December 2007

Social Capital, Structural Holes and Team Composition: Collaborative Networks of the Open Source Software Community

Yong Tan
University of Washington

Vijay Mookerjee
University of Texas at Dallas

Param Singh
University of Washington

SOCIAL CAPITAL, STRUCTURAL HOLES AND TEAM COMPOSITION: COLLABORATIVE NETWORKS OF THE OPEN SOURCE SOFTWARE COMMUNITY

Param Vir Singh • Yong Tan

University of Washington Business School, Seattle, Washington 98195-3200

{psidhu • ytan}@u.washington.edu

Vijay S. Mookerjee

School of Management, University of Texas at Dallas, Richardson, Texas 75083-0688

vijaym@utdallas.edu

Abstract

OSS developers bring to a project more than the skills and knowledge acquired through years of experience. They bring with them the ability to acquire resources from their social networks. The skills and capabilities of the developers referred to as 'human capital' by economists are supplemented by the capabilities to access resources from the network referred to as 'social capital' by sociologists. Since, OSS is developed in an open and social environment, social capital drawn from the collaboration network of developers would play an important role in developer productivity. In this study, we investigate the influence of social capital of team members on the productivity of the OSS development team. Specifically, we investigate whether teams composed of developers who form cohesive ties in their social space or teams composed of developers who tap into different knowledge pools are more productive in terms of technical development and commercial success of the software they produce. To examine this issue we focus on three key elements of the team members' social network: direct ties, indirect ties and structural holes. Results from empirical analysis of OSS projects hosted at Sourceforge, suggest that both direct and indirect ties positively influence the productivity of teams. We further found that the greater the cohesive ties that the team members form in their social network the more productive they are.

Keywords: social capital, team design, structural holes, network cohesion, open source software

SOCIAL CAPITAL, STRUCTURAL HOLES AND TEAM COMPOSITION: COLLABORATIVE NETWORKS OF THE OPEN SOURCE SOFTWARE COMMUNITY

Introduction

Recent years have witnessed the growing popularity of the open source way of developing software. Some of the well known examples are Linux operating systems, Apache web servers, Perl, the Statistical programming language R and MySql databases. Major technological firms such as Sun Microsystems, IBM, Unisys, Novell and large venture capitalists are investing generously in open source communities (Shankland 2002). This community based model of software development has been so successful that the paradigm has extended into new arenas such as pharmaceutical development, knowledge repositories (e.g., Wikipedia), biological research, movie production, space exploration, and education as well as to unexpected domains such as developing recipes for open-cola, coffee and beer (Gatlin 2005, Goetz 2003). Open source software (OSS), however, still remains the most prominent face of the open source paradigm. OSS philosophy is based on principles and practices that promote the sharing of production and design processes and outcomes. The surprising success of OSS has radically changed our conception of how innovation should be managed and prompted calls for developing new models and theories to better understand innovation among distributed developer communities (Fleming and Waguespack 2007, von Hipple and von Krogh 2003).

OSS is developed by voluntary developers through individual incremental efforts and collaboration. New contributions to the code often involve, to a large extent, a recombination of known conceptual and physical materials (Fleming 2001, Narduzzo and Rossi 2003). Developers with better access to and familiarity of such materials are advantaged in their code development efforts. Because information about and knowledge of resources often lies spread across developers in the community, *social capital*, i.e. a developer's access to resources from a network of relationships, may emerge as a key factor that differentiates those who are more productive than others.

As developers work across several projects, they weave a network of relationships (collaboration networks) that act as channels which facilitate the flow of information and technical know-how. Our understanding of how collaboration networks affect productivity, however, remains unclear because the specific elements of network structure that influence team productivity have yet to be identified. Investigating this issue is important because it potentially has important implications for optimal team composition and output in software development.

Researchers in social capital theory typically focus on how social networks enhance the ability of team members to achieve team goals (Ancona and Cadwell 1992, Gargiulo and Benassi 2000, Reagans et al 2004). However, two contradicting views exist. Coleman (1988) argues that densely embedded networks with many connections between an ego's alters[1], i.e. *network cohesion/closures*, are facilitative to an ego. Network closure fosters group identity, solidarity and shared understanding among members, which in turn encourages knowledge sharing and ensures the fidelity of the shared information. A contrary view purported by Burt (1992), contends that the egos that connect otherwise disconnected alters can benefit from the access to diverse sources of information. Such network positions are called *structural holes* (Burt 1992, 2004).

Successful software development calls for efficient communication, coordination, resource pooling and knowledge sharing. Given the absence of formal management structures, the above attributes of success are especially important in OSS. The presence of *cohesive ties* among team members promotes trust and cooperation among members (Coleman 1988, 1990). Cohesive ties develop among a group of individuals through repeated cross-interactions over time. According to the cohesive ties argument, the developers in a team should form a single cohesive group. On the other hand, the structural holes view emphasizes a team member's abilities to access a wide range of information, resources or perspectives as a key factor for efficient team performance (Burt 1992, 1997, 2004). The structural holes theory emphasizes the virtues of diversity: teams should be composed of individuals that

[1] In social network analysis, the focal actor is termed 'ego'. The actors who have ties (relationships) to the ego are termed 'alters'.

can tap into different knowledge pools or resources. Diversity is beneficial because knowledge and skill sets are likely to be homogeneous within a cohesive group rather than across groups. Hence, teams should be composed of diverse individuals that can span across different groups.

In this study, we draw upon structural sociology research to examine the impact of three key structural elements of a social network – direct ties, indirect ties, and cohesive ties/ structural holes – on the productivity of developer teams. Direct ties, i.e. strong relationships that facilitate intense interactions, are conducive to the exchange of both tacit and explicit knowledge and, hence, allow large volumes of technical know-how and information to be shared (Kogut and Zander 1992, Ahuja 2000, von Hippel 1988). Indirect ties, i.e. arms-length relationships, typically do not provide opportunities for intense interaction and are conducive to the exchange of explicit knowledge and, hence, only allow information to be shared (Kogut and Zander 1992, von Hippel 1988, Wasserman and Faust 1994). Though direct ties provide richer knowledge benefits, they are costly to maintain. We argue that the existence of direct and indirect ties have a significant effect on the technical and commercial success of projects measured in terms of both the amount of code developed as well as the adoption of the software by users.

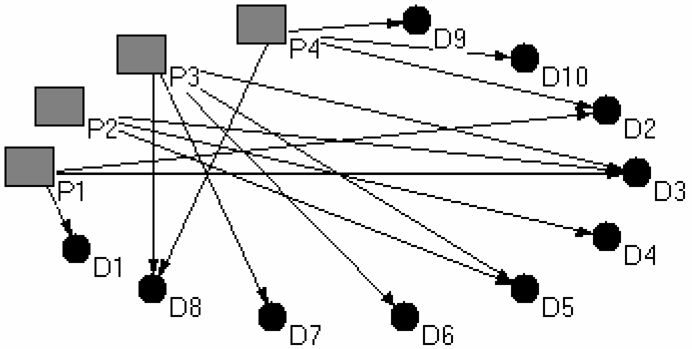
The elements of the network structure underlying the team provide important and, sometimes, contradictory managerial implications. The advantages and the costs associated with direct and indirect ties create obvious tradeoffs in the design of an optimal network structure. The network closure argument suggests that the optimal team is one with densely interconnected developers, whereas, the structural holes argument recommends that the optimal team is one that is composed of developers who can bring to the table diverse knowledge and resources by tapping into their local networks.

We test our hypothesis on detailed data collected from Sourceforge for PHP foundry. Our results suggest that both direct and indirect ties positively influence the productivity of teams. Moreover the greater the cohesive ties that the team members form in their social network the more productive they are.

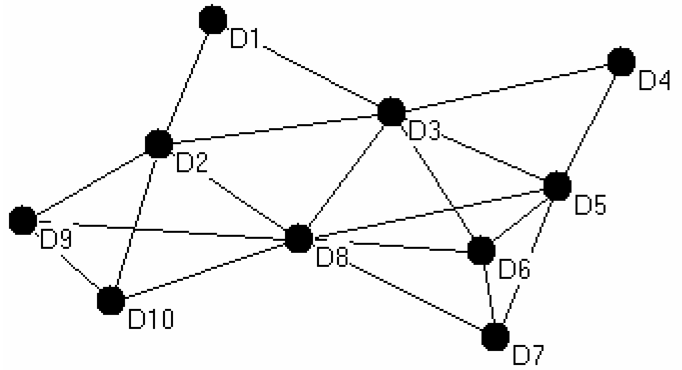
Developer Collaboration Network: Direct Ties, Indirect Ties and Structural Holes

OSS developers may work on several projects at the same time and, hence, belong to multiple projects. A relationship exists between two developers if they work together on a project. Similarly, a relationship between two projects exists if they share some developer(s). Such a scenario of relationships can be represented by an *affiliation* network. An affiliation network is a special kind of two mode social network that represents the affiliation of a set of actors with a set of social events (Wasserman and Faust 1994). Developers and the projects represent the two nodes of this network.

A social network can be represented as a graph where nodes represent actors and lines joining the nodes represent relationships. Graphically, the developer-project social network of an OSS community can be represented by a bipartite graph. Figure 1a shows a bipartite graph with 4 projects and 11 developers. A relationship between a project and a developer is shown by a line joining the two entities. A unipartite projection, i.e. the relationship among developers is shown in Figure 1b; all the developers who work on a same project are related to one another. Note that, the line between any two nodes indicates the presence of a relationship and not the quality of the relationship. As more developers join the community and new projects are started the network evolves into a giant mesh of relationships.



(a) Developer Project Affiliation Network. Developers are indicated by black spheres and projects are indicated by grey squares.



(b) Developer Network (Unipartite Projection from Affiliation Network)

Figure 1: Social Network of 4 projects and 11 developers.

Direct ties exist among developers who work together on a project. For example, Developers D1 and D3 share a direct relationship as both of them work on project P1. Indirect ties exist among developers who have no direct tie, but are *connected*; there exists a string of relationships through which they can reach one another. For example, an indirect relationship exists between developers D1 and D4. Note that D1 and D4 do not work together on any project but they can reach each other through developer D3. Developer D3 spans a structural hole as this developer connects developers (D1, D2) with (D4, D5) who would otherwise be disconnected. Network Closure or cohesiveness is measured by the absence of structural holes. For example, developers D1, D2 and D3 form a closed triad (i.e. network closure) as each one of them interact with the other two. Whereas, developers D2, D3 and D4 form an open triad (i.e. presence of structural hole and absence of network closure) as D2 and D4 interact only through D3.

OSS Development Process

OSS is developed by voluntary developers working in teams. Software is developed under a modular structure which allows developers to work in parallel (Baldwin and Clark 2006, MacCormack et al 2006, Weber 2004). Developers follow an adaptive process to solve technical problems. There are an infinite number of paths to solve a problem and choices among various alternatives involve high degree of experimentation which necessitates leeway and discretion among developers (Mockus et al 2002, Raymond 1998, Weber 2004).

New contributions to the code most often involve, to a substantial extent, a recombination of known conceptual and physical materials (Fleming 2001, Narduzzo and Rossi 2003). Rather than designing a software module from scratch, previously existing modular architecture is frequently inherited (Baldwin and Clark 2006, MacCormack et al 2006, Narduzzo and Rossi 2003). For example, the GNU and the FreeBSD projects closely resemble the UNIX architecture (Narduzzo and Rossi 2003). Software evolves from an initially integrated system by recombination of modules and evolutionary adaptation (Baldwin and Clark 2006, Fleming 2001, MacCormack et al 2006, Narduzzo and Rossi 2003). Developers exploit their access to broad range of technological solutions, by making analogies between current design problems, related existing solutions and failed approaches (Baldwin and Clark 2006, Fleming 2001, Hargadon and Sutton 1997).

Critical inputs into this process include access to and understanding of a variety of knowledge elements such as current design problems, related existing solutions, technical know-how, failed approaches, current market trends etc (Fleming 2000, Hargadon and Sutton 1997, Schilling and Phelps 2007, Schilling 2003). Teams composed of developers that have greater access to and understanding of these resources should be advantaged in their software development efforts. As an environment in which both technical and commercial success matters, developer effort is focused on achieving both of them.

Network Relationships and Knowledge Benefits

Relationships among developers in a network provide them with two types of knowledge benefits — resource sharing and knowledge spillovers (Ahuja 2000, Grewal et al 2006, Kogut and Zander 1992). Resource sharing allows them to combine know-how and physical assets whereas knowledge spillovers provide information about current design problems, failed approaches, new breakthroughs, and opportunities (Ahuja 2000, Fleming 2000, Kogut and Zander 1992). The subtle difference between know-how and information needs to be emphasized here. Know-how corresponds to the accumulated practical skill or expertise and involves a significant portion of tacit component (Ahuja 2000, von Hippel 1988). Information is primarily explicit knowledge that includes facts, propositions, and symbols that can be easily codified and transmitted without loss of integrity (Kogut and Zander 1992).

Network relationships provide developers with knowledge that helps them in solving technical problems as well as incorporating the features in the software that would make it more attractive to potential adopters. These relations also act as a source of bug reports/fixes, feature requests, user support and patch contributions (Grewal et al 2006, Lakhani and von Hippel 2002). Information may also be shared about organizational issues that may affect the reputation of the developers. For instance, information about opportunistic behavior of a member developer may be passed on into the network which may prohibit such behavior. Such information sharing may also promote richer contributions from developers who want to accrue reputation benefits by showing their programming prowess to others (Lerner and Tirole 2002, Roberts et al 2006).

Direct Ties

Developers who work together on a project share direct ties with each other. These ties provide opportunities for repeat, intense interactions. Hence, they are conducive for resource sharing as well as knowledge spillover. This provides a developer team of a project with benefits of knowledge sharing. Developers can spread the effort required to solve a problem over a large number of direct relations. Besides, the knowledge existing in the knowledge stocks of each project in which a developer is involved is available to him. This also leads to a richer knowledge stock of know-how, existing solutions, design problems, failed approaches and new breakthroughs for the developer (Gomes-Casseres, Hagedoorn and Jaffe, 2006; Mowery et al. 1996). The knowledge acquisition depends on knowing who knows what and developers with large number of direct ties are likely to be privy to such information (Borgatti and Cross 2003).

Hypothesis 1: *The number of direct ties that a team maintains will positively influence its subsequent success.*

Indirect Ties

Indirect relationships (where two developers do not work together but can be reached through mutual acquaintances) are less likely to provide opportunities for repeat interactions and, hence, are not conducive to resource sharing. However, knowledge spillovers do not require repeat interactions and, hence, indirect relationships will be conducive to them.

Developers in a relationship also bring with them the knowledge and experience from their interactions with other partners (Gulati and Gargiulo 1999). Hence, a developer's relationship with another developer provides it with access to not just its own partners but to its partner's partners (Ahuja 2000, Gulati and Gargiulo 1999, Owen-Smith & Powell, 2004). Developers with higher number of indirect ties will have access to a larger variety of knowledge resources. Information is susceptible to decay during transmission over long distances (Bartlett 1932). An indirect ties' measure accounts for the distance between indirectly connected developers. A high distance weighted indirect ties measure for developers in a team ensures the integrity of the received information besides increasing the speed of transmission. Hence, larger number of indirect relationships for developers in a team may influence its productivity by providing it reliable and speedy access to a wide variety of knowledge resources.

Hypothesis 2: *The number of indirect ties that a team has will positively influence its subsequent success.*

Direct X Indirect Ties

The knowledge advantages that a developer may receive from an indirect relationship are likely to be contingent on the number of direct ties he has. Specifically, the relative knowledge addition from indirect ties to a developer with few direct ties is going to be high compared to that for a developer with many direct ties.

Hypothesis 3: *The influence of the indirect ties that a team has on the subsequent success will be negatively moderated by the number of its direct ties.*

Network Cohesion vs Structural Holes

Cohesiveness means that ties are redundant to the degree that they lead back to the same actors. Such redundancy increases the information transmission capacity in a group of developers having cohesive ties. It promotes sharing and makes information exchange speedy, reliable and effective. Information between two developers in a cohesive group can flow through multiple pathways; this increases the speed as well as reliability of information transfer. Multiple path ways ensure that the information once introduced in the group quickly reaches all the members (Schillings and Phelps 2007). Developers can compare information received through multiple sources for integrity (Coleman 1988).

Cohesiveness in the group gives rise to trust, reciprocity norms, and a shared identity, all of which leads to a high level of cooperation and can facilitate collaboration by providing self-enforcing informal governance mechanisms (Coleman 1988, Granovetter 1972, Schillings and Phelps 2007). Thus, cohesive ties enable richer and greater amounts of information and knowledge to be reliably exchanged.

The groups also provide meaningful context for information and resource sharing. The trust among the members in the group affords them to be creative. This creativity helps in coming up with alternative interpretation of current problems, or novel approaches to solve these problems (Powell & Smith-Doerr 1994). Team members belonging to a same cohesive group in social space can easily develop shared understanding of problems and solutions which greatly facilitates communication and further learning (Brown & Duguid 1991).

However, cohesion can also lead to norms of adhering to established standards and conventions, which can potentially stifle innovation (Burt 1992, 1997, 2004, Schillings and Phelps 2007, Uzzi & Spiro 2005). The standards, conventions and knowledge stocks vary across groups (Hargadon and Sutton 1997). Structural holes are the gaps in the information flow between these groups. Developers who connect different groups are said to fill these structural holes. Teams composed of developers who span different groups may offer several advantages. First, the technical or organizational problems and difficulties of a developer in one group can be easily and reliably relayed to developers in other groups (Burt 1992). The solutions for these problems may be obvious to someone and would be quickly and reliably relayed back. Second, each group has its own *best practices* (organizational or technical perspectives) which may have value for other groups. The developers who connect these groups can see how resources or practices in one may create value for other and synthesize, translate as well as transfer them across groups (Burt 1992, 2004). Third, Resource pooling across groups provide developers opportunities to work on different but related problem domains, which may help them in developing a better understanding of their own problems (Schilling et al 2003).

These two arguments promote two contradicting predictions. The network cohesion argument predicts that teams formed of developers who share cohesive ties will be more successful due to better coordination and communication resulting from increased information transmission capacity. The structural holes argument predicts that teams formed of developers who span structural holes will be more successful due to access to wider range of knowledge resources.

Hypothesis 4a: *The more the structural holes spanned by a team's network the greater the team's subsequent success.*

Hypothesis 4b: *The more the structural holes spanned by a team's network the less the team's subsequent success.*

Methods

To test our hypothesis, we collect data from Sourceforge.net which is the primary hosting place for OSS projects and accounts for about 90% of all OSS projects. Though not all OSS projects are hosted at Sourceforge, it has been argued that it is the most representative of the OSS movement, in part because of its popularity and the large number of developers and projects registered there (Grewal et al 2006, Madey et al 2002, Xu et al 2005). Researchers interested in investigating issues related to OSS phenomenon have predominantly used Sourceforge data (Grewal et al 2006, Madey et al 2002, von Hippel and von Krogh 2003, Xu et al 2005). Sourceforge provides web space as well as services such as mailing facility, discussion forums, CVS repository hosting, and download servers to OSS projects to organize and coordinate the software development. Data regarding projects and developers associated with each project was obtained from the Sourceforge database[2]. This data was supplemented with the data about downloads and code development from the project homepages at the Sourceforge.

Construction of Collaboration Network of Developers

Social network research has suggested two techniques for collecting network data: Egocentric and Whole network. In the present scenario the ego centric measure would require us to pre-select a random set of project/developers and then snowball the developer relationships to include new developers (Burt 2002, Grewal et al 2006, Wasserman and Faust 1994). This snowball process goes on till the data set involves a large enough set of developers for analysis. However, this approach can cause inaccuracies into the network data as developers or projects that do not have many relationships are less likely to be reached through this technique. Whole network studies examine actors that are regarded for analytical purposes as bounded social collectives (Marsden 2005). In this case, defining an appropriate boundary around the network, the set of individuals who are interconnected is critical (Laumann et al 1983). This is the predominant approach used in situation where an appropriate network boundary can be established (for example, interfirm alliance networks (Ahuja 2000, Gulati 1999, Schillings and Phelps 2007), regional innovation networks (Fleming et al 2007), broadway artist collaboration networks (Uzzi and Spiro 2005)). We follow this approach and use participation in PHP foundry (broad technological platform) as the network boundary. The choice of using participation in a foundry as a network boundary is particularly important and acceptable for two reasons. First, a foundry represents a focused technology platform, and hence, more efficient knowledge sharing would be possible within a foundry rather than across foundries. Foundry as a network boundary for OSS projects at Sourceforge has been used in related research (Grewal et al 2006). Second, we analyzed memberships for 5000 randomly selected developers who work on multiple projects and found that only approx 4% of those worked across two or more foundries.

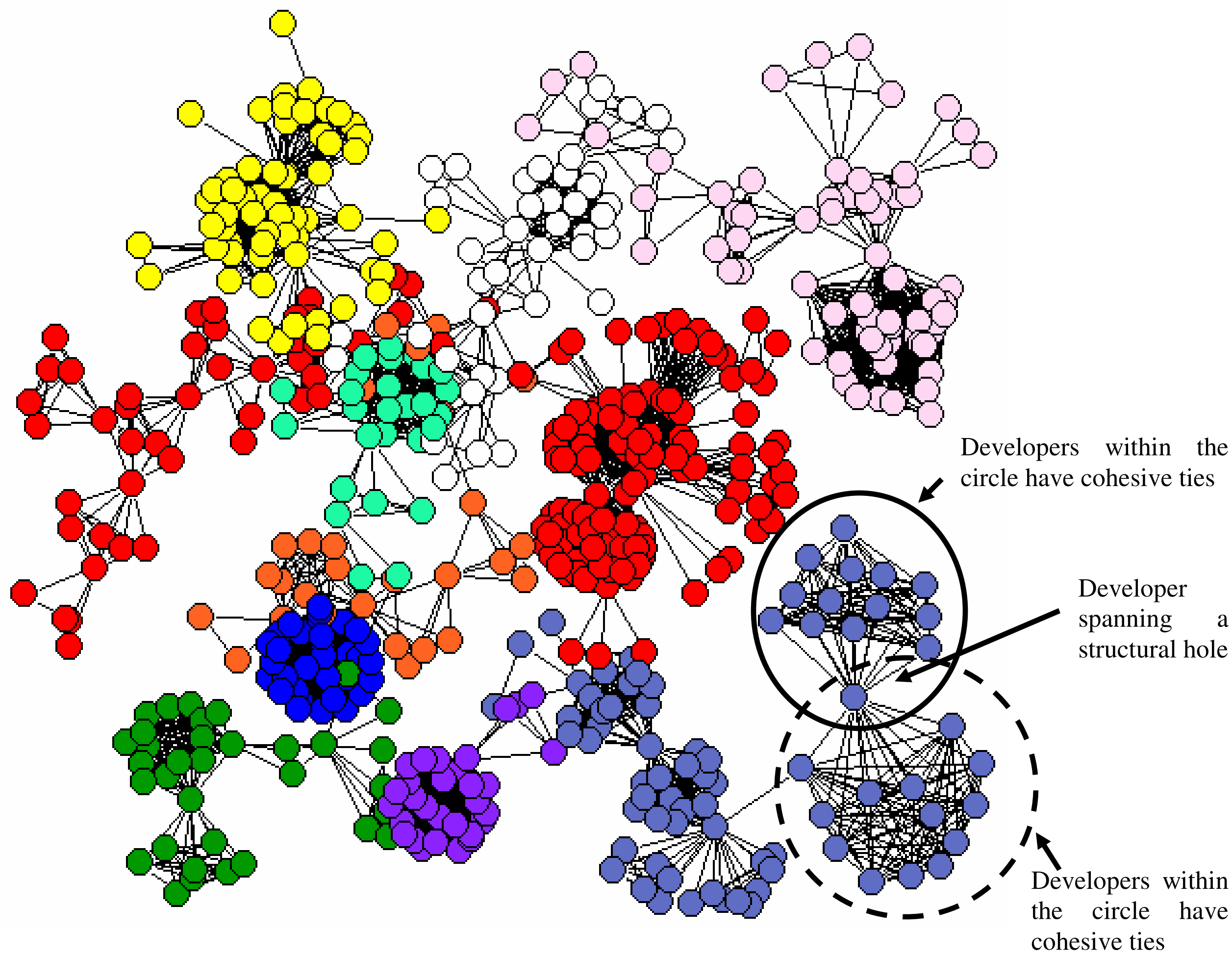
Developer and project affiliation data was collected for projects that were registered from Nov 1999 (Sourceforge's inception date) to Nov 2004 at Sourceforge from the Sourceforge database. Information in the Sourceforge database is available on a roughly monthly basis starting Nov 2004. The only prior snapshot of the database is available for Nov 2002. We chose November 2004 data for construction of the network as Sourceforge had enough mass of registered developers and projects in each foundry for social capital to be a significant resource. All projects within the PHP foundry were selected along with the associated developers. An affiliation network was constructed for the foundry.

Each affiliation network was used to create unipartite projection for the developer network. From unipartite projection, an undirected binary adjacency matrix was developed. An adjacency matrix represents the relationships between any two developers in the network. The row and the column headings represent the developers. A value of one corresponding to two developers in the matrix indicates a presence of a relationship between them and a value of zero indicates the absence of such. The adjacency matrix is undirected as relationship among two developers is mutual. For each network, measures were calculated from the undirected binary adjacency matrices. Social network software package UCINET[3] was used to calculate these measures.

There are a total of 5191 projects and 10973 developers. A component in a network is composed of all developers who can reach each other through strings of relationships, i.e. any two developers in the component have a finite path length between them. There are 8521 developers who belong to a component of size 2 or more. There are 2452 developers which do not work with any other developer. The size of the largest component for the developer network was 1902. A partial view of the developer collaboration network is shown in Figure 2.

[2] https://zerlot.cse.nd.edu/mywiki/index.php?title=Main_Page

[3] http://www.analytictech.com/



Developers within the circle have cohesive ties

Developer spanning a structural hole

Developers within the circle have cohesive ties

Figure 2: Partial Developer Network for the PHP foundry. Developers are represented by spheres and color coded by component. An arc joining two developers indicates the existence of a direct tie between them. An example of a structural hole and cohesive ties is also shown. The developer indicated spanning the structural hole connects the two groups of developers which have cohesive ties within each group. Project teams that would include the indicated developer will have direct access to the knowledge resources of the two groups.

Measures

Dependent Variable: CVS commits and Downloads

To operationalize project success, we use two different measures proposed in literature: number of CVS commits and number of downloads (Crowston et al 2003, Grewal et al 2006). CVS commits represent technical success whereas downloads represent commercial or economic success (Grewal et al 2006). Similar measures have been used in a related study by Grewal et al (2006) who investigate the impact of network embeddedness on project success. These measures are consistent with the literature on Information System success (DeLone and McLean 1992), as each of these measures either indicates the technical achievement or user adoption or interest.

Software developers use Concurrent Versioning System (CVS) to manage software development process. CVS stores the current version(s) of the project and its history. A developer can check-out the complete copy of the code, work on this copy and then check in her changes. The modifications are peer reviewed. The CVS updates the modified file automatically and registers it as a *commit*. CVS keeps track of what change was made, who made the change, and when the change was made. This information can be gathered from the log files of the CVS repository of a project. CVS commits provide a measure of novel invention that is externally validated by peers. As CVS commits signify only meaningful changes to the code (Crowston et al 2003, Grewal et al 2006), we use number of CVS commits as measure of technical success. We measure the dependent variable, *CVSDiffit,* as the number of CVS commits for project *i* in *t* years since Nov 2004. For instance, for *t* = 1, we count the number of CVS commits for a project from Nov 2004 to Nov 2005.

To operationalize commercial or economic success of the project, we use the number of downloads. The download servers at Sourceforge act as the sole source of software downloads for the hosted projects (Crowston et al 2003, Grewal et al 2006). In such a scenario, the number of downloads may act as a measure of popularity for the project. This also represents external validity of the success of the projects from a user's perspective. We measure the dependent variable, *DownDiffit*, as the number of downloads for project *i* in *t* years since Nov 2004.

The relationships among developers, once established, at Sourceforge stay for a long time as— (1) OSS projects do not have a fixed deadline and are in a continuous development state (Weber 2004) and (2) the associated developers at Sourceforge rarely drop out (Crowston et al 2003). Benefits from a relationship can be availed only after the relationships have been established. That calls for the use of lag specification between network structure and success measures as constructed earlier (Ahuja 2000, Fleming et al 2007, Gulati and Gargiulo 1999, Schillings and Phelps 2007).

Independent Variables

Direct Ties:

We count the number of direct ties for each developer from the adjacency matrix. For projects having more than one developer we take an average of the direct ties for the associated developers.

Indirect Ties:

A direct count of the number of indirect ties is inappropriate as the tie strength weakens with the increase in distance between two developers. To account for this decay while calculating the indirect ties measure, we use the frequency decay function proposed by Burt (1992). It is developed under the argument that the rate at which the strength of the relation decreases with the increasing length of its corresponding path distance should vary with the social structure in which it occurs (Ibid). This decay function for developer *i* is given as:

$$z_{ij} = 1 - f_{ij} / (N_i + 1),$$

where f_{ij} is the number of developers that *i* can reach within and including path length *j*, and N_i is the total number of developers that *i* can reach in the network. Then z_{ij} is the decay associated with the information that is received from developers at path length *j*. The indirect ties measure for developer *i* is then calculated as:

$$\text{Indirect Ties}_i = \sum_{j=2}^{n} z_{ij} w_{ij} ,$$

where n is the total number of developers in the network and w_{ij} is the number of developers that lie at a path length of *j* from *i*. For example, assume developer i can reach a total of 6 developers: 3 lie at path distance 2, 1 at path distance 3 and 2 at path distance 4. Then, $\text{Indirect Ties}_i = (1-3/7)\cdot 3 + (1-4/7)\cdot 1 + (1-6/7)\cdot 2 = 17/7$. The larger the group over which a developer has to devote its time and energy, the weaker the relationship that it can sustain with any one member of the group, and the stronger the relationship with the closer ones. For projects having more than one developer we take an average of the indirect ties for the associated developers.

Network Cohesion or Structural Holes:

Following Burt (1992), our measure of Network Cohesion is indirect structural constraint. This measure is computed as:

$$\text{Constraint}_i = \left(\sum_j \sum_q p_{iq} p_{qj}\right) / M_i, \quad j \neq q\,,$$

where p_{iq} is the proportion of *i*'s relations invested in the relationship with *q*, and p_{qj} is the proportion of *q's* relations invested in *j*. M_i represents the number of direct ties for developer *i*. Constraint measures the extent to which ego has ties with few alters and those alters have ties between them (Burt 1992). The higher values of constraint for developer *i* indicate network closure or social cohesion around him and the lower values indicate the presence of structural holes or brokerage opportunities. For projects having more than one developer we take an average of the constraint for the associated developers.

Control Variables:

Team Network

To control for a team's ability to be central to the flow of information and resources in the network, we computed the *betweenness centrality* of a team by calculating the betweenness centralities of each associated developer and then taking an average of it (Wasserman and Faust 1994). Team with a high betweenness centrality score lie in the shortest path of information flow between a numbers of other developers. Such a team can exert control over the flow of information among others (Grewal et al 2005).

The connectivity of alters may influence the amount of information and resources that an ego can draw upon in the network. Relationships with teams that are highly connected in the network are likely to be more beneficial than with teams which are less highly connected (Grewal et al 2005, Gulati 1999). To control for this effect, we computed the *eigen-value centrality* for each developer on a team and then took its average for the team.

Team Human Capital and Ability

We included the number of developers (*Developers*) associated with a project to account for human capital actively involved in the project. To gauge the past ability of the team, we calculated pre-sample outcome variables: pre-sample downloads and pre sample CVS commits by measuring their cumulative values for each team by November 2004. This specification follows the pre-sample information approach (Blundell et al 1995) to control for different knowledge stocks of the teams at the time of production and other historical factors that cause current differences in the dependent variables which are difficult to control for in cross-sectional regressions (Wooldridge 2006).

User Input and Market Potential

Though all the teams belong to the same foundry and the characteristics identified by Cooper (2000) for new product success may be similar across them, the software produced by the team may differ in its market potential and extent of user participation. Users play a critical role in the evolution of an OSS product (von Hipple and von Krogh 2003). Activities such as bug reports, bug fixes, and user-support are user driven activities. We control for these activities by construction two variables *Support* as the cumulative number of support requests answered by November 2004 and *Bugs* as the cumulative number of bugs closed by November 2004. We follow Grewal et al (2005) and construct a variable *Page Views* as the cumulative number of project pages viewed to control for the market potential and general interest in the project.

Project Age

We control for the age of the project by calculating a *Project Age* variable as the number of months since a project's inception at Sourceforge by November 2004. To control for potential non linear effect of project age on the dependent variables, we also incorporate a square of the *Project Age* in the model.

Project Characteristics

We constructed an extensive range of variables to account for the project level heterogeneity on the dependent variables. A team is required to report the project's type, intended audience, language and user interface at Sourceforge. We control for all these variables by constructing dummies variables. The type of the project indicates about the potential market size of the software. We control for 14 different types of the projects ranging from games to education to software utilities. Intended audience may influence the quality of developers that are attracted towards a project. For example, software that is aimed towards system administrators is likely to attract more sophisticated developers (Lerner and Tirole 2002, Roberts et al 2006). We control for this affect through constructing dummies for system administrators, developers and end users. Project's whose language is not English restrict both the number of users and developers that can participate in it. To control this, we construct a dummy variable for English. User interface may also have an influence on the market size of a project. For example, software with graphical user interface is easier to use and is likely to be adopted more widely. We control for these effects by controlling for four different types of user interface through dummy variables.

Model Specifications

Though both our dependent variables are count variable, both of them are heavily skewed. We take a logarithmic transformation and treat them as continuous variables. The direct ties are very highly correlated with the interaction term between direct and indirect ties. We mean centered both direct and indirect ties and then calculated their interaction which reduced the correlation between these variables significantly (Jaccard and Turrisi 2003). Project Age and its square are also highly correlated; hence we mean centered project age and then calculated its square. Some of the variables, Support, Bugs, Developer, Pre-sample CVS, Pre-sample Download are heavily skewed, so we performed a logarithmic transformation (Gelman and hill 2007). The parameters were estimated by Generalized Linear Regression framework and the Huber White standard errors which adjust for heteroskedasticity and autocorrelation were calculated. These estimated parameters with their robust standard errors are reported in Table 1. To check for robustness of our results, we ran several models with and without control variables. One concern about our model could be specification of the 1 year and 2 year lag structure. We performed the analysis for different lag specifications to test for any misspecification. We found the results to be quite robust to different lags specifications. To check for possible bias caused by outliers, we re-performed the analysis after removing the outliers. Our results were found to be quite insensitive to these outliers. Another, concern could be our transformation of the dependent variables and their use as continuous variables. We performed the analysis on with dependent variables as count variables. Since, both of them showed high dispersion, we estimated the parameters using a Negative Binomial specification. The model for CVS Diff did not produce qualitatively different results. As a Negative Binomial model controls for unobserved heterogeneity through the conditional mean specification, it provides a strong support for our results. The results for this model are available from the author upon request. However, the procedure did not converge for the model for Down Diff.

Table 1: Generalized Linear Regression Results

Dependent Variable	**Down Diff 2**		**Down Diff 1**		**CVS Diff 2**		**CVS Diff 1**	
Intercept	4.930***	4.930***	4.060***	4.070***	6.221***	6.230***	4.830***	4.840***
	(0.10)	(0.105)	(0.105)	(0.106)	(0.022)	(0.022)	(0.029)	(0.029)
Direct Ties	0.546**	0.546**	0.431***	0.476***	0.995***	1.080***	1.305***	1.390***
	(0.219)	0.0219	(0.124)	(0.220)	(0.023)	(0.040)	(0.030)	(0.050)
Indirect Ties	0.447*	0.447**	0.550**	0.668**	2.570***	2.640***	3.373***	3.443***
	(0.233)	0.203	(0.266)	(0.305)	(0.049)	(0.056)	(0.064)	(0.074)
Direct X Indirect Ties		-1.170		-1.510		-0.944***		-0.995**
		(1.900)		(1.910)		(0.350)		(0.454)
Constraint	0.237***	0.237***	0.243***	0.239***	0.153***	0.150***	0.321***	0.317***
	(0.055)	(0.055)	(0.055)	(0.055)	(0.010)	(0.010)	(0.014)	(0.014)
Eigenvalue Centrality	-0.072	-0.072	-0.0713 ·	0.062	0.011	0.074	0.013	0.020
	(0.055)	(0.055)	(0.054)	(0.055)	(0.011)	(0.011)	(0.014)	(0.014)
Betweenness Centrality	89.0	89.0	98.2	105	-12.1	-6.17	-22.2	-24.5
	(138)	(138)	(139)	(139)	(31.1)	(31.2)	(40.5)	(40.5)
Pre-Sample Downloads	0.407***	0.407***	0.372***	0.375***				
	(0.012)	(0.012)	(0.018)	(0.018)				
Pre-Sample CVS					0.132***	0.132***	0.131***	0.128***
					(0.003)	(0.003)	(0.003)	(0.003)
Support Answered	0.241***	0.241***	0.283***	0.283***	-0.006	-0.006	-0.020	-0.022
	(0.046)	(0.046)	(0.047)	(0.047)	(0.008)	(0.009)	(0.013)	(0.015)
Bugs Closed	0.359***	0.358***	0.384***	0.384***	0.012**	0.013**	0.021***	0.022***
	(0.025)	(0.025)	(0.026)	(0.026)	(0.005)	(0.005)	(0.006)	(0.006)
Age	-0.070***	-0.070***	-0.067***	-0.067***	-0.006***	-0.006***	-0.011***	-0.011***
	(0.003)	(0.003)	(0.003)	(0.003)	(0.001)	(0.001)	(0.001)	(0.001)
Age-Square	0.001***	0.001***	0.0004***	0.0004***	0.001***	0.001***	0.002***	0.002***
	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)
Developer	0.000	0.000	-0.024	-0.021	0.057***	0.059***	0.0738***	0.076***
	(0.042)	(0.042)	(0.043)	(0.043)	(0.008)	(0.008)	(0.010)	(0.010)
Project Level Controls								
Type	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Intended Audience	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
User Interface	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Translation	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Null Deviance (df)	19006	19006	20244	18591	1341.14	1341.14	2260.01	2260.01
	(5190)	(5190)	(5190)	(5190)	(3262)	(3262)	(3262)	(3262)
Residual Deviance (df)	12057	12056	12128	12128	193.59	186.15	323.61	316.12
	(5161)	(5161)	(5161)	(5161)	(3232)	(3232)	(3232)	(3232)

*** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$ (two tailed *t*–test for all variables)

The direct ties and the indirect ties were scaled down by a factor of 1000 after mean centering them. The dataset for the technical success of the project only includes 3263 projects as remaining projects were not hosting CVS repositories on Sourceforge.

Results

Direct, Indirect Ties and Direct x *Indirect:*

To understand the true impact of direct and indirect ties we need to understand the structure of these variables along with their interaction term. The direct and indirect ties are both mean-centered. The models involve an interaction term between these variables; hence the estimated parameters corresponding to these variables in Table 1 may not represent their true effects. For Down Diff the interaction term is insignificant for both the lag specifications. The likelihood ratio test showed that the models without the interaction term fit the data better than the ones wit it for the Down Diff dependent variable. Hence, we can focus on the results for Down Diff 1 or 2 without the interaction term. Since, there is no interaction term; the parameters corresponding to these variables represent their true effect. The direct ties and the indirect ties have a positive and significant effect on the commercial success of the project. This supports our hypothesis one and two. However, we the interaction term Direct X Indirect is insignificant for the commercial success of the project. Hence, we do not find support for hypothesis three for the commercial success.

For the technical success, we find that the interaction term is negative and significant. The associated parameters for direct and indirect ties do not represent their true effects. Consider CVS Diff 01 model with interaction term. A unit change in direct ties causes a change of 1.390 -0.995*Indirect in the dependent variable. This implies an effect of 1.390 – 0.995*(±0.203) = (1.19 to 1.59). Here, 0.203 is the standard deviation of indirect ties. This effect is positive and decreasing in the value of indirect ties. This supports our hypothesis one and three. Similarly the effect of unit change in indirect ties causes a change of 3.443 – 0.995*(±0.092) = (3.35 to 3.53). Here, 0.092 represents the standard deviation of direct ties. Here also the indirect ties have a positive impact on the technical success of the project and this effect is decreasing in increasing values of direct ties. This supports our hypothesis 2 and 3.

Network Cohesion versus Structural Holes

The results indicate that across all models the constraint variable has a positive and significant impact on the technical and commercial success of the project. Since, constraints represents the absence of a structural hole, this provides evidence that teams formed of developers who share cohesive ties in their social space perform better than other teams. This result is a bit surprising as related research in organizational innovation finds contradictory evidence. However, there is one critical difference between proprietary organizational innovation and open source innovation which may explain this result. In organizational networks the relationships among organizations or teams act as closed conduits characterized by legal arrangements (e.g. non disclosure agreements) designed to ensure that only the specific parties to a given connection benefit from the information that is exchanged (Owen-Smith and Powell 2004). This implies that proprietary conduits or strong connection (direct ties) are the primary source of competitive advantage. In that case, the more direct access to variety of resources provides larger knowledge benefits as the indirect relations are not very beneficial. However, the open source philosophy is based on the principles of sharing and un-hindered access to intellectual properties. Here, the developers inherently believe in sharing and the developers who connect different knowledge groups facilitate the unadulterated flow of information, resources and knowledge across groups. In contrast to the proprietary conduits, the relationships among developers in OSS represent channels that diffusely and imperfectly transfer flow between developers and facilitate knowledge and information spillovers. Here, the developers who are in a unique position to access variety of resources would not take brokerage benefits but would incur costs of transferring the information and technical know-how across groups. Such developers would be beneficial more to the over all community than to the individual team they belong to. Benefits from team members being cohesive in social space still exist due to its effect on shared understanding and coordination among members.

Conclusions Limitations and Future Research Directions

The primary question address in this study deals with successful team compositions in open innovation environment such as OSS. We found that teams composed of individuals who share cohesive ties in social space are more success than teams that teams composed of individuals who do not share such ties. This finding though contradictory to the organizational innovation literature, is consistent with recent studies in OSS which suggests that effective norms are critical for team effectiveness. Cohesive ties promote a normative environment that facilitates trust and cooperation among members. Given that cohesiveness among members makes team more effective, managerial attention should focus on encouraging interactions and exchanges among existing and new members which would lead to future cohesive ties.

We do not want to overstate our results; there are several limitations. The first limitation is due to the bipartite nature of our data. We followed the social networks literature, in using the unipartite projection of at the individual level from the affiliation network (Grewal et al 2006, Uzzi and Spiro 2005). However, this raises several concerns. Our dependent variables are at the team level, however the network we deal with is at the individual level. The measures for the team were constructed y average the network measures for associated developers. Simple arithmetic average may lead to significant loss of information. We are trying to address this issue in future research by doing a multi-level analysis in which we focus on the developer level outcome as well as team level outcome. Second, the dataset for the technical success is significantly reduced due to the unavailability of the CVS data. This may cause sample selection issues. We did preliminary analysis by applying Heckman's correction for sample selection. We used all the other variables except CVS related in the first stage selection model for the whole dataset. The results from the two stage model did not alter our results. Third, due to the affiliation nature of our data, we have to assume that all the developers within a team are related to each other. This may not be true and the internal relationship structure of the team may influence the effectiveness of the team. However due to the structure of available data we can not investigate this issue.

This research also raises several interesting issues. One, if teams composed of cohesive developers are more effective, this implies within team interactions are more important. It would be interesting to investigate how a developer's knowledge, skills and understandings do within team interactions influence. At the team level the structural holes were found to be not effective, but the ability of structural holes to act as pathways for diffusion of new knowledge is well accepted. It would be interesting to see if the presence of structural holes in the whole network influence the amount and variety of knowledge shared across the network and hence its influence on the productivity of all the teams in the network. All these questions represent an exciting area for future research

References:

Ahuja, G. "Collaboration Networks, Structural Holes, and Innovation," *Administrative Science Quarterly* (45), 2000, pp 425–455.

Ancona, D.G., and Caldwell, D.F. "Bridging the Boundary: External Activity and Performance in Organizational Teams," *Administrative Science Quarterly* (37), 1992, pp 634–665.

Baldwin, C.Y., and Clark, K.B. "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?" *Management Science* (52:7), 2006, pp 1116-1127.

Bartlett, F. C. *Remembering: A Study in Experimental and Social Psychology*. London: Cambridge University Press, 1990.

Blaauw, G.A., and Brooks, Jr., F.P. *Computer Architecture: Concepts and Evolution*. Reading, MA: Addison-Wesley, 1997.

Blundell, R.R., Griffith, R., and Van Reenen, J. "Dynamic Count Data Models of Technological Innovation," *The Economic Journal* (105), 1995, pp 333–344.

Burt, R.S. *Structural Holes*. Cambridge, Mass.: Harvard University Press, 1992.

Burt, R.S. "The Contingent Value of Social Capital," *Administrative Science Quarterly* (42), 1997, pp 339–365.

Burt, R.S. "Structural Holes and Good Ideas," *American Journal of Sociology* (110), 2004, pp 349–399.

Coleman, J. "Social Capital in the Creation of Human Capital," *American Journal of Sociology* (94), 1988, pp S95–S120.

Coleman, J.S. *Foundations of Social Theory*. Cambridge, Mass.: Harvard University Press, 1990.

Cooper, R.G. Winning at New Products: Accelerating the Process from Idea to Launch. Persues Publishing. Boulder, CO, 2001.

Crowston, K., Annabi, H. and Howison, J. "Defining Open Source Software Project Success," *Proceedings of ICIS*, Seattle, WA, 2003.

DeLone, W. H., and McLean, E.R. "Information systems success: The quest for the dependent variable," *Information Systems Research* (30), 1992, pp 60–95.

Fleming, L. "Recombinant Uncertainty in Technological Search," *Management Science* (47:1), 2001, pp 117–132.

Fleming, L., and Waguespack, D.M., “Brokerage, Boundary Spanning, and Leadership in Open Innovation Communities,” *Organization Science* (18:2), 2007, pp 165–180.

Gatlin, G. “Web Power to the People,” *Boston Herald.com*, September 29, 2005

Gargiulo, M., and Benassi, M. “Trapped in Your Own Net? Network Cohesion, Structural Holes, and the Adaptation of Social Capital,” *Organization Science* (11:2), 2000, pp 183-196.

Goetz, T. "Open Source Everywhere," *Wired* (11:11), 2003.

Gomes-Casseres, B., Hagedoorn, J., and Jaffe, A.B. “Do alliances promote knowledge flows?,” *Journal of Financial Economics* (80:1), 2006, pp 5-33.

Granovetter, M. “The Strength of Weak Ties,” *American Journal of Sociology* (78:6), 1973, pp 1360–1380.

Grewal, R., Lilien, G.L., and Mallapragada, G. “Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems,” *Management Science* (52:7), 2006, pp 1043–1056.

Gulati, R. “Alliances and Network,” *Strategic Management Journal* (19:4), 1998, pp 293–317.

Gulati, R., and Gargiulo, M. “Where Do Interorganizational Networks Come From?” *American Journal of Sociology* (104:5), 1999, pp 1439–1493.

Hargadon, A., and Sutton, R.I. “Technology Brokering and Innovation in a Product Development Firm,” *Administrative Science Quarterly* (42), 1997, pp 716–749.

Kogut, B., and Walker, G. “The Small World of German Corporate Networks in Global Economy,” *American Sociological Review* (66), 2001, pp 317–335.

Kogut, B. and Zander, U. “Knowledge of the firm, combinative capabilities, and the replication of technology,” *Organization Science* (3), 1992, pp 383-397.

Krishnamurthy, S. “Cave or Community: An Empirical Examination of 100 Mature Open Source Projects,” *First Monday* (7:2), 2002.

Kuk, G. “Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List,” *Management Science* (52), 2006, pp 1031–1042.

Laumann, E. O., Marsden, P. V., and Prensky, D. “The Boundary Specification Problem in Network Analysis,” In R. S. Burt, & M. J. Minor (Eds.), *Applied Network Analysis*: 18-34. Beverly Hills: Sage, 1983.

Lerner, J., and Tirole, J. “Some Simple Economics of Open Source,” *Journal of Industrial Economics* (46:2), 2002, pp 125–156.

MacCormack, A., Rusnak, J., and Baldwin, C.Y. “Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Prop Code,” *Management Science* (52:7), 2006, pp 1015–1030.

Madey, G., Freeh, V., and Tynan, R. “The open source software development phenomenon: An analysis based on social network theory,” *Proceedings of the 8th AMCIS,* 2002.

Marsden, P. V. “Recent Developments in Network Measurement,” In P. J. Carrington, J. Scott, & S. Wasserman (Eds.), *Models and Methods in Social Network Analysis*: 8-30. Cambridge: Cambridge University Press, 2005.

Mockus A., Fielding, R., and Herbsleb, J. “Two Case Studies of Open Source Software Development: Apache and Mozilla,” *ACM Trans on Software Engineering and Methodology* (11:3), 2002, pp 309–346.

Mowery, D.C., Oxley, J.E., and Silverman, B.S. “Strategic Alliances and Interfirm Knowledge Transfer,” *Strategic Management Journal* (17), 1996, pp 77-91.

Narduzzo, A., and Rossi, A. “Modularity in Action: GNU/Linux and Free/Open Source Software Development Model Unleashed,” 2003, Available at: http://opensource.mit.edu/papers/narduzzorossi.pdf

Reagans, R., Zuckerman, E., and McEvily, B. “How to Make the Team: Social Networks vs. Demography as Criteria for Designing Effective Teams,” *Administrative Science Quarterly* (49:1), 2004, pp 101–133.

Roberts, J.A., Hann, I-H., and Slaughter, S. A. “Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects,” *Management Science* (52:7), 2006, pp 984–999.

Schilling, M.A. “Towards a General Modular Systems Theory and Its Application to Inter-Firm Product Modularity,” *Academy of Management Review* (25), 2001, pp 312–334.

Schilling, M.A., and Phelps, C.C. “Interfirm Collaboration Networks: The Impact of Large Scale Network Structure on Firm Innovation,” *Management Science* (53:7), 2007, pp 1113–1126.

Shankland, S. “*Standardized Linux gains support,”* San Francisco, CA: CNET Networks, ZDNet UK, 2002.

Singh, P.V., and Tan, Y. “Stability and Efficiency of Communications Networks in Open Source Software Development,” *Proceedings of the 15th annual WITS, 2005.*

Singh, P.V., and Tan, Y. “Planning to First Release: A Conditional Hazard Function Approach for Investigating Open Source Software Development Time,” *Proceedings of the 16th annual WITS,* 2006.

Stewart, K J., and Gosain, S. “The Impact of Ideology on Effectiveness in Open Source Software Development Teams,” *MIS Quarterly* (30:2), 2006, pp 291–314.

Stewart, K. J, Ammeter, T.A., and Maruping, L. "Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects," *Information Systems Research* (17:2), 2006, pp 126–144.

Ullmann-Margalit, E. *The Emergence of Norms*. Oxford: Clarendon, 1977.

Uzzi, B., and Spiro, J. "Collaboration and Creativity: The Small World Problem," *American Journal of Sociology* (11:2), 2005, pp 447–504.

Von Hippel, E. *The Source of Innovation*, Cambridge: MIT Press, 1988.

Von Hippel, E., and von Krogh, G. "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science," *Organization Science* (14:2), 2003, pp 209–225.

Walker, G., Kogut, B., and Shan, W. "Social Capital, Structural Holes, and the Formation of an Industry Network," *Organization Science* (8:2), 1997, pp 109–125.

Wasserman, S., and Faust, K. *Social Network Analysis*. London: Cambridge University Press, 1994.

Weber, S. *The Success of Open Source.* Cambridge, Mass.: Harvard University Press, 2004.

Wooldridge, J. Econometric analysis to Cross Sectional and Panel Data. Cambridge, MA, MIT Press, 2006.

Xu, J., Gao, Y., Christley, S., and Madey, G. "A Topological Analysis of the Open Source Software Development Community," *Proceedings of the 38th HICSS*, 2005.