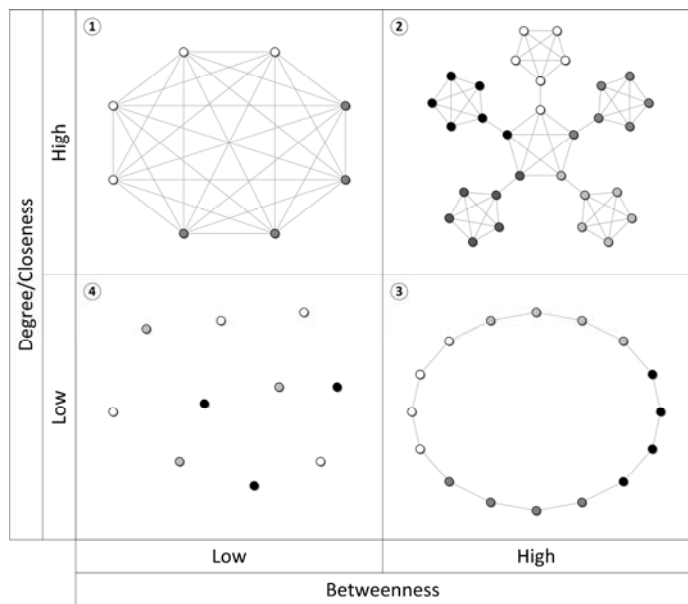


FIGURE 2. Combinations of betweenness and degree/closeness centrality in Social Networks.

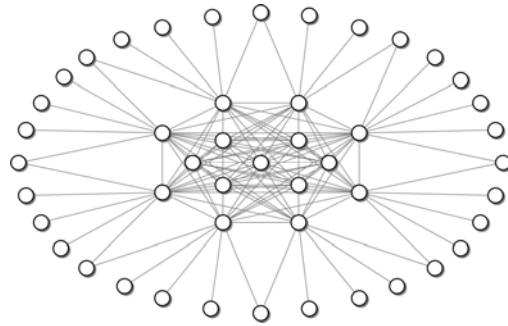


FIGURE 3. The core-periphery network structure typical of Open Source software development contexts.

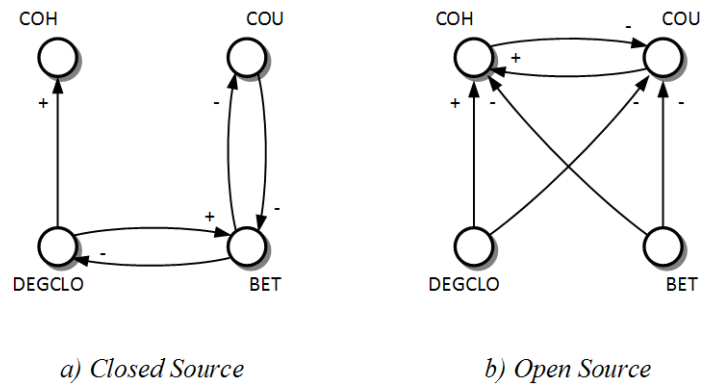
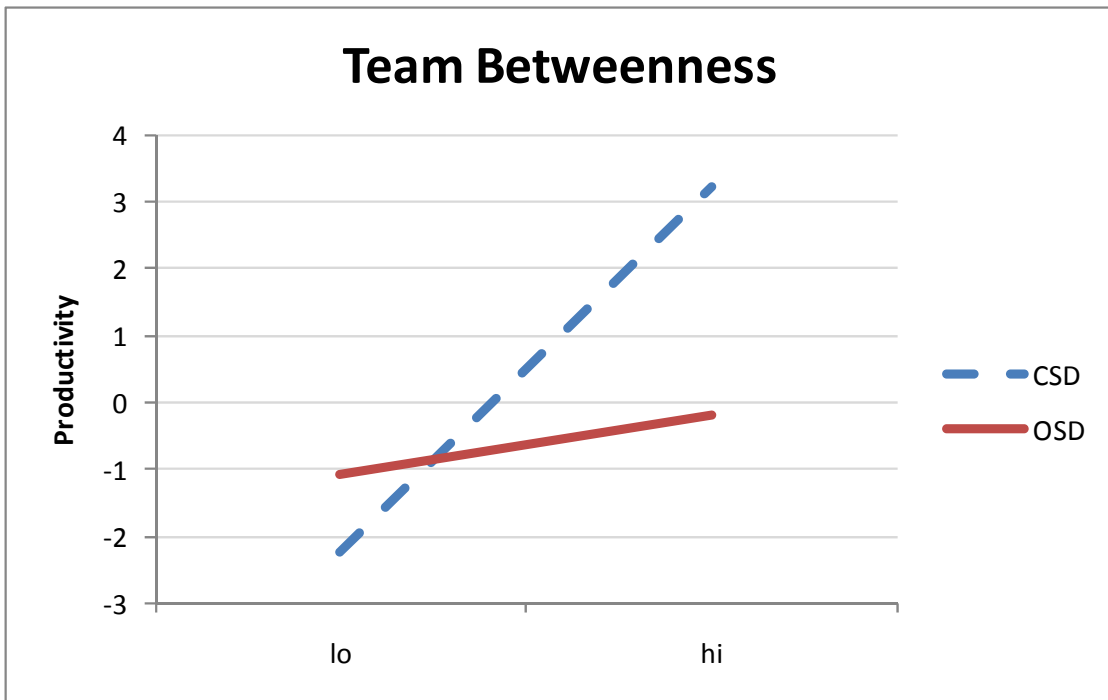
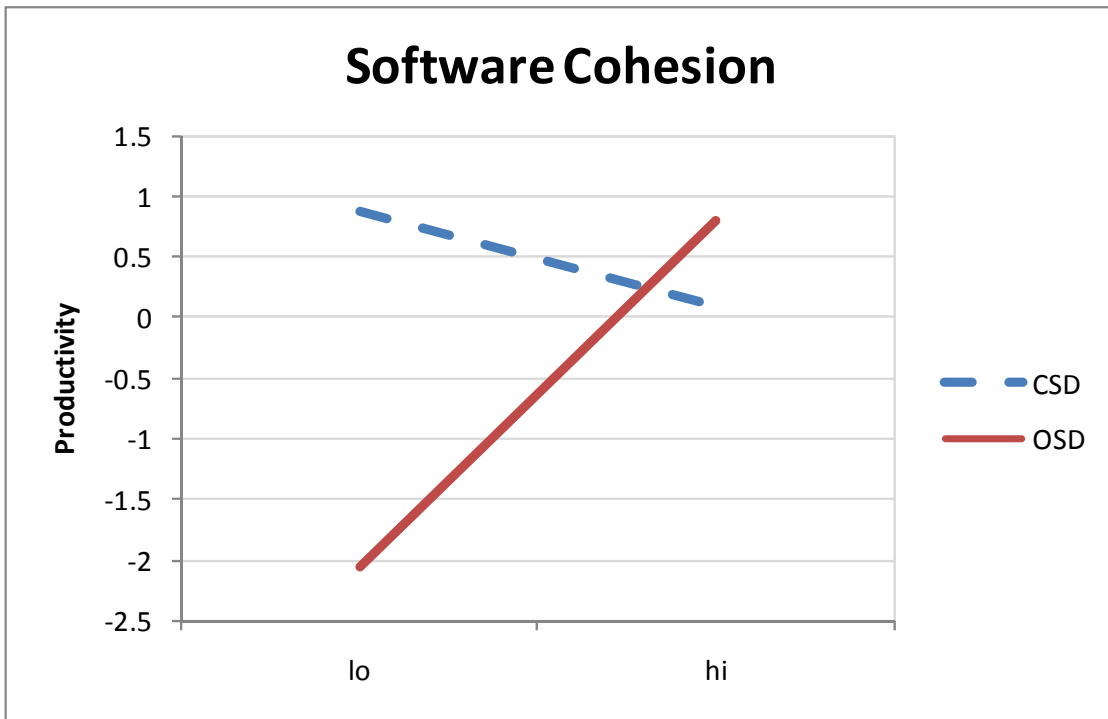


FIGURE 4. Summary overview of statistically significant relationships among software and network structure dimensions in a) Closed Source context and b) Open Source context.



FIGURES 5 and 6. Graphs illustrating the impacts of software structure (Figure 5) and organizational structure (Figure 6) on productivity for Open Source and Closed Source projects.

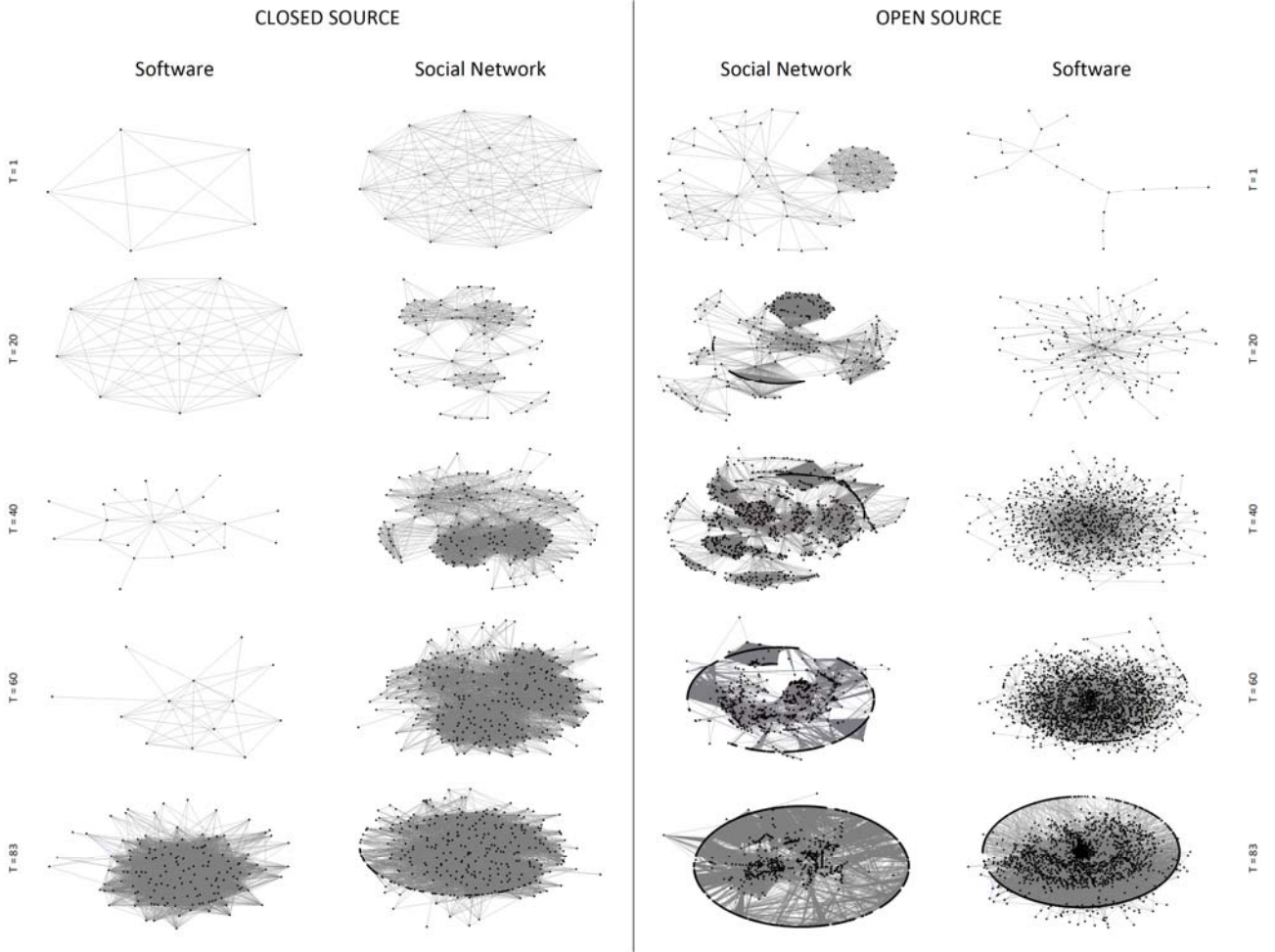


FIGURE 7. Evolution of software structures and development team structures in the Closed and Open Source contexts.

		Closed Source (N = 23)		Open Source (N= 37)	
Variable	Symbol	Value	St. Dev.	Average	St. Dev.
Average number of versions per project		15.5	±8.1	18.7	±18.9
Total number of team members		357	-	424	-
Total number of administrators		-	-	90	-
Total number of developers		-	-	334	-
Average number of team members per project	n	35.9	±22.1	11.0	±13.6
Average number of administrators per project	n^{admin}	-	-	2.0	±1.7
Average number of developers per project	n^{devel}	-	-	9.0	±12.6
Total projects size (FP)		84,136	-	56,981	-
Average project size (FP)		3,658	±3,808	1,540	±1,653
Average normalized degree centrality	C_D	0.228	±0.161	0.079	±0.049
Average normalized closeness centrality	C_C	0.509	±0.106	0.233	±0.114
Average normalized betweenness centrality	C_B	0.008	±0.006	0.003	±0.005
Average coupling between modules	$COU^{CS/OS}$	4.070	±0.002	3.245	±0.926
Average cohesion of modules	$COH^{CS/OS}$	26.730	±48.047	5.555	±1.307
Average productivity	P	5.340	±10.36	4.870	±7.59

TABLE 1. Descriptive statistics of the Closed Source and Open Source samples.

Team member class	n total	n active	Average (hr/week)	St. Dev.	Conf. Int. ($\alpha/2 = .025$)	α	β
Administrator	156	111	6.21	±7.40	±0.21	0.1553	0.7115
Developer	497	313	6.63	±9.65	±0.13	0.1658	0.6298
Global	653	424	6.41	±8.45	±0.11	0.1603	0.6493

TABLE 2. Summary statistics of data on the average productivity of Open Source team members.

Variable	Closed Source				Open Source			
	Coeff. (b)	Std. Err.	Conf. Int. ($\alpha/2 = .025$)		Coeff. (b)	Std. Err.	Conf. Int. ($\alpha/2 = .025$)	
COU	0.830***	0.049	0.735	0.926	0.487***	0.092	0.307	0.668
COH	0.911***	0.040	0.832	0.990	1.105***	0.095	0.919	1.290
DEGCLO	0.838***	0.053	0.734	0.942	0.953***	0.063	0.830	1.077
BET	0.803***	0.068	0.670	0.937	0.929***	0.125	0.685	1.173

Key: n = 83 for each panel. Significance levels. * at $p < .10$, ** at $p < .05$, *** at $p < 0.001$.

TABLE 3. Results of vector autoregression models of software and network structure dimensions.

		CLOSED SOURCE				OPEN SOURCE			
Ind. Var.	Dep. Var.	Coeff. (β)	Std. Err.	Z	Sig. ($P > z $)	Coeff. (β)	Std. Err.	Z	Sig. ($P > z $)
COU	COU	0.830	0.049	17.06	0.000	0.487	0.092	5.30	0.000
	COH	-0.055	0.037	-1.52	0.129	-0.216	0.049	-4.39	0.000
	DEGCLO	0.026	0.047	0.55	0.582	-0.219	0.052	-4.19	0.000
	BET	-0.078	0.038	-2.05	0.041	-0.445	0.090	-4.92	0.000
	Constant	-0.040	0.027	-1.50	0.133	0.040	0.019	2.09	0.036
COH	COU	-0.040	0.054	-0.74	0.457	0.339	0.176	1.92	0.055
	COH	0.911	0.040	22.62	0.000	1.105	0.095	11.66	0.000
	DEGCLO	0.095	0.053	1.80	0.072	0.210	0.100	2.09	0.037
	BET	0.028	0.042	0.66	0.507	0.308	0.174	1.77	0.076
	Constant	-0.024	0.030	-0.82	0.410	0.017	0.037	0.46	0.649
DEGCLO	COU	-0.074	0.055	-1.35	0.178	-0.008	0.111	-0.07	0.945
	COH	0.034	0.041	0.84	0.399	-0.075	0.060	-1.26	0.206
	DEGCLO	0.838	0.053	15.77	0.000	0.953	0.063	15.13	0.000
	BET	-0.092	0.043	-2.16	0.031	0.072	0.109	0.66	0.510
	Constant	-0.063	0.030	-2.09	0.037	-0.012	0.023	-0.51	0.608
BET	COU	-0.178	0.087	-2.04	0.041	0.070	0.127	0.56	0.578
	COH	0.005	0.065	0.07	0.944	0.056	0.068	0.82	0.411
	DEGCLO	0.313	0.085	3.68	0.000	0.012	0.072	0.17	0.865
	BET	0.803	0.068	11.81	0.000	0.929	0.125	7.46	0.000
	Constant	0.001	0.048	0.01	0.996	-0.049	0.027	-1.88	0.060

Note: N = 83 for each panel. Significance levels are shown in Table. Significant variables are in Bold type.

TABLE 4. Vector Autoregression analysis of software and network structure dimensions for Closed Source (left panel) and Open Source (right panel)

Equation	Excluded factor	CLOSED SOURCE			OPEN SOURCE		
		X ²	d.f.	Prob. > X ²	X ²	d.f.	Prob. > X ²
COU	COH	2.298	1	0.129	19.256	1	0.000
	DEGCLO	0.303	1	0.582	17.593	1	0.000
	BET	4.189	1	0.041	24.208	1	0.000
	ALL	15.374	3	0.002	24.582	3	0.000
COH	COU	0.554	1	0.457	3.678	1	0.055
	DEGCLO	3.245	1	0.072	4.370	1	0.037
	BET	0.440	1	0.507	3.142	1	0.076
	ALL	4.751	3	0.191	4.637	3	0.200
DEGCLO	COU	1.816	1	0.178	0.005	1	0.945
	COH	0.710	1	0.399	1.598	1	0.206
	BET	4.655	1	0.031	0.434	1	0.510
	ALL	4.845	3	0.183	27.220	3	0.000
BET	COU	4.169	1	0.041	0.309	1	0.578
	COH	0.005	1	0.944	0.677	1	0.411
	DEGCLO	13.523	1	0.000	0.029	1	0.865
	ALL	18.669	3	0.000	2.218	3	0.528

Note: N = 83 for each panel. Significance levels are shown in Table. Significant variables are in Bold type.

TABLE 5. Granger causality test on software and network structure dimensions.

Variable	Coeff. (b)	Std. Err.	z	Sig. (P> z)
Constant	-10.652	2.847	-3.740	0.000
COU_{it}	0.958	0.601	1.590	0.111
COH_{it}	-0.390	0.497	-0.790	0.432
DEGCLO_{it}	0.463	0.825	0.560	0.574
BET_{it}	2.712	0.789	3.440	0.001
TIME	0.360	0.118	3.050	0.002
TIME²	-0.002	0.001	-2.300	0.021
OC_i	9.881	4.790	2.060	0.039
OC_ixCOU_{it}	2.025	1.605	1.260	0.207
OC_ixCOH_{it}	1.819	0.864	2.100	0.035
OC_ixDEGCLO_{it}	0.957	1.372	0.700	0.486
OC_ixBET_{it}	-2.274	1.545	-1.470	0.141
OC_ixTIME	-0.380	0.196	-1.930	0.053
OC_ixTIME²	0.003	0.002	1.810	0.071

Note: N = 166. Estimated using a correction for correlated and heteroscedastic panels. Significance levels are shown in Table. Significant variables are in Bold type.

TABLE 6. Panel data results of Productivity.

Hyp.	Test	z	Sig. (P> z)	Result
H2A	BET_{it} > COU_{it}	3.70	0.0271	verified
	BET_{it} > COH_{it}	14.34	0.0001	verified
	DEGCLO _{it} > COU _{it}	0.15	0.3481	not verified
	DEGCLO _{it} > COH _{it}	1.04	0.1536	not verified
H2B	COU_{it}+OC_ixCOU_{it} > BET_{it}+OC_ixBET_{it}	5.11	0.0119	verified
	COU _{it} +OC _i xCOU _{it} > DEGCLO _{it} +OC _i xDEGCLO _{it}	0.91	0.1665	not verified
	COH _{it} +OC _i xCOH _{it} > BET _{it} +OC _i xBET _{it}	1.21	0.1353	not verified
	COH _{it} +OC _i xCOH _{it} > DEGCLO _{it} +OC _i xDEGCLO _{it}	0.01	0.4967	not verified

Note: significance levels are shown in Table. Verified tests are in Bold type. Tests are one sided.

TABLE 7. Results of hypothesis tests for research hypotheses H2A and H2B.

REFERENCES

- Allen, T. J. 1977. **Managing the flow of technology**. MIT Press, Cambridge.
- Baldwin, C. Y., & Clark, K. B. 2000. **Design Rules (vol. 1): The power of modularity**. MIT Press, Cambridge.
- Banker, R. D., & Slaughter, S. A. 2000. The Moderating Effects of Software Structure on Volatility and Complexity in Software Enhancement. **Information Systems Research**, 11(3): 219-240.
- Batagelj, V., & Mrvar, A. 1998. Pajek – Program for large network analysis. **Connections**, 21(2): 47-57.
- Beauchamp, M. A. 1965. An improved index of centrality. **Behavioral Science**, 10: 161-163.
- Bieman, J. M., & Ott, L. M. 1994. Measuring Functional Cohesion. **IEEE Trans. Software Eng.**, 20(8): 644-657.
- Borgatti, S. P., Carley, K. M. & Krackhardt, D. 2006. On the robustness of centrality measures under conditions of imperfect data. **Social Networks**, 28(2): 124-136.
- Brooks, F. P. (1995) **The mythical man-month**. Addison-Wesley.
- Burt, R. S. 1997. A note on social capital and network content. **Social Netw.**, 19(4): 355-373.
- Capra, E., Francalanci, C., & Merlo, F. “An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects. **IEEE Trans. Software Eng.**, 34(6), 765-782.
- Card, D. N., & Glass, R. L. 1990. **Measuring software design quality**. Prentice-Hall.
- Chiang, I. R., & Mookerjee, V. R. 2004. A Fault Threshold Policy to Manage Software Development Projects. **Information System Research**, 15(1): 3-21.
- Chidamber, S. & Kemerer, C. 1994. A metrics suite for object oriented design. **IEEE Trans. Software Eng.**, 20(6): 476–493.
- Choi, B., Raghu, T. S. & Vinze, A. 2006. An empirical study of standards development for E-Businesses: A social network perspective. **Proc. Hawaii Int’l Conf. System Sciences**, 139-148.
- Conway, M. E. 1968. How Do Committees invent? **Datamation**, 14(4): 28-31.
- Crowston, K., Wei, K., Li, Q., & Howison, J. 2006. Core and periphery in Free/Libre and Open Source software team communications. **Proc. Hawaii Int’l Conf. System Sciences**.
- Darcy, D.P., Kemerer, C. F., Slaughter, S. A., & Tomayko, J. E. 2005. The structural complexity of software: an experimental test. **IEEE Trans. Software Eng.**, 31(11): 982-995.
- De Souza, C., Froehlich, J., & Dourish, P. 2005. Seeking the source: software source code as a social and technical artifact. **Proc. Int’l ACM Conf. Supp. Group Work**, 197-206.
- Dijkstra, E.W. 1968. Letter to the editor: GOTO statement considered harmful. **Communications of the ACM**, 11(3): 147-148.

- Freeman, L. 1979. Centrality social networks: conceptual clarification. **Social Netw.**, 1:215-239.
- Gao, Y., Van Antwerp, M., Christley, S., & Madey, G. 2007. A Research Collaboratory for Open Source Software Research. **Proc. Int'l Wks. Emerging Trends in FLOSS Res. and Devel.**
- Ghezzi, C., Mandrioli, D., and Jazayeri, M. 2003. **Fundamentals of Software Engineering, Second Edition.** Pearson Education.
- Granger, C., & Newbold, P. 1977. **Forecasting Economic Time Series**, London Acad. Press.
- Halstead, M. H. 1977. **Elements of software science**, Elsevier.
- Henderson, R., & Clark, K. 1990. Architectural Innovation. **Admin. Sci. Quarterly.** 35(1): 9-30.
- Hossain, L., Wu, A. & Chung, K. K. S. 2006. Actor Centrality Correlates to Project Based Coordination. **Proc. Computer Supported Cooperative Work Conf.**, 363-372.
- Howison, J. & Crowston, K. 2004. The perils and pitfalls of mining SourceForge. **Proc. Int'l Workshop Mining Software Repositories**, 7-12.
- Howison, J., Inoue, K., & Crowston, K. 2006. Social dynamic of free and open source team communications, **Int'l Fed. for Information Processing Publications (IFIP)**, Springer-Verlag.
- Kemerer, C. F. & S. A. Slaughter, 1999. An Empirical Approach to Studying Software Evolution. **IEEE Trans. Software Eng.**, 25(4), 1-17.
- Kim, K. K. 1988. Organizational coordination and performance in hospital accounting information systems: an empirical investigation. **Accounting Review**, 5(3): 472-489.
- Kim, K. K., & Umanath, N. S. 1993. Structure and perceived effectiveness of software development subunits: a task contingency. **Jnl. of Mgmt. Inf. Sys.**, 9(3): 157-181.
- Knoke, D., & Yang, S. 2008. **Social Network Analysis, Second Edition.** SAGE Publications.
- Koch, S., & Schneider, G. 2002. Effort, co-operation and co-ordination in an open source software project: GNOME. **Information Systems Journal**, 12(1): 27-42.
- Kossinets, G., & Watts, D. J. 2006. Empirical analysis of an evolving social network. **Science**, 311(5757): 88-90.
- Kraut, R. E., & Streeter, L. A. 1995. Coordination in software development. **Communications of the ACM**, 38(3): 69-81.
- Lehman, M. M. 1980. Programs, life-cycles, and laws of software evolution. **Proc. IEEE Special Issue Software Eng.**, 68(9): 1060-1076.
- Malone, T. W., & Crowston, K. 1994. The interdisciplinary study of coordination. **ACM Computing Surveys**, 26(1): 87-119.
- MacCormack, A., Rusnak, J., & Baldwin, C.Y. 2006. Exploring the structure of complex software designs: An empirical study of Open Source and Proprietary code. **Management Science**, 52(7): 1050-1030.

- Mockus, A., Fielding, R., & Herbsleb, J. 2000. A case study of open source software development: the Apache server. **Proc. Int'l Conf. Software Eng.**, 263–272.
- Newman, M. E. J. 2005. A measure of betweenness centrality based on random walks. **Social Networks**, 27(1): 39-54.
- Papoulis, A., & Pillai, S. U. 2001. **Probability, random variables and stochastic processes**. McGraw-Hill.
- Parnas, D. L. 1972. On the criteria to be used in decomposing systems into modules. **Communications of the ACM**, 15(12): 153-158.
- Sabidussi, G. 1966. The centrality index of a graph. **Psychometrika**, 31(4): 581-603.
- Simon, H. A. 1962. The architecture of complexity. **Proc. American Philosophical Society**, 106(6): 467-482.
- Sims, C. A. 1980. Macroeconomics and reality. **Econometrica**, 48(1): 1-48.
- Sosa, M. E., Eppinger, S. D., & Rowles, C. M. 2004. The misalignment of product architecture and organizational structure in complex product development. **Management Science**, 50(12): 1674-1689.
- Sosa, M. E., Eppinger, S. D., & Rowles, C. M. 2003. Identifying modular and integrative systems and their impact on design team interactions. **Jnl. of Mech. Design**, 125(2): 240–252.
- Sparrowe, R. T., Liden, R. C., Wayne, S. J., and Kraimer, M. L. 2001. Social Networks and the Performance of Individuals and Groups. **The Academy of Management Jnl.**, 44(2): 316-325.
- Stevens, W., Myers, G., & Constantine, L. 1974. Structured Design. **IBM Sys. Jnl.**, 13: 115-139.
- Tervonen, I., & Kerola, P. 1998. Towards deeper co-understanding of software quality. **Information and Software Technology**, 39(14-15): 995-1003.
- Van de Ven, A. H., Delbecq, A. L., & Koenig, R. 1976. Determinants of coordination modes within organizations. **American Sociological Review**, 41: 322-338.
- Von Krogh, G., Spaeth, S. & Lakhani, K.R. 2003. Community, joining, and specialization in open source software innovation: a case study. **Research Policy**, 32(7): 1217-1241.
- Wasserman, S., & Faust, K. 1994. **Social network analysis: methods and applications**. Cambridge University Press.
- Xu, J., Christley, S., Madey, G. 2006. Application of the social network analysis to the study of open source software. In J. Bitzer & P. J. H. Schroder (eds.), **The economics of Open Source software development**, Elsevier B.V.
- Zmud, R.W. 1984. Design alternatives for organizing information systems activities. **MIS Quarterly**, 8: 79-93.