FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SOCIAL NETWORK STRUCTURE AS A CRITICAL SUCCESS CONDITION FOR OPEN SOURCE SOFTWARE PROJECT COMMUNITIES

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

BUSINESS ADMINISTRATION

by

David Hinds

2008

To: Dean Joyce Elam
College of Business Administration

This dissertation, written by David Hinds, and entitled Social Network Structure as a Critical Success Condition for Open Source Software Project Communities, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Susan Clemmons

Kaushik Dutta

Kenneth Lipartito

Debra VanderMeer

Mary Ann Von Glinow

Ronald M. Lee, Major Professor

Date of Defense: March 13, 2008

The dissertation of David Hinds is approved.

Dean Joyce Elam
College of Business Administration

Dean George Walker
University Graduate School

Florida International University, 2008

DEDICATION

I dedicate this dissertation to my mother and to my father. There are no words to express how important they have been in my life and how much I love them.

Lillian Marie Hinds

1916 - 2007

Richard Howard Hinds

1916 – 2007

ACKNOWLEDGMENTS

I wish to thank my Major Professor, Ronald Lee, who walked with me throughout this entire process, and who was both invaluable guide and source of inspiration. I am grateful for all of my other committee members, each of whom provided me with his or her special perspective and contribution to my thinking. I especially wish to thank Susan Clemmons for her help and guidance through the statistical work. I am also appreciative of the reading and helpful comments provided by J.C. Wang, as well as all of the help and support provided by the "Musketeers," my PhD student associates.

The data collection efforts for this research were extensive and I wish to thank Karel Alemany who spent endless hours in extracting and compiling data, and also Joseph Imperato who provided critical advice and help with the database work. We utilized the research databases of the University of Notre Dame and the Libresoft Project, and my thanks go to Greg Madey and the others who prepared and made these databases available. I also recognize that funding for this work was provided through the Dissertation Year Fellowship of the University Graduate School.

Without the love and support of my family, none of this would have been possible. I thank my children Matthew, Kimberly, and Christopher for being so wonderful and for encouraging me to go on. Finally, I am so grateful for my wife Brenda who has always been the love of my life and who is my partner in so many ways.

ABSTRACT OF THE DISSERTATION

SOCIAL NETWORK STRUCTURE AS A CRITICAL SUCCESS CONDITION FOR OPEN SOURCE SOFTWARE PROJECT COMMUNITIES

by

David Hinds

Florida International University, 2008

Miami, Florida

Professor Ronald M. Lee, Major Professor

In recent years, a surprising new phenomenon has emerged in which globally-distributed online communities collaborate to create useful and sophisticated computer software. These open source software groups are comprised of generally unaffiliated individuals and organizations who work in a seemingly chaotic fashion and who participate on a voluntary basis without direct financial incentive.

The purpose of this research is to investigate the relationship between the social network structure of these intriguing groups and their level of output and activity, where social network structure is defined as 1) closure or connectedness within the group, 2) bridging ties which extend outside of the group, and 3) leader centrality within the group. Based on well-tested theories of social capital and centrality in teams, propositions were formulated which suggest that social network structures associated with successful open source software project communities will exhibit high levels of bridging and moderate levels of closure and leader centrality.

The research setting was the SourceForge hosting organization and a study population of 143 project communities was identified. Independent variables included measures of closure and leader centrality defined over conversational ties, along with measures of bridging defined over membership ties. Dependent variables included source code commits and software releases for community output, and software downloads and project site page views for community activity. A cross-sectional study design was used and archival data were extracted and aggregated for the two-year period following the first release of project software. The resulting compiled variables were analyzed using multiple linear and quadratic regressions, controlling for group size and conversational volume.

Contrary to theory-based expectations, the surprising results showed that successful project groups exhibited low levels of closure and that the levels of bridging and leader centrality were not important factors of success. These findings suggest that the creation and use of open source software may represent a fundamentally new socio-technical development process which disrupts the team paradigm and which triggers the need for building new theories of collaborative development. These new theories could point towards the broader application of open source methods for the creation of knowledge-based products other than software.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

1. INTRODUCTION

Communities of volunteer individuals and organizations are collaborating to create and use public domain computer programs, commonly known as "open source software." In recent years, these communities have had a surprisingly powerful impact. For example, 78 million web server sites now utilize the software products which were created and freely distributed by the Apache open source community. Apache holds a 50% "market share" of this huge software base compared with a 35% share held by Microsoft. What is even more surprising is that the Apache volunteers have maintained a substantial market lead over Microsoft since 1995.

Industry players, such as IBM, HP, Computer Associates, Novell, Sun, and Netscape, view the open source movement as a strategic opportunity, and are dedicating significant resources to open source projects (Bessen 2005) and/or releasing their previously closed source software, such as Eclipse, Open Office, and Mozilla, in an attempt to create open source projects (West and O'Mahony 2005). Red Hat, a distributor of Linux software, has a market capitalization value of $2 billion. Over a recent eighteen-month period, 50 new ventures with an open source business model have attracted some $400 million in venture capital (Lacey 2005). Governments and NGOs around the world, including both industrial and developing countries, are mandating the purchase of open source software by their agencies and are encouraging the development of such software for public purposes (Evans and Reddy 2003, Weber 2003). In particular, the Chinese government is supporting open source software by funding the development of a Chinese version of Linux, and by promoting the use of open source as part of an ongoing program to combat software piracy (Trombly 2005).

In summary, open source software project communities have created much of the software infrastructure of the internet, they are changing the structure of the computer industry, they have spawned new entrepreneurial opportunities, and their activities are increasingly viewed by governments as an important policy issue. Most organizations and individuals can now benefit directly from the computer programs being produced by these communities. Yet, all of this has been accomplished by non-paid volunteers and/or by the employees of corporations who do not directly profit from their employees' activities. These open source developers operate from remote locations around the globe, they choose their own tasks, and they work at their own pace. The result has been described as a kind of "bazaar" of activity (Raymond 1999).

How can this be? Traditional economic theory would predict that open source projects should not even survive, let alone thrive. Efforts to explain this intriguing phenomenon have referred to open source as a new form of organization, a new model for production, and a new kind of innovation. Benkler (2002, 2006) considers open source to be part of a more generalized set of web-based collective activities which are characterized by a governance structure that is neither hierarchical nor market-directed, but rather is a "bottom-up" communal type in which participation is open and voluntary and is not motivated by economic incentive. Benkler (2002) refers to this phenomenon as "commons-based peer-production." Benkler (2002) and Lessig (2001) argue that these kinds of open and web-based forms of development, production, and innovation offer certain advantages over market-based and hierarchical forms. They suggest that these advantages include access to a broader pool of talent, more efficient matching of contributors to tasks, improved motivation of contributors, and increasing returns (network externalities) associated with contributor and user participation.

Prior to the introduction of the internet, these "web-based initiatives" were constrained by high transaction costs associated with communication, coordination, and transportation. The internet and worldwide web are now drastically lowering these costs, thereby enabling new forms of collective action and collaboration. In essence, this phenomenon is now possible because thousands of individuals throughout the world can work together in developing a single product, as long as that product can be digitized and made available on the web.

What exactly is open source software? In essence, it is computer software in which the source code is revealed to the public. This is in contrast to proprietary software, in which the source code is hidden from the public (e.g. as in the case of most Microsoft products). The physical significance of revealing the source code is that it enables anyone with the necessary skills to copy, modify, use, and/or distribute the software. However, the application of this simple idea has broad and significant implications with regards to collective production methods, innovation, property rights, virtual communities, and even culture.

Similar to communities of practice (Wenger 1998, Brown and Duguid 2000), open source software communities self-organize around a shared interest in the practice of producing and using certain software applications. However, unlike communities of practice in which members are often co-located and familiar with each other, these open source communities are globally-distributed and comprised of largely unaffiliated individuals. While these groups are referred to in this study as "communities," they often do not even resemble the common notion of a community. In effect, they are more like "communities of strangers."

While most of the public attention has been directed to large efforts such as Linux and Apache, the future of open source software may lie in the more than 100,000 open source projects that have already been registered on the host site SourceForge.net. However, only a small fraction of these projects have achieved clear success. A study of SourceForge projects by Capiluppi et. al. (2003) concluded that most of the projects hosted at the site in 2003 were dead, with only a small fraction showing any activity over a six-month period. A review of SourceForge by the author showed that 87 projects have been registered in the domain of genealogy, and yet only 4 or 5 of these appear to have achieved any significant level of success. Why did these particular projects succeed, while the others did not?

Efforts to explain the workings of open source software projects have taken various perspectives, including technological, psychological, ecological, and organizational. For example, a modular software design is considered to be a critical technological feature (MacCormack et. al. 2006). In terms of psychological factors, much research has been conducted into understanding the motivation of contributors who spend time and effort on open source projects even though many of them receive no direct financial compensation (Raymond 1999; von Hippel and von Krogh 2003; Lerner and Tirole 2002; Lakhani et. al. 2002). From an ecological perspective, a survival of the fittest argument has been proposed based on a limited set of niche opportunities for particular types of software. As organizational entities, open source software projects have been studied in terms of the types of online groups or communities that form to support and enact the projects.

While its roots reach back into the 1960's, the current open source software movement only began in the 1980's, with the most rapid growth occurring within the last 10 years. As would be expected with a relatively new phenomenon, most of the open source research has been exploratory, descriptive and/or anecdotal. Explanatory work has been mostly limited to studies of developer motivational factors, with very little quantitative research involving the correlates of project success. In fact, the very definition of "success" of an open source software project has been problematic. Based on the current state of research, we are still unable to adequately address the question: "Why do some open source software projects succeed while others fail?"

1.1. Research Approach

Part of the difficulty in addressing the mystery of success is the novelty of the open source phenomenon and the fact that research is still at an early stage. However, another part of the difficulty is that open source projects are dynamic and complex entities, with many influencing factors and emergent properties that are difficult to define and measure. In some respects, a new open source software project is similar to a start-up new venture, in terms of defining the goal/mission, acquiring human and physical resources, coordinating work efforts, and competing with other projects and organizations.

An appropriate research perspective is needed which can adequately represent these complex and dynamic entities and which can then address their conditions of success. A social network structural perspective is chosen in reference to that purpose. Studies of social network structure have been conducted since the 1930's in the social sciences, and, more recently, are gaining prominence in many fields, ranging from corporate strategy to network-based physics. A social network perspective focuses on the nature and structure of the relationships between social entities, rather than the attributes of the entities themselves.

The social structuralist perspective is useful because it provides a unifying framework for a wide range of interdisciplinary concepts, and it also allows for the precise definition of constructs and the quantitative investigation of success factors. In addition, very little social network research has been conducted on open source software project communities and the potential insight to be gained from such an approach is expected to be significant. In this regard, Healy and Schussman (2003) suggest that:

> ... researchers should attend more closely to the social structure of the open source software community. The process of open source software development is embedded in particular structural and organizational contexts that theorists of open source software have so far paid little attention to. Investigating them offers a promising route for an original sociological perspective on this exciting phenomenon.

A social network perspective is taken, based on the assertions of social capital theory, which is one of the most prominent of the social network theories. Also considered are other more domain-specific network studies of the impact of social structure on the effectiveness of teams and work groups. The associated social network concepts are used as a platform for synthesizing the results of theory and prior research in a diverse set of related areas including open source software, teams, communities, and innovation.

1.2. Research Question

The primary motivation for this research is to investigate the conditions which are associated with success in open source software project communities. Specifically, the research is designed to apply a social network perspective towards the study of social network structures which may be related to success. In pursuit of this goal, the following research question is defined:

> What is the relationship, if any, between the social network structure of an open source software project community and the success of the community?

This research question defines the phenomenon of interest as being open source software project communities, with social network structure and community success as the primary constructs for investigation. The research definitions for these three concepts are presented in the following section.

1.3. Definitions

In this section, three key constructs are defined which are central to the specification of the research question, and which also help to define the scope and approach for the overall research effort.

Open source software project community. In defining the notion of an open source software project community, it is first necessary to define an open source software project. For the purposes of this research, an "open source software project" is defined as a software development project which utilizes an open source license accepted by the Open Source Initiative (OSI 2004), and which has a unique identity and repository of source code.

The "community," then, consists of the population of individuals that emerges to carry out the open source project. Specifically, this includes individuals who spend a non-trivial amount of their time and effort on project-related activities. These individuals are considered to be "members" of the project community (also referred to in this research as "actors" or "participants"). While it is possible to think of all open source developers as comprising a kind of community, the study definition is limited to the community of individuals who are associated with a particular project.

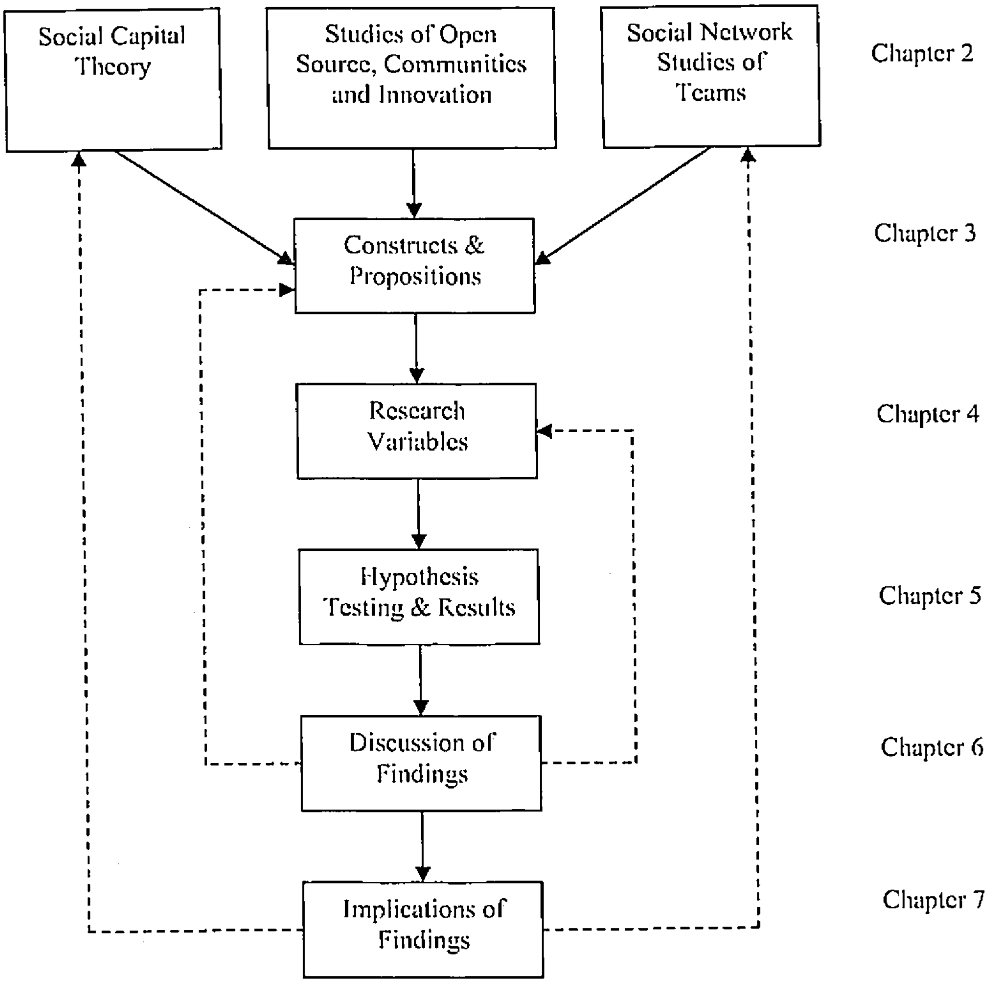
Social network structure. For the purposes of this research, the social network structure of an open source software project community is defined as the pattern of interactions and relationships among and between the members of the community (ingroup ties), and between members of the community and other individuals outside of the community (outgroup ties). The focus, then, is on the relationships between individuals rather than the attributes of the individuals themselves.

Community success. The construct of open source software project community success can be defined in various ways, depending upon the perspective of the relevant stakeholder, as well as the type of community that is involved (Crowston et. al. 2004). Perhaps the most fundamental definition for community success is "the general level of activity associated with the community.“ Thus, successful communities are those which attract many participants who collectively spend a significant amount of time and effort on community activities. For certain communities which create a product such as software, another construct of success can be defined as the "output" of the community (e.g. the quantity of software produced). Finally, community success can also be measured in terms of "impact" beyond the boundaries of the community (e.g. extent to which software produced by a community has resulted in industry-wide changes.) For the purposes of this research, however, the success of an open source software project community will be conceptualized in terms of its member activity level and its output of software.

1.4. Dissertation Structure

In the opening chapter, the intriguing nature of the open source software phenomenon is described along with its surprising impacts on business and society. Some of the efforts to explain the "economic mystery" of its very existence are discussed, although it is noted that much of this mystery seems to remain. The "success mystery" is then described along with the social network based research approach that is being used to address this mystery. The primary research question was posed and key related constructs were defined. The remainder of this dissertation is organized into the following six chapters and follows the conceptual framework which is presented on Figure 1.

Figure 1
Conceptual Framework



Chapter 2: This chapter begins with an overview of theoretical and conceptual foundations, involving a description of the various relevant knowledge domains and how they relate to this particular research work. For each domain, a review of the literature is presented with special emphasis on aspects of the literature that relate to the research question.

Chapter 3: Here, the overall research model is described. This model incorporates the foundational theories and other research results into a conceptual model of the relationship between social network structure and open source software project community success. This is followed by a definition and description of all research constructs and a description and justification of the propositions to be considered.

Chapter 4: In this chapter, the research method and study design are presented, along with a description and formal specification of the research variables that are used in defining the testable hypotheses in Chapter 5. Then, the sampling strategy is presented and the procedures for extracting and compiling archival data for the sampled projects are described, followed by a description of the resulting project sample and the associated research dataset.

Chapter 5: Chapter 5 includes a discussion of the analytical procedures that were performed on the research dataset to test the hypotheses. Key data analysis procedures included principal component analysis, regression assumption testing, and regression analysis, including both linear and quadratic. This is followed by a presentation of the results.

Chapter 6: In this chapter, the results presented in Chapter 5 are summarized in reference to the hypotheses and prior literature. This is followed by the presentation and analysis of a set of conjectures for explaining the results.

Chapter 7: In the last chapter, the fundamental conclusions of the research are summarized and discussed, along with their implications for theory, research and practice. This includes a discussion of the contributions to research and practice, research limitations, and the directions for future research work.

2. LITERATURE REVIEW

This chapter contains a review of literature regarding theoretical work, empirical studies and other publications which are relevant to the dissertation. In the first section, each literature domain is noted and its relevance is described. In subsequent sections, each of these domains is reviewed including social networks, open source software, teams and work groups, communities, and innovation. Each section focuses on aspects of the domain that are important for this work, ending with a subsection which describes the social network perspectives and studies that have been conducted in the domain.

2.1. Theoretical and Conceptual Foundations

Social network analysis and theories involving social structure are fundamental to the work. The structural dimension of social capital theory and social network studies of centrality and prominence provide the primary theoretical foundations. Studies of social structure that have been performed in various relevant domains including open source software, teams, communities, and innovation are considered. Social network analytical techniques are also applied in defining and calculating social structural measures for the purpose of operationalizing and testing the hypotheses.

Of course, the target phenomenon for this work is open source software, and the scope of the research includes the projects which are formed to create and update the software as well as the communities of individuals that emerge to carry out the projects. Beyond the social network studies, the other areas of interest regarding open source software include explanatory mechanisms, community formation and participant roles, developer motivation, work processes, and the measures and factors of success.

The concept of "team" has been selected as the primary reference phenomenon, and open source software project communities are presumed to be a kind of software development team, considering that both groups are task-driven and that the software product created by an open source project community may be virtually indistinguishable from the software created by a traditional team. Key aspects of the team literature include social structural studies of team and work group effectiveness as well as virtual or emergent organizations, in that open source project communities are sometimes described as virtual organizations.

While the team is used as the primary reference concept, it is also recognized that open source software project groups are a kind of community. Therefore, prior studies of communities are considered, especially those involving online or virtual communities. The connection of open source software projects with innovation is recognized and therefore some of the key aspects of innovation research are also reviewed, especially the literature regarding open and distributed innovation.

2.2. Social Networks

In fundamental terms, a social network is a network representation, in which the nodes of the network are social entities (such as people or organizations), and the links of the network are relations between the social entities (such as advice-giving or trade). The term "social network analysis" refers to a broad set of methods and tools for coding and analyzing social network representations. In contrast, the domain of social network theory involves the application of network concepts and perspectives to various aspects of social psychology, sociology, and organizational science. The basic concepts of social network analysis are described in the next section, followed by a review of relevant social network theories and a discussion of network-based theories of social capital.

2.2.1. Social Network Analysis

First noted in 1934 in the "sociograms" of Moreno (1934), social network analysis has grown into a large collection of methodologies, measurements, and tools that can be used for the description and analysis of social networks and social structure (Wasserman and Faust 1994, Scott 2000, Carrington et. al. 2005). The primary mathematical foundation for social network analysis is provided by graph theory, and the methods draw heavily on matrix algebra for coding and manipulating network data.

The basic units of analysis are the dyads and triads which represent pairs and triples of nodes. Features of dyads that are commonly studied include reflexivity, symmetry, and transitivity (Wasserman and Faust 1994). At the network level, the primary types of constructs that are defined include density, centrality and centralization, cliques and components, and positions and structural equivalence (Scott 2000). The social network analytical method is, by definition, a multi-level method, in that the nodes reflect data at an individual unit of analysis, the links reflect data at the relational (dyadic) level of analysis, and the resulting measures of network structure are produced at the group or network level of analysis.

Centrality is one of the most ubiquitous of the social network measures. It is typically described as a "location" of an individual actor within a network which is associated with importance or prominence (Wasserman and Faust 1994). Many alternative ways of defining centrality have been proposed, with the most popular being degree (the number of ties of the focal actor with other actors), closeness (the extent to which the focal actor can reach other actors through "short paths"), and betweenness (the extent to which the focal actor is located on paths which connect other actors to each other).

A fairly recent extension of the notion of centrality has been suggested by Everett and Borgatti (1999), in which the centrality definitions are applied to subgroups (of a larger group or network) rather than to individual actors within a network. Questions which could be addressed with such methods include: 'how central are the women within an organization, as opposed to the men?' or 'to what extent are financially-oriented individuals central to the advice-giving networks of the firm?'

Social network analysis has a number of positive features with respect to its use as an analytical tool. Its use can reveal patterns that are not discernable with other methods. These patterns may be reflected in quantitative social network measurements or they may be observed qualitatively in two- or three-dimensional graphical network representations. Further, the use of social network analysis provides a quantitative method for studying complex social phenomena such as kinship, community structure, corporate interlocks, and elite power, whose investigation would otherwise be limited to the use of qualitative tools.

2.2.2. Social Network Theory

Social network theories utilize a social structural perspective in which the focus of investigation is the pattern of interactions and relationships among and between the social entities. These theories consider the relationships between members rather than the attributes of the members themselves, and they involve the study of the social network structures of groups and their impact on either individual outcomes or group outcomes.

There are two primary branches of theory development in social networks. The oldest branch is based in the social sciences, primarily in sociology, social psychology, and organizational theory. One of the primary theoretical domains of this branch is that of social capital (which is described in the following section). The other main branch of theory development is centered in the physics community. The physics studies began in the late 1990's based on the work of Watts (2003). In the process of studying the small-world phenomenon, Watts discovered that a particular network structure, often identified by a power-law distribution (also known as a Pareto or Zipf curve), is startlingly common, and is found in a wide range of natural, social, and artificial phenomena (Barabasi 2002, Watts 2003, Buchanan 2002). Such networks, which are often described as "small world networks" or "scale-free networks," are characterized by a set of relatively large "hub nodes" which comprise 20 percent of all the nodes but which account for 80 percent of all the links. This stream of research does not often connect with the social science based structural research, even though many of the problems addressed are essentially the same (Freeman 2004). Some of the structural research work associated with open source software has been based on this physics genre.

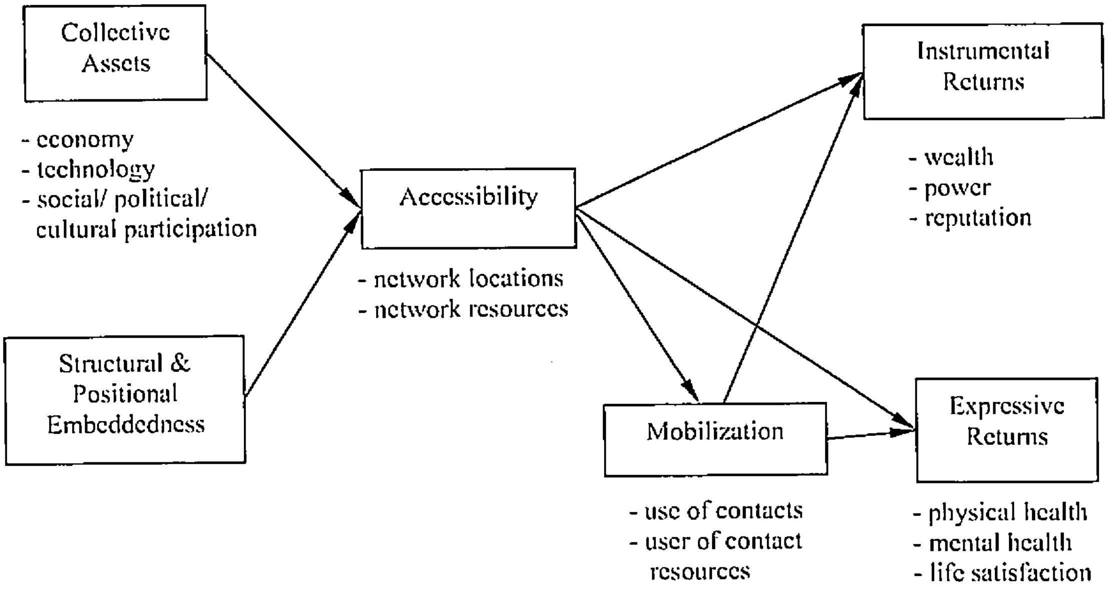
In one respect, social network theory is a frame of reference which connects a wide variety of organizational research including theories of resource allocation, power differences, routine decision rules, complex cognitive constructions, sets of contractual relationships, rational solutions to incentive problems, and complex adaptive systems (Lomi and Pattison 2004). Lomi and Pattison (2004) argue that organizational researchers in many of these areas have a common interest in understanding the role of network ties in the evolution of various social forms and settings such as firms, markets, industries, and states. Within these research communities, they state that network-based models and methods are valued for their ability to address a wide variety of substantive and analytical issues.

2.2.3. Social Capital Theory

One domain in which social network theory is perhaps the most prominent is the area of social capital. Social capital theory provides a collective context in which individual relationships are embedded within a network of relationships (Granovetter 1985). Social capital consists of both the network itself and the assets that may be mobilized through the network (Bordieu 1986). Social capital can be applied at an individual level (considering individual benefits) or at a group level (considering group benefits). Groups can be defined as teams, communities, organizations, and even regions (Putnam 2000) and nations (Fukuyama 1995).

Social capital theory uses an information processing paradigm (Simon 1976) to explain how social network structure affects social outcomes at the individual level and at the group level. Social ties are viewed as conduits for the flow of information, knowledge or other resources. Lin (2001) argues that social networks are the foundation of social capital. As noted on Figure 2, his theory of social capital begins with the collective assets of the network as a whole and the structural and positional embeddedness of particular actors. These constructs are related to accessibility (extent to which resources can be accessed) and mobilization (extent to which these resources are used). These constructs then result in returns to the individual or to the group, including both instrumental returns and expressive returns.

Figure 2
A Theory of Social Capital
(Adapted from Lin 2001)



Nahapiet and Ghoshal (1998) identify three dimensions of social capital including cognitive, relational, and structural. The cognitive dimension includes the shared vocabulary and narratives of the social group. The relational dimension considers the constructs of trust, norms, and identification. However, it is the structural dimension that is most relevant to social structural research. This dimension considers constructs of network ties, network configuration, and appropriable organization (whereby organizations that create value in one context may have value in another context).

Burt (2001) addresses an apparent paradox regarding the value of an open network (with links extending outside of a social group) versus a closed network (which is internally cohesive). He argues that both types of networks are valuable, depending upon the context. Open or brokerage networks, which are the basis of Burt's structural holes theory (Burt 1992), are considered to be valuable if there is a need for accessing resources outside of the group, where such resources tend to be non-redundant. Closed networks, which are studied by Coleman (1988) and others, seem to be most useful when resources are already available and the focus is on their use.

However, Burt's notion of "brokerage" has an alternative interpretation based on the intention of the actor in the brokering position. This type of network position can be used to keep the other actors isolated in order to appropriate value from them. This is referred to as the "tertius gaudens" orientation (or "the one who benefits"). The contrasting viewpoint is a "tertius iungens" orientation (or "the one who joins"), in which the focal actor utilizes the brokering position to help connect the other actors to their benefit. This alternative interpretation of the structural holes position is often referred to as "bridging."

2.3. Open Source Software

Most of the research regarding open source software has been conducted within the last 10 years, and much of it has been descriptive and exploratory. The most commonly used methods are qualitative except in the case of contributor motivation in which surveys are primarily used. The main objectives of the work completed to date have been to describe the phenomenon in general and to address the mystery regarding how these projects can work at all.

The literature review of open source software is divided into five sections. The first section covers general descriptive literature, some of which has been written by open source practitioners who often take the perspective of an advocate. In the next section, the various metaphors are covered which have been used to describe the mechanisms by which open source software projects function. The third section includes a review of fairly extensive studies that have been performed which attempt to explain the motivation of volunteer non-paid contributors. In the fourth, the limited studies that directly address open source software project success factors are reviewed. In the final section, social network studies of the open source phenomena are presented.

2.3.1. Descriptive Studies

The open source movement is characterized by self-organization, a modular structure of goods, and a culture containing certain identifiable norms and standards, such as notions of freely-redistributable products, strict customs regarding the rights of the founder-leader, and contributor attribution (Raymond 1999, O'Reilly 1999, Iannacci 2003). Perhaps the most fundamental and enduring aspect of open source culture is the notion of freely available software, as originally expressed in The GNU Manifesto (Stallman 1985). As described by Raymond:

> All members agree that open source (that is, software which is freely re-distributable and can readily be evolved and modified to fit changing needs) is a good thing and worthy of significant and collective effort. This agreement effectively defines membership in the culture. (Raymond 1998)

However, it must be noted that the notion of "freely available software" refers to its accessibility and not its price. As such, even though open source software is often made available free of charge, the fundamental premise of the movement is that the software must be accessible to anyone for their use and modification, and charging a fee for open source software is not prohibited.

Weber (2004) notes three essential features of the culture that are reflected in the Open Source Definition (OSI 2004):

1. Source code must be distributed with the software or otherwise made available for no more than the cost of distribution.
2. Anyone may redistribute the software for free, without royalties or licensing fees to the author.
3. Anyone may modify the software or derive other software from it, and then distribute the modified software under the same terms.

Descriptions of open source software projects indicate that they are typically initiated by an individual (or a small group) who assumes the role of founder and usually provides (or provides access to) systems and development components, as well as communication infrastructure. Once an initiative has been started, a maintainer (administrator or leader) role typically emerges that continues to monitor the progress of the project and provides certain ongoing services such as maintenance of the enablement system (e.g., the web site) and enforcement of (or possibly adjustment to) the project norms (Almarzouk et. al. 2005).

The development and communications infrastructure is often provided by a hosting organization such as SourceForge (2005) or Savannah (2005), which in some respects acts as an incubation center for new projects. SourceForge, for example, provides a web-based host platform which includes a source code repository (version control system), public forum facilities, project web pages and a search engine. This host platform also includes the rules and policies which govern the behavior of community members. The host organization will typically provide some general policies while individual community leaders will often provide more specific policies geared to the needs of their particular community.

As a project community grows, various developers may become aware of the project and gain sufficient interest to join the community and to assist in expanding the code. This process may progress as other individuals start to use the project software and then sometimes choose to participate (e.g. by reporting bugs or requesting new features). In large well-developed projects, third party organizations such as code distributors may become involved to package, distribute and service the software. If the project is aligned with their strategy, sponsoring corporations may provide contributions of cash or facilities or in-kind contributions of employees who act as developers on the project. Non-profit foundations may be formed to assist in promotional efforts, hold any physical assets that may be needed, manage the intellectual property of the project (under open source licenses), and protect the developers from law suits.

The individuals that participate in open source software projects are often described as comprising a community. These communities have been described as having an onion-like structure, with a central core of highly active individuals surrounded by other layers of progressively less active individuals. One example of this is presented by Ye et. al. (2005) in which the central core is composed of the project leaders and core members, with five outer layers containing active developers, peripheral developers, bug reporters, passive users, and stakeholders, respectively.

Most studies of open source software do not differentiate the various types of projects that may have quite different characteristics. However, there appear to be significantly different kinds of projects that warrant separate treatment and a few studies have addressed this issue. For example, West and O'Mahony (2005) describe mature projects that require a kind of transformation in order to achieve their mature status:

> Mature community managed projects have developed a series of major releases. They have defined membership criteria or boundaries: contributors know whether they are in or out of the project. Mature projects have adopted governance mechanisms that enable representation in commercial and legal settings. They also have an ecology of institutions that support and/or extend their work. These institutions may be non-profit organizations such as the Open Source Development Lab, firms developing complementary products, or other community projects with which they collaborate. (West and O'Mahony 2005)

Ye et. al. (2005) identify three types of projects that are suggested to have different characteristics in terms of goals, styles of control, and patterns of evolution for the software code and the project community. These types include:

1. Exploration-oriented projects - attempt to create leading edge solutions which involve innovative approaches.

2. Utility-oriented projects - are directed towards filling a void in functionality.

3. Services-oriented projects – are geared to maintaining stable code and providing ongoing services to large groups of stakeholders

Another typology of projects is noted by West and O'Mahony (2005), who distinguish between community-founded projects and spin-off projects, in which organizations attempt to open up previously proprietary code. The authors note that spin-off projects seem to have a different life cycle. In the start-up phase, for example, the "seed" code base is usually large and well-established, and its introduction to a new open source software project community often raises special technical, relational, and legal issues. They hypothesize that mature spin-off projects require different kinds of project leadership in order to address issues related to the intentions of the sponsor, assuming that the sponsoring organization remains heavily involved. They further clarify that spin-off projects are different from corporate-sponsored projects, in which corporations supply various types of support but do not become directly involved in the governance of the project.

Other types of projects may involve those which are dominated by paid individuals working for sponsoring corporations, as opposed to those which are dominated by non-paid volunteers. In terms of software type, Raymond (1999) has suggested that open source software projects may have different characteristics depending upon the type of software involved, where he identifies three types: infrastructural software, application software, and middleware.

Somewhat related to the identification of different project types, developmental taxonomies have been proposed to identify different project growth stages that are associated with different project characteristics. For example, SourceForge recognizes seven categories of "development status" (the first six of which are described by Rothfuss 2002), including:

1. Planning – No code has been written. The scope of the project is still in flux.

2. Pre-alpha – Very preliminary source code has been released. The code is not expected to compile or even run.

3. Alpha – The released code works at least some of the time, and begins to take shape. Preliminary development notes may show up. Active work to expand the feature set of the application continues.

4. Beta – The code is feature-complete, but retains faults. These are gradually weeded out, leading to software that is ever more reliable.

5. Production/Stable – The software is useful and reliable enough for daily use. Changes are applied very carefully, and the intent of changes is to increase stability, not new functionality.

6. Mature – There is little or no new development occurring, as the software fulfills its purpose very reliably. Changes are applied with extreme caution, if at all.

7. Inactive – There is no project activity of any kind.

The above life cycle description is somewhat idealized, and there is evidence that many projects never move beyond the early stages (Capiluppi et. al. 2003). These types of projects appear to become inactive without ever achieving any useful level of functionality. Capiluppi et. al. (2003) suggest that this may be due to the limited supply of open source software developers in relation to the large demand for such developers that is generated by the many new open source software project startups.

In some cases, descriptions of open source software projects have been presented as normative or prescriptive, although the basis for most of these descriptions is limited because they are typically based on a single case, a very small sample of projects, and/or non-systematic studies. Some of the important social and technological features that have been proposed (Raymond 1999, Weber 2004, Sturmer 2005) include:

- Large number of project participants
- A bias against forking a single project into multiple projects
- Evolution of cooperative norms

- The lack of specific deadlines or task assignments

- Version releases that begin early in the project and continue on a frequent basis

- Separate releases for stable versions versus cutting edge versions

- Toleration for many different ideas and allowing for code branches that remain within the scope of the project

- A large and diverse group of developers and users with different skill sets

- Modular software design

- Sufficiently good seed code that must run and must have a compelling design

- Sufficient promotional activities designed to "get the word out"

- Application of an appropriate open source license

- Use of a well-known programming language

In terms of desirable features of the open source software project community, Raymond (1999) has suggested that a strongly interconnected core combined with loosely coupled collaborations in peripheral parts of the community is a necessary feature to address the problem associated with Brooks' Law, which states that the complexity and communication cost of a software development project increases with the square of the number of developers[1], while the amount of work accomplished increases linearly (Brooks 1975). However, this "solution" to the problems associated with Brooks' Law does have its cost, in terms of redundant efforts that typically occur within the loose collaborations at the periphery. This problem appears to be mediated, at least in some

[1] This geometric effect is noted if the software development team is conceptualized as a social network of developers. In this case, if the team includes "g" developers, then the maximum number "L" of possible links between the developers is calculated as L = g (g-1) / 2. (Raymond 1999)

cases, by a global supply of open source software developers who may be willing to participate.

The role of the project leader(s) has also been suggested to be of critical importance (Pavlicek 2000), and some of the important features of open source software project leadership that have been proposed (Raymond 1999, Weber 2004) include:

- Leadership style which is not based on a power relationship
- Delegation of as much as possible
- Treating users as co-developers
- Keeping developers and users constantly stimulated and rewarded
- Listening to the beta-testers
- Having the ability to recognize good designs and incorporate them into the project
- Having good design and coding skills as well as people and communication skills

2.3.2. Mechanisms and Metaphors

Various metaphors have been proposed in an attempt to describe the mechanisms involved in open source software projects and to explain how they can work at all. These metaphors have included collective actions (Benkler 2002; von Hippel and von Krogh 2003), forms of production (Benkler 2002; Kogut and Metiu 2001), forms of innovation (von Hippel and von Krogh 2003; von Krogh et. al. 2005), organizational ecologies (Chengalur-Smith and Sidorova 2003), interactive social systems (Lanzara and Morner 2003), self-organizing processes (Morner 2003), complex adaptive systems

(Muffato and Faldani 2003), social networks (Gao et. al. 2003), virtual communities (Crowston and Scozzi 2002), and political economies (Weber 2004).

The metaphors of community, innovation, and social network are discussed in later sections. In this section, the metaphors of collective action, organizational ecologies, and self-organizing agent-based systems are discussed. While these metaphors can be useful in conceptualizing the kinds of mechanisms at work in open source software, they do not, by themselves, represent an explanation of the antecedents for success.

Open source software as a collective action. Collective action theory addresses the logic and problems associated with the production and use of public goods (Hardin 1982). Public goods are defined as goods which are sometimes nondepletable but are always nonexcludable (Barry & Hardin 1982, Olson 1965).[2] Viewed from the perspective of the consumer, public goods are nondepletable in that one individual's consumption does not impact another individual's consumption – everybody can get a copy. They are also nonexcludable in that consumption is open to every member of the group, whether or not they have contributed to the provision of the good – everybody has a right to a copy. Viewed from the perspective of a potential developer (contributor), these properties describe a type of social dilemma (Dawes 1980, Hardin 1968), whereby individuals may not be motivated to contribute but rather may choose to wait for others to

[2] For example, public television is both nondepletable and nonexcludable, while a public park is only nonexcludable – because it is physical space, it is depletable.

make contributions, thereby leading to suboptimal results (involving quality, usefulness, usability, stability, timeliness or even existence).

Open source software is clearly a public good, in that it is nondepletable (due to its digital nature) and nonexcludable (due to the nature of open source licenses). Therefore, open source software projects are viewed as collective actions, where the projects must address the social dilemma and the fundamental supply problem. It is addressing this collective action problem that has inspired the many studies of contributor motivation.

Open source software as an organizational ecology. When viewed as an organizational ecology, the persistence of certain open source software projects can be explained by using a "survival of the fittest" argument, with respect to various niches that exist for particular types of software. Also implied by an ecological view is the existence of a first-mover advantage.

Lanzara and Morner (2003) view open source projects as knowledge creation efforts which operate within an ecology of agents, artifacts, rules, resources, activities, practices, and interactions. They examine the creation and use of knowledge artifacts, and support the application of the metaphor by identifying ecological mechanisms of variation, selection, and stabilization that are manifested in open source projects.

Chengalur-Smith and Sidorova (2003) use a population ecology perspective, and propose (but do not test) four related hypotheses:

1. More reliable open source projects are more likely to survive.

2. Size of the open source project will be positively related to project reliability and hence to project survival.

3. Age of the open source project will be positively related to project reliability and hence to project survival.

4. Open source projects that occupy a broad niche are less likely to survive in the short term.

Open source software as a self-organizing agent-based system. A number of researchers have concluded that open source software project communities are self-organizing systems. For example, Morner (2003) uses autopoietic organization theory (Luhman 1984), which is based on the self-organizing concept of autopoiesis ("self-maintenance") to describe and analyze open source projects. She concludes that communication connectivity and systemic memory are important stabilizing factors because "they reduce the overall need for coordination and therefore make the self-organization of developers easier." Muffatto and Faldani (2003) view open source software as a complex adaptive system in which mechanisms of self-organization result in emergent behaviors. They identify particular features of open source projects which correspond with the complexity-related concepts of variation, interaction, and selection.

Another group of researchers take an explicit agent-based view and create agent-based simulation models in an attempt to understand the dynamic mechanisms involved. Madey et. al. (2004) have created a Swarm-based simulation model with parameters based on data collected from the SourceForge archives. In their model, they define a project swarm (for a particular project), which is embedded in a cluster swarm (a group of interconnected projects), which is embedded within an open source software development swarm (representing, for example, the entire set of projects hosted by SourceForge). Developers are represented as agents who, at each time point in the simulation, can choose to start a new project, join an existing project, or quit an existing project. The growth of an "artificial SourceForge" is then simulated and the results are compared with empirical data from SourceForge. They conclude that preferential attachment modified by a dynamic "fitness factor" provides the best fit (Barabasi 2002) and they use this observation to conclude that open source software project communities are self-organizing entities.

Wagstrom (2004) has created an agent-based model (Wagstrom et. al. 2005) with parameters based on data collected from three sources: 1) the Advogato.org social networking site, 2) web log aggregators which capture the blogs of open source developers, and 3) mailing lists of selected open source projects. In the model, developers are represented as agents who are seeking a particular kind of software. This desire is represented using an NK model (Kauffman 1993) to represent a string of features, and agents are able to change features at each time point in order to achieve a better fitness value. Agents then make decisions regarding project participation based on the extent to which the project features fit with their desires. The resulting simulated growth and decline curves show patterns which resemble those observed in actual projects.

2.3.3. Developer Motivation

Studies have shown that contributors are not normally motivated by traditional economic incentives, but rather by instrumental factors associated with fulfilling a need, and by intrinsic factors such as enhanced reputation, expertise development (learning), self-fulfillment, and basic fun and enjoyment (Raymond 1998, 1999, von Hippel and von Krogh 2003, Lerner and Tirole 2002, Lakhani et. al. 2002). Raymond (1998) explains this by characterizing the open source movement as a "gift culture," where benefits accrue from the reputation for giving away one's time, effort, and creativity. However, he also notes that some contributors may be more motivated by the notion of pride of craftsmanship, which also accrues benefits in terms of reputation, but based on a different motivational concept.

A great deal of this research has been motivated by the collective action problem and for finding factors which explain how this problem can be overcome in active open source software projects. It should be noted, however, that a survey by Lakhani and Wolf (2005) shows that approximately 40 percent of open source developers are not volunteers, but rather are paid employees of organizations which encourage or even direct their employees to work on particular open source projects. In this context, the collective action problem does not seem to apply and, in fact, a new avenue of research that is developing involves studying the motivational factors of organizations that provide such support (Bessen 2005).

2.3.4. Success Studies

While many of the studies described in the previous sections have implications regarding factors of success, none of these studies address the question of success factors for specific projects in a systematic way. For example, the agent-based models of Madey et. al. (2004) and Wagstrom et. al. (2005) attempt to suggest the general mechanisms by which projects grow and decline. However, these results are not applicable to the success or failure of particular projects. In terms of studies of contributor motivation, Weber (2004) recognizes the limitation of these works: "The summary point is that individual motivations do not make up anything like a full explanation for the success of open source." In this section, a few studies that directly address open source software project success factors are described.

A statistical analysis in April and May of 2002 by Krishnamurthy (2002) was conducted on SourceForge projects which were categorized as being in a "mature" development status. Descriptive statistics for these projects show that "the vast majority were developed by a relatively small number of individuals, few of these projects generated much discussion, projects with more developers tended to be viewed and downloaded more often, the number of developers working on the project was correlated with the age of the project, and a smaller percentage of participants were assigned as project administrators in larger groups." In this study, the implied measure of success was the project's status as "mature."

In a large sample study of SourceForge projects, Healy and Schussman (2003) take an approach similar to Krishnamurthy (2002) by generating various descriptive statistics for active open source software projects including developers, commits, downloads, site views, unique message authors, and messages. They observe that many of these measures exhibit a power law distribution and that only a few projects achieve clear success. They recognize that the work to date does not address the success question, and they offer the following hypotheses for future research regarding success:

> 1. The more successful an open source project, the more professional its core contributors will be.
>
> 2. Successful open source projects will tend to have core participants mobilized in a way similar to core participants in successful social movement organizations. (Effective project leadership seems to us one of the most likely candidates for differentiating successful projects from unsuccessful ones.)
>
> 3. Successful open source projects will tend to have a strong hierarchical component, at least in the ways that they manage the relationships between lead (and core) developers and other contributors.
>
> 4. The closer a successful project is to the core of the broader open source software community, the more hierarchy will be found in its management style. Thus, for instance, the social organization of kernel hackers will be more hierarchical than that of developers of add-on applications for the GNOME or KDE desktop environments, because the kernel is the essence of the operating system, whereas additional text editors or desktop calculators are much less important. (Healy and Schussman 2003)

Stewart and Ammeter (2002) conducted an analysis of 240 open source software projects to investigate factors which lead to attracting user attention ("popularity") and developer activity ("vitality"). They examined the effect of organizational sponsorship, target audience (developer versus end-user), license choice, and development status. Their preliminary results indicate that vitality significantly affects popularity, and that sponsored projects are more popular than non-sponsored projects. The surprising preliminary conclusion was that vitality was not affected by sponsorship, development status, or target audience.

Crowston and Scozzi (2002) conducted a multiple regression analysis of SourceForge data from 2001 to test success measures that might support Katzy and Crowston's (2000) theory of competency rallying which relates to the success of virtual organizations. Four open source software project measures were defined which were somewhat related to the four independent variables described by competency rallying theory: 1) identification and development of individual competencies, 2) identification of market opportunities, 3) marshalling of competencies, and 4) management of a short-term cooperative effort. Three measures of success are defined: 1) interest shown by users, 2) development status, and 3) intensity of work undertaken by developers. They find some support for their hypotheses for two of the three success measures.

In a subsequent paper devoted to the subject of success measures, Crowston et. al. (2004) present a range of measures that could be used to assess the success of open source projects. They develop these measures based on a literature review, a consideration of the nature of the open source development process, and the opinions of open source project participants. They describe measures along the following dimensions, based on the type of analysis that they conducted, and note that the use of a particular set of measures is dependent upon the research purpose and the particular stakeholder perspective of interest:

1. Review of literature
   System and information quality
   User satisfaction and use
   Individual or organizational impacts

2. Consideration of the open source process
   Project output and process
   Outcomes for project members

3. Opinions from open source project participants
   User - satisfaction and involvement
   Product - meets requirement, code quality, portability, availability
   Process - activity, adherence to process, bug fixing, time, age
   Developers - involvement, varied developers, satisfaction, enjoyment
   Use – competition, number of users, downloads
   Recognition – referral, attention and recognition, spin-offs, influence

In a more recent effort to address open source project success, Crowston et. al. (2005) outlined an approach for studying the work practices of open source project groups and relating these practices to team effectiveness. In this paper, the authors utilize the Hackman model of group effectiveness (Hackman 1986)[3], and combine it with theories of coordination and collective mind to suggest a set of propositions for relating work practices to team performance in open source software projects.

2.3.5. Social Network Perspectives

A limited number of studies of open source software projects and communities have been conducted with the use of social network analysis, and of these, even fewer have taken a social network theoretical perspective. Most of these studies have used social network analytical methods to describe and characterize the projects and associated project groups, while only a very small number have used a social network perspective as a framework for theory building.

With the objective of determining what a "typical" open source software project looks like, Hunt and Johnson (2002) studied the activity distribution of approximately 4,000 projects on the "most active list" of SourceForge in October and November of 2001, using number of downloads per week as the measure of activity. They found that the distribution generally followed a Pareto curve. They suggest that this may result from the winner-take-all nature of the projects.

Madey et. al. (2002) studied the social networks of 39,000 SourceForge projects from January 2001 to March 2002. They defined a link to exist between two developers

[3] The Hackman model is illustrated on Figure 3 and discussed in section 2.4.1.

if those developers were both registered for the same open source project. They observed that the number of developers on a project, number of projects served by a developer, and cluster size (excluding the largest cluster) all followed power law distributions. Further, they noted that networks associated with individual projects are connected together into clusters by a small number of "linchpin developers." They interpret the power law results as evidence that open source projects are self-organizing entities.

In a subsequent study of 50,000 SourceForge projects by Gao et. al. (2003), they define two types of nodes (bipartite graph): developer nodes and project nodes, and they define a link to exist between a developer and a project if that developer is registered on that project. The study was conducted over a two year period between 2001 and 2003 in an attempt to identify dynamic patterns that exist within the overall SourceForge network of practice. They also observed the power law in the degree distribution and the cluster distribution, and they observed a clustering coefficient of 0.7 (compared with 0.2 for a random network of similar size). In terms of the dynamics over the two year period of study, they observed that the network diameter decreased from 8 to 6 and that the average degree increased (indicating greater connectivity).

This line of research was continued by Xu, et. al. (2005). Using a 2003 data dump from SourceForge, they again found the power law distributions in various measures that are indicative of small-worlds networks. Based on an analysis of diameters, they conclude that both core developers and non-core developers are important in connecting the overall open source community, primarily due to their facilitation of communication flow between projects.

Wagstrom et. al. (2005) studied the structure of the overall open source community by using a variety of data sources, including: 1) the Advogato.org social networking site, 2) web log aggregators which capture the blogs of open source developers, and 3) mailing lists of selected open source projects. Comparing his results with the studies of Madey and others, Wagstrom concludes that there are more links between projects than was originally thought, which indicates that the overall open source community is cohesive. He further notes that the prior assumption that cliques exist within this overall community may not be valid, in that such cliques were not found.

Crowston and Howison (2004) examined 120 project teams (communities) from SourceForge and analyzed interactions associated with the bug reporting archives. In particular, they measured and compared the "communication centralization"[4] measures of the different projects. They found a wide variation of centralization among the projects, and further found that this variation was negatively correlated with the number of developers and active users associated with the bug reporting system – i.e., the larger projects were less centralized. They conclude that it is wrong to assume that all open source projects are associated with a particular social structure and that the examination of social structure offers an interesting avenue for future research. In a practice sense, they suggest that open source project teams should spend more effort on creating social structures which are considered to be favorable.

In summary, the works of Madey and Wagstrom are focused on the overall open source community (across many projects), and do not address the networks associated

[4] The authors differentiate "communication centralization" from "code development centralization", and suggest that the "onion models" of community structure depict the development-based measure, but not the communication-based measure.

with particular projects. Crowston and Howison do address the social networks of individual projects. However, none of these works are explicitly informed by theories of social structure, but rather they are based on research associated with software development and team effectiveness (Crowston), or they are motivated by the desire to parameterize agent-based models (Madey and Wagstrom).

2.4. Teams and Work Groups

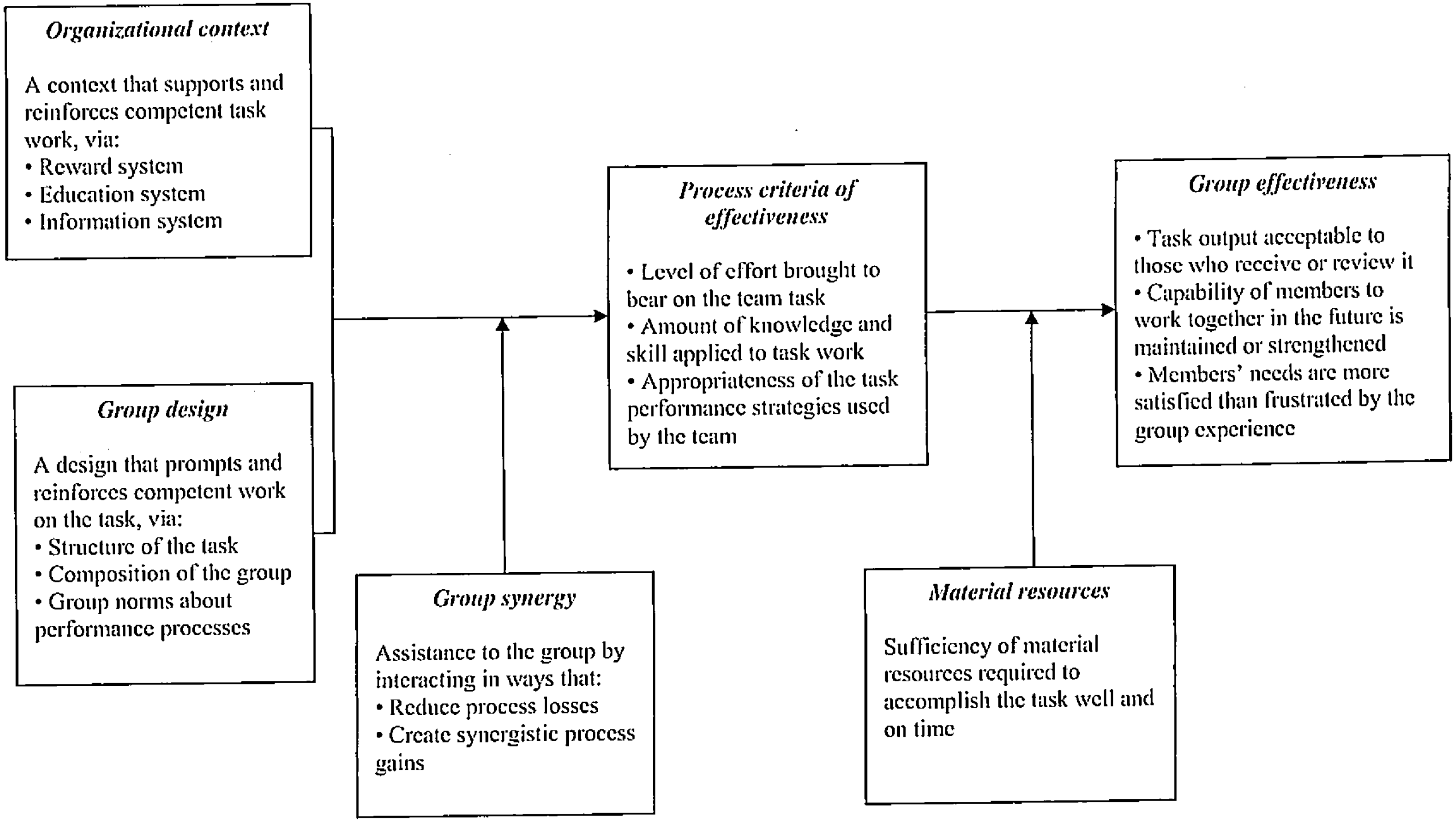
This section begins with a review of relevant studies of teams and work groups especially with regard to their effectiveness. Open source project communities are often described as "emergent," and the next section includes a discussion of emergent and virtual organizations. This is followed by a discussion of social network perspectives that have been applied to teams and work groups within an organizational context.

2.4.1. Work Group Effectiveness

Literally hundreds of studies of teams and work groups and the factors which contribute to their effectiveness have been conducted over the past 50 years (Kozlowski and Bell 2003). Some of the factors which have been suggested as antecedents of team effectiveness include "collective efficacy, group cohesion, team-level goals, and interpersonal conflict" (Balkundi and Harrison 2006).

One prominent example of a model for group effectiveness is the Hackman framework (Hackman 1986). As shown on Figure 3, this model relates organizational

Figure 3
Hackman's Normative Model of Group Effectiveness
(from Hackman 1986)


Organizational context
A context that supports and reinforces competent task work, via:
• Reward system
• Education system
• Information system
Group design
A design that prompts and reinforces competent work on the task, via:
• Structure of the task
• Composition of the group
• Group norms about performance processes
Group synergy
Assistance to the group by interacting in ways that:
• Reduce process losses
• Create synergistic process gains
Process criteria of effectiveness
• Level of effort brought to bear on the team task
• Amount of knowledge and skill applied to task work
• Appropriateness of the task performance strategies used by the team
Material resources
Sufficiency of material resources required to accomplish the task well and on time
Group effectiveness
• Task output acceptable to those who receive or review it
• Capability of members to work together in the future is maintained or strengthened
• Members' needs are more satisfied than frustrated by the group experience

context and group design to process factors which then drive the group effectiveness result. Mediating factors consist of group synergy effects and the material resources required to perform the group tasks. In Hackman's model, three dimensions of group effectiveness are suggested: 1) task output acceptable to those who receive or review it, 2) capability of members to work together in the future is maintained or strengthened, and 3) members' needs are more satisfied than frustrated by the group experience.

A recent review of team effectiveness studies recognizes two dimensions of team or group effectiveness (Kozlowski and Bell, 2003). These are team performance and team viability. Team performance is mostly aligned with instrumental functions and is the extent to which the team achieves its objectives and produces suitable output. Team viability is more of a social or expressive concept, which relates to the team's cohesion and its ability to retain its members and to continue functioning. While these two dimensions are conceptually distinct, a recent meta-analysis has established that there is a close connection and cross-correlation between team performance and team viability (Balkundi and Harrison 2006).

A virtual team is a particular type of team which has been defined by Luri and Raisinghani (2001) as a "group of people who work together although they are often dispersed across space, time, and/or organizational boundaries." (Luri and Raisinghani 2001) In their study of the effectiveness of virtual teams, the authors identified team processes and the relations among team members as having the strongest impact on team performance and member satisfaction, while the leadership style was only moderately associated with team effectiveness.

2.4.2. Emergent Organizations

With respect to formal organizations versus emergent social structures, Brown and Duguid (2000) comment that:

> … self organization and formal organization are not simple alternatives. Nor are they simply complementary. They live in tension with one another. Innovation is often developed in the productive management of related tension between emergent practice and formal process. (Brown and Duguid 2000)

Virtual organizations are sometimes viewed as emergent, and they are defined by Malone and Davidow (1992) as being "a cooperation of independent partners who combine their knowledge and skills to fulfill a certain goal, in the form of research or a product." Strader et. al. (1998) define a virtual organization as "a temporary network of companies that comes together quickly to exploit fast-changing opportunities." Mowshowitz (2002) considers virtual organizations to be a type of organizational approach, rather than a particular organizational form. In his view, the key characteristics of virtual organizations are:

> the separation of conceptualization from execution of tasks, and the use of objective criteria for the allocation of resources… one that allows for crafting structures that enable management to switch at will between different options for implementing an organization's requirements. (Mowshowitz 2002)

Crowston and Scozzi (2002) argue that open source software project communities are actually virtual organizations and they support this argument by showing the relevance of the competency rallying theory (Katzy and Crowston 2000) which relates to the success of virtual organizations. Markus et. al. (2000) describe the concept of a virtual organization and then use the open source project community as their canonical example of such an organization.

Strader et. al. (1998) define the life cycle of an electronic virtual organization for the purpose of discussing the requirements for a supportive information technology infrastructure. The four phases of life cycle include:

1. Identification – opportunity identification and selection.
2. Formation – partner identification and selection, and partnership formation.
3. Operation – design, marketing, financial management, manufacturing, and distribution.
4. Termination – Operational termination and asset dispersal.

2.4.3. Social Network Perspectives

Ahuja and Carley (1999) suggest a network model for virtual organizations in which the fit between task characteristics and network structure is an important determinant of network performance, where "network" refers to a virtual organization. They use this model as a framework to study a research-based virtual organization involving the design and development of an artificial intelligence architecture.

In a review of computational and mathematical organization theory, Carley (1995) compares hierarchical-centralized structures (often associated with traditional organizations) with democratic-decentralized structures (which are associated with virtual organizations). She notes that hierarchical or centralized structures tend to exhibit lower performance than democratic or decentralized structures. However, under certain circumstances, hierarchical structures are more reliable. For simple tasks, decentralized structures perform better, while for complex tasks, hierarchies, network-forms, and matrix-forms are superior. More democratic structures tend to learn faster and therefore perform better in the short run, while hierarchical and centralized structures tend to respond more slowly but more accurately to environmental changes.

In a meta-analysis of studies of the effect of social structure on team effectiveness, Balkundi and Harrison (2006) conclude that teams with a high density of ties within the team are more effective, and that teams that are more central within a network of other teams are also more effective. Finally, team performance is positively associated with the centrality of the team leader within the team network. These results were applicable for both instrumental ties (associated with task-oriented activities) and for expressive ties (associated with socially-oriented activities). The authors further assess mediating factors, and found that the structural effects on team effectiveness are weakened as a team matures and members become more familiar with each other.

Two particular types of teams that are especially relevant to open source software project communities include software development teams and virtual teams. In the case of software development teams, a social network study by Yang and Tang (2004) concluded that group cohesion was positively related with performance and that the group structures were critical to the overall team effectiveness. While no social network studies of virtual teams were found, a study of effectiveness of virtual teams by Luri and Raisinghani (2001) suggests that team cohesiveness is positively related with effectiveness, a result which is consistent with the conclusion of Balkundi and Harrison (2006) that teams with high density are the most effective.

2.5. Communities

The individuals who participate in open source software projects have been frequently described as communities. In one context, the individuals who work on a particular project are viewed as comprising a project community. In another context, all individuals who work on any open source project are viewed as members of the overall open source community. Weber (2004), expands on this metaphor of community:

> The open source community ... is indeed marking out a set of organizing principles. These include criteria for entering (and leaving), leadership roles, power relations, distributional issues, education and socialization paths, and all the other characteristics that describe a nascent culture and community structure. (Weber 2004)

In this section, a variety of organizational forms are discussed including communities of practice, online communities, and networks of practice. While the community of practice form has been fairly well defined (Wenger 1998, Brown and Duguid 2000), the other forms are somewhat overlapping and conflicting definitions have been offered (Brown and Duguid 2000, Teigland 2003). In order to better understand the connections between these various kinds of "communities," a framework is developed as shown on Table 1.

The framework involves two dimensions: 1) the primary motivation for the community (social-driven, practice/knowledge-driven, or task-driven), and 2) the primary communication mode for member participation (face-to-face or electronic / virtual / online). The framework is consistent with the descriptions of communities of practice offered by Wenger (1998), and with the classification of network of practice proposed by

Brown and Duguid (2000). The definition proposed by Teigland (2003) maps to multiple cells within the framework.

Table 1
A Framework of Community Types

Motivation of the community	*Face- to-face interaction*	*Electronic (virtual, online) interaction*
Social-driven	Social clubs	Online (social) communities
Practice/knowledge-driven	Communities of practice	Networks of practice
Task-driven	Community action organizations (e.g. Habitat for Humanity)	Open source software project communities Content production communities (e.g. Wikipedians)

2.5.1. Communities of Practice

Huysman et. al. (2003) define communities as: "social entities whose actors share common needs, interests, or practices: they constitute the basic unit of social experience." A community of practice, then, is a particular type of community in which practices are shared. Communities can exist to develop the expertise of their members, to take action (solve problems), and/or to satisfy member needs for group interaction.

Wenger views a community of practice as being both an organizational form and a theory or mechanism of learning. The term "community of practice" was coined in 1991 by Lave and Wenger (1991) as an outgrowth of their research into "situated learning." The social theory of learning which is represented by Lave and Wenger within the context of communities of practice conflicts with traditional theories of learning which typically assume that learning results from teaching. In the context of communities of practice, the authors suggest that learning results from "doing."

Wenger defines the boundary of a community of practice as a layered construct:

> ... a community of practice is a node of mutual engagement that becomes progressively looser at the periphery, with layers going from core membership to extreme peripherality." (Wenger 1998)

Multiple communities of practice can intersect in various ways, resulting in "constellations" of communities. These intersections provide important links to the rest of the world through boundary objects (artifacts) and/or brokers.

In a related stream of work, Brown and Duguid also define and analyze the features of communities of practice (Brown and Duguid 1991). Brown (1998) observes that members of the community:

> ... pick up valuable 'know-how' ... from being on the periphery of competent practitioners going about their business and from being able to move from the periphery to the center to participate in aspects of the practice and then move back to the periphery to observe some more.

Wenger identifies two kinds of communities – communities of practice and communities of interest – and compares them with two kinds of teams (Table 2). In general, the communities are viewed as "emergent" forms of organization in that they tend to evolve or end organically and are not the result of a planned action or any specific hierarchical governance mechanism. The boundaries of these emergent forms tend to be fuzzy or undefined, and their purpose is based on the needs and interests of the community. In contrast, the formal operational teams and project teams are all "planned" forms, in that they are typically organized and planned by management. The boundaries of these planned forms are normally quite clear, and their purpose is based on the needs of the hierarchical organization in which they are embedded.

Table 2
A Framework of Communities and Teams
(Adapted from Wenger et. al 2002)

	What's the purpose?	*Who belongs?*	*How clear are the boundaries?*	*What holds them together?*	*How long do they last?*
Communities of Practice	To create, expand, and exchange knowledge	Self-selection based on expertise or passion for topic	Fuzzy	Passion, commitment and group identification	Evolve and end organically
Communities of Interest	To be informed	Whoever is interested	Fuzzy	Access to information	Evolve and end organically
Operational Teams	To take care of an ongoing operation or process	Membership assigned by management	Clear	Shared responsibility for the operation	Last as long as the operation exists
Project Teams	To accomplish a specified task	People with a role in accomplishing the task	Clear	The project's goals and milestones	Begin and end per project schedule

Wenger et. al. (2002) provide some guidance regarding the facilitation of communities of practice. In general, they suggest that communities are not planned organizational forms, and therefore are not managed in the traditional sense. Rather, they are emergent organizational forms, and the most effective "management style" is one of stimulation and facilitation, rather than command and control.

Based on experiences with 60 communities of practice, Gongla and Rizzuto (2001) have defined five evolutionary stages for these types of communities:

1. Potential stage – the fundamental function is connection as individuals find one another and link up.

2. Building stage – the fundamental function is the promotion of memory and context as core members learn about each other, share experiences, create roles and norms, and share a repertoire of stories.

3. Engaged stage – the fundamental function is access and learning as members build trust and commitment to the community and begin to reach out to new members.

4. Active stage – the fundamental function is collaboration as individuals engage with other community members and rely on the community's knowledge in their work.

5. Adaptive stage – the fundamental function is innovation and generation as the community develops new capabilities and adapts to new environments.

2.5.2. Online Communities

There has been considerable discussion of online (or virtual) communities, and yet there is little work which defines what an online community is and how it relates to a community of practice. The generally accepted concept of an online community is as shown on Table 1. When compared with a community of practice, an online community mostly uses an electronic form of communication, while a community of practice is primarily face-to-face. Another distinction shown on the table is that online communities tend to be more socially driven, while communities of practice are more practice- or knowledge-driven (although this observation is not relevant if open source software project communities are viewed as being an online community). Little research was found which focuses on the implications of these differences.

Brown and Duguid (2000) refer to the notion of "net communities" and view them as being formed around textual documents:

> Net communities extend a long tradition of communities forming around documents ... Textual communities may be as old as texts themselves. Shared and circulating documents, it seems, have long provided interesting social glue.

By extending this concept to include both source code repositories as well as textual artifacts, it could be argued that open source software project communities are online or "net" communities. However, the task-orientation of open source project communities would seem to differentiate them from other forms of electronically-mediated communities.

2.5.3. Networks of Practice

Brown and Duguid (2000) define the notion of "networks of practice" as: "networks that link people to others whom they may never get to know but who work on similar practices." They state that networks of practice are known for their reach, and that this reach has been significantly enhanced by information and communication technology. They recognize Wenger's definition of community of practice, and view such communities as "subsections" of networks of practice.

Interpreted in terms of an open source software project community, then, the overall network of developers who work on various projects (e.g. all developers registered on at least one SourceForge project) can be viewed as a network of practice, while the specific group of developers who work on a particular project can be viewed as a task-driven (online) community.

2.5.4. Social Network Perspectives

Structural studies of communities, in the sense of communities of practice and related forms, are limited. Schenkel et. al. (2000) define five structural properties which can be used to characterize communities of practice. These include:

1. Connectedness – In a community of practice, every member is connected, directly or indirectly, to every other member.

2. Graph-theoretic distance – Relative to organizational networks in general, communities of practice have shorter graph-theoretic distances between all pairs of members.

3. Density – Relative to organizational networks in general, communities of practice have a greater density of ties.

4. Core/periphery structure – Communities of practice have core/periphery structures rather than clique structures.

5. Coreness – The greater an individual's participation in a community of practice, the greater his or her coreness score.

Further, Schenkel et. al. (2000) propose (but do not test) a set of relationships between social structure of communities of practice and knowledge sharing and performance. These are:

- Proposition 1A: For smaller communities of practice (less than or equal to 40 members), knowledge transfer increases linearly with density.

- Proposition 1B: For larger communities of practice (more than 40 members), knowledge transfer increases curvilinearly with density.

- Proposition 2A: For communities of practice solving more complex problems, performance will increase as the variance among members' coreness values decreases.

- Proposition 2B: For communities of practice solving more routine problems, performance will increase as the variance among members' coreness values increases.

- Proposition 3: Community participants with higher coreness scores will have more community-specific knowledge and thus a higher level of individual performance.

Using collective action theory as their conceptual framework, Wasko and Teigland (2002) studied the social structure of a network of practice – a professional legal association in the United States. They found that the pattern of contributions of information was that of a generalized exchange network, in which direct reciprocity was rare. They also found that a few contributors tended to provide a large portion of the contributions, and these core contributors are viewed by the authors as forming a "critical mass." They further note that membership in this critical mass group is significantly related to occupation, expertise, the availability of local resources, and the desire to enhance one's reputation.

In comparing and synthesizing her prior studies, Teigland (2003) notes that there are significant differences in the social structures of different community forms. She notes that communities of practice are characterized by strong ties based on personal relationship, with a high degree of connectedness and "critical mass individuals" tied to one another. This compares with electronic networks of practice in which individuals are connected by weak ties based on online interaction, a high degree of connectedness is noted, and critical mass individuals are not tied to one another.

In the physics genre, Adamic and Huberman (2000) studied the social structure of visitors to web sites on the world wide web. (Such visitors might be viewed as online communities.) They found that site popularity fit a power-law distribution, which they note is characteristic of winner-take-all markets. Further, they developed a dynamic theory of site popularity which attempts to explain the distribution based on the age of the site, its mean growth rate, and the variance of its usage fluctuations.

2.6. Innovation

Open source projects have been viewed as a form of innovation. For example, von Krogh (2003) states that

> The open-source movement's unique development practices are challenging the traditional views of how innovation should work. ... The open-source movement also provides important management lessons regarding the most effective ways to structure and implement innovation.

Von Hippel and von Krogh (2003) propose that open source projects reflect a compound "private-collective" model of innovation, in which aspects of the private model of innovation (incentives to innovate are provided through the protection of intellectual property rights) are combined with the collective action model (innovators freely collaborate to produce innovation in the context of market failure).

However, the level of innovation associated with particular open source projects may vary considerably. Taking the project typology offered by Ye et. al. (2005), it would seem that exploration-oriented projects might involve radical or disruptive innovation, utility-oriented projects might involve incremental or sustaining innovation, and that service-oriented projects might involve little innovation at all. Raymond (1999) notes that Linus Torvalds, the founder of the Linux project, was not seeking innovation as a major objective: "Suppose Linus Torvalds had been trying to pull off fundamental innovations in operating system design during the development; does it seem at all likely that the resulting kernel would be as stable and successful as what we have?"

Further, the concept of innovation generally involves both the creation of new ideas and the diffusion of those ideas. In the context of open source software projects, the emphasis seems to be on the creation of the new idea, while the diffusion process occurs at least partly within a broader environment than the project itself.

In the following sections, the notion of exploration versus exploitation is discussed, followed by a review of research in open and distributed innovation. The final section presents social network perspectives that have been applied to innovation, particularly as they relate to the "development" side of innovation (development of innovations in groups) as opposed to the diffusion side (adoption of the innovation).

2.6.1. Exploration versus Exploitation

In the context of organizational learning, March (1991) describes the tension between the exploration of new possibilities and the exploitation of old certainties, and he discusses issues regarding the allocation of resources between the two approaches. He considers innovation to be part of the exploration activity and production to be part of the exploitation activity. The application of this argument to open source projects seem relevant, given that projects tend to have an innovation component and a production component. March (1991) discusses the tradeoffs between exploration and exploitation in terms of organizational communication and coordination. He suggests that organizations with effective instruments of communication and coordination (tightly coupled) are more reliable in terms of performance variance, while more loosely coupled organizations are less reliable in terms of performance, but have a greater chance of achieving an advantage over their competitors, due to their superior ability to execute multiple independent projects.

2.6.2. Open and Distributed Innovation

The notion of open innovation has been described by Chesbrough (2003) as a new and more effective model of innovation, in which individuals and organizations beyond the boundary of the firm play a greater role in the process of innovation. New ideas may originate from these outside entities or from internal sources. Then, the deployment of the resulting innovations may be executed through in-house pathways to the market or by utilizing outside firms for this purpose. This open model of innovation contrasts with the traditional closed model, which focuses on internally generated ideas and in-house pathways to the market.

Von Hippel and von Krogh (2003) argue that open source software is a manifestation of a new "private-collective" model of innovation, and they describe this model as a kind of "distributed innovation." Based on their observation that the leaders of open source project communities often designate who can be a member of a particular social category (e.g. who is authorized to commit source code), the authors suggest that: "... leadership in distributed innovation might in fact be analogous to that performed by a playing coach."

Kogut and Metiu (2001) also describe open source software as a form of distributed innovation:

> Open source software development is a production model that exploits the distributed intelligence of participants in internet communities. This model is efficient because of two related reasons: it avoids the inefficiencies of a strong intellectual property regime and it implements concurrently design and testing of software modules.

2.6.3. Social Network Perspectives

In a study of the social networks of individuals involved in organizational innovation, and their behavioral orientation, Obstfeld (2005) compares the tertius iungens ("the third who joins") orientation associated with the notion of introducing connected individuals and facilitating their collaboration, with the tertius gaudens ("the third who benefits") orientation associated with the structural holes notion of acting as a broker between individuals in order to extract personal benefits. He finds that participation in innovation (development) is positively related to the tertius iungens orientation, and that other antecedents include dense social networks and diverse social knowledge.

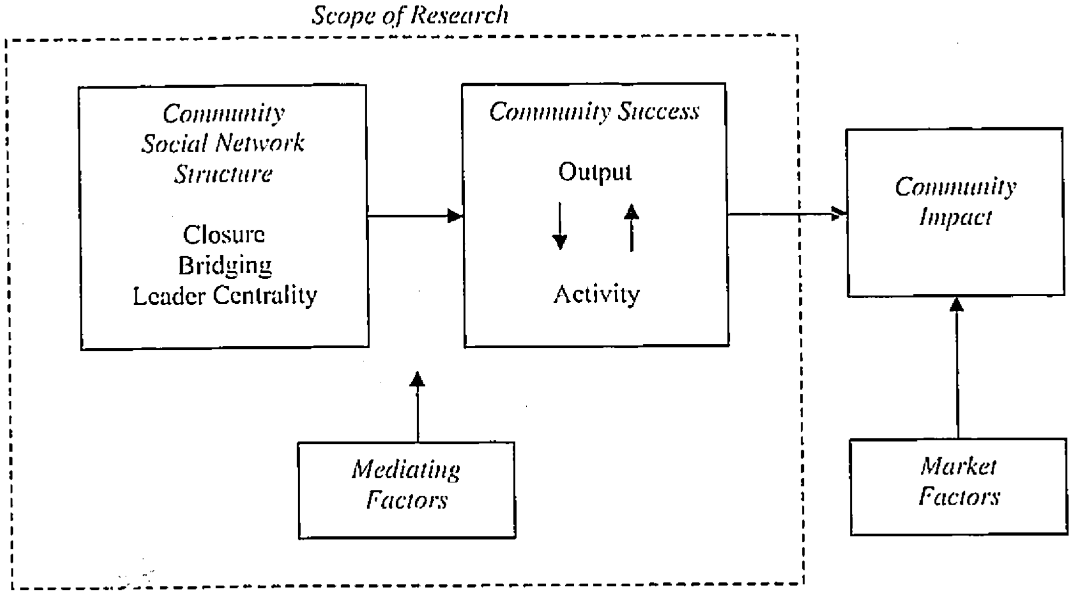
3. RESEARCH MODELS AND PROPOSITIONS

In this chapter, the research question is addressed by first presenting a conceptual research model and then defining a set of research constructs which expand upon the model. The constructs are then incorporated into a social network model of success for open source software project communities and a set of six propositions is proposed and justified.

3.1. Conceptual Research Model

In consideration of the research question and the review of theoretical and empirical literature, a conceptual research model was formulated and is presented on Figure 4. The model shows the relationship between social network structure and success for open source software project communities.

Figure 4
Conceptual Research Model



Three kinds of social network structures are included in the model: closure, bridging, and leader centrality. The closure and bridging structures are suggested based on the assertions of social capital theory which have been made in various social contexts, but especially with regard to team and work group outcomes. The leader centrality structure refers to prior social network studies regarding team leaders and the effect of their network position on the group effectiveness of the team.

In the model, community success is conceptualized as consisting of two dimensions: output and activity. The output dimension consists of the quantity of software that is produced by the project community while the activity dimension reflects the quantity of participation by community members. As noted on Figure 4, these two dimensions are modeled as having a reciprocal relationship. This is based on the suggestion that the production of more software will generally lead to greater community participation, and that increased participation will tend to attract and motivate even more developers to produce more software. To the extent that higher quality software will tend to generate a greater level of community activity than lower quality software, it is suggested that community activity can also be viewed as a proxy for software product quality.

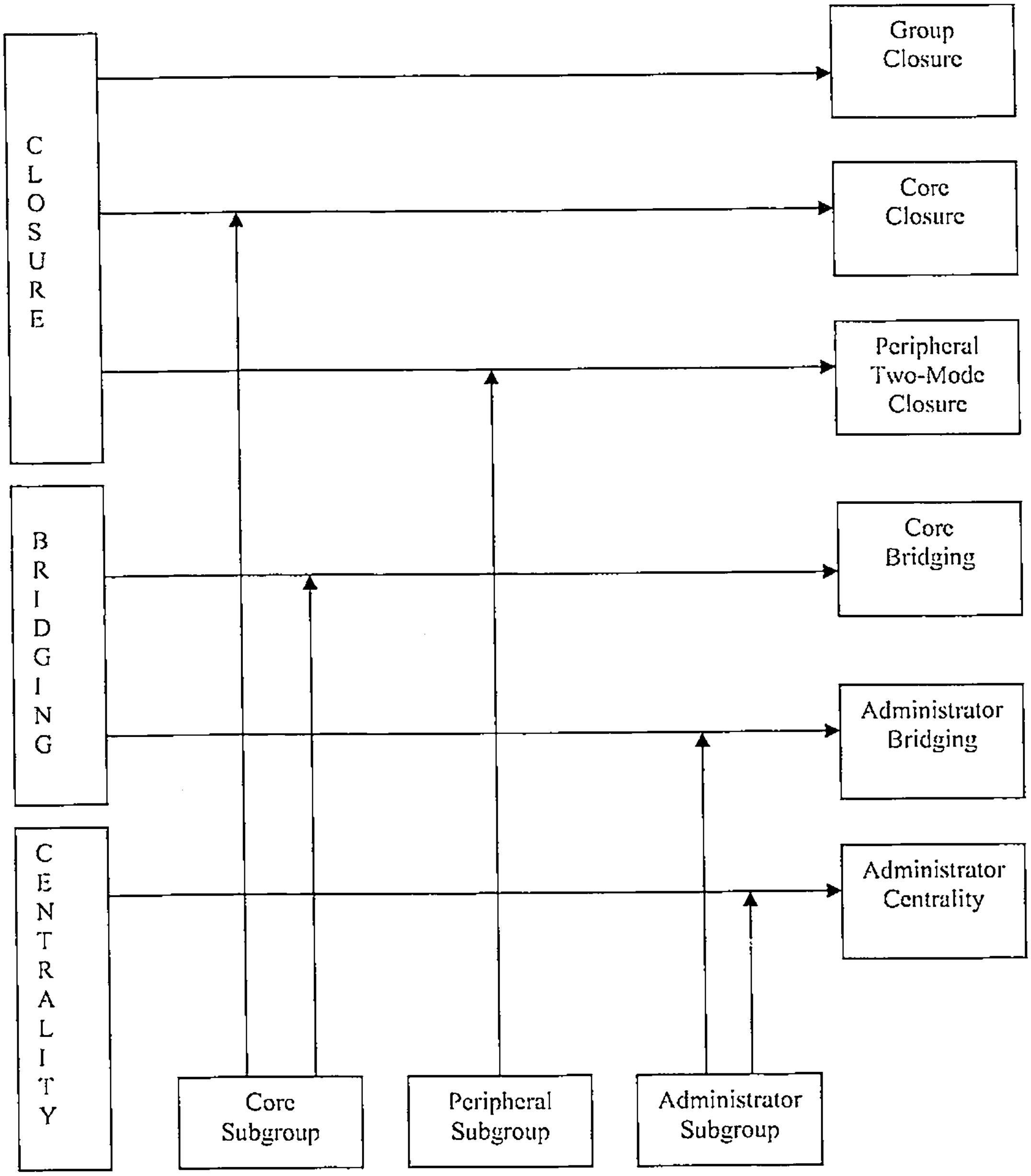
As shown on Figure 4, it is recognized that various factors may mediate the relationship between social network structure and success. These factors include group size, project type, project maturity, process/task structure, community norms, and organizational environment, among others. Even though the research propositions are limited to constructs of social network structure and success, steps are taken to control for the effect of these mediating factors, as further discussed in Sections 4.1.2 and 4.3.2.

This model suggests that community success in terms of output and activity will be related to the impact of the community beyond its boundaries, and that such community impact will be affected by market factors such as user demand or competition. Examples of community impact might include the incorporation of the produced software into the broader internet infrastructure (e.g. Linux) or the widespread acceptance of the software by the public (e.g. Mozilla). As discussed in Chapter 1, it is recognized that community impact can also be considered as a dimension of success. However, for the purposes of this research, success is defined as consisting of the output and activity dimensions and the consideration of community impact is beyond the scope of the research.

3.2. Research Constructs

Expanding on the social structural concepts of closure, bridging, and leader centrality, a set of social network constructs are proposed including Group Closure, Core Closure, Peripheral Two-Mode Closure, Core Bridging, Administrator Bridging, and Administrator Centrality. The theories and concepts which influenced the development of these constructs are illustrated on Figure 5. Using the area of teams and work groups as a primary reference domain, social network theories associated with closure, bridging, and centrality are considered in defining the six corresponding constructs. Five of these constructs consider the role of the three key community subgroups of core developers, peripheral developers, and project administrators. Aspects of these subgroup roles and positions which might be relevant to community success are considered, based on the

Figure 5
Development Framework for Social Network Constructs



review of the open source software literature, as well as other literatures in the areas of communities and innovation. In addition to the social network constructs, this section also includes a discussion of the construct of Community Success.

3.2.1. Subgroups

In adapting the theories of team effectiveness to open source communities, it is recognized that communities typically have cores and peripheries while teams generally do not. Therefore, three key community subgroups are identified for the purpose of devising social network constructs: core developers[5], peripheral developers, and administrators. The core and peripheral subgroups are relevant because they connect with prior research regarding core and peripheral developers in open source software literature (Almarzouk et. al. 2005), as well as the core-periphery concepts reflected in general studies of communities (Wenger 1998). The administrator subgroup is relevant because it connects with team research regarding team leaders and leader centrality (Balkundi and Harrison 2006) while reflecting the fact that many open source software project have more than one administrator/leader.

As noted on Table 3, the three subgroups are defined based on their different functional roles and/or levels of participation on the project. Core developers are developers who are actively involved with the project and who contribute the majority of design concepts and source code for the project software. Peripheral developers are developers who are somewhat involved with the project and who have either contributed

[5] For the purposes of this research, all members of the open source software project community are referred to as "developers" because they all contribute in some way towards the development of the software product. However, we recognize that many of these members are software users who have little or no technical expertise in the methods of software development.

source code or have posted requests or comments to the public project communication records. By definition, a developer cannot be both a core developer and a peripheral developer at the same time, although individuals will sometimes move from one subgroup to another during the course of a project, as their role and activity level changes.

Table 3
Community Subgroups

Subgroup	*Defining criteria*	*Possible indicators*
Core developers (or "Core")	Individuals who are actively involved with the project and who contribute the majority of design concepts and source code for the project software	Official designation in project records Writes and submits source code Makes design or coding suggestions
Peripheral developers (or "Periphery")	Individuals who are somewhat involved with the project and who have either contributed source code or have posted requests or comments to the public project communication records	Submits bug reports and feature requests Participates in project forum discussions May write and submit source code
Administrators	Leaders of the project who take responsibility for monitoring and guiding the progress of the project, and who are recognized as such by most group members	Official designation in project records Founded the project Designated by the project founder or by existing administrators Exerts access control over project source code repositories (is a "committer")

Administrators are developers who lead the project. They take responsibility for monitoring and guiding the progress of the project, and their special role is recognized by most group members. By definition, an administrator is also a core developer. Many projects have only one administrator, although it is not uncommon for a project to have multiple administrators who share in the leadership and administrative tasks (Almarzouk et. al. 2005, Sturmer 2005, and Ye et. al. 2005). For communities which have only one administrator, the subgroup notion is not meaningful and the "administrator subgroup" collapses to a single individual community member.

3.2.2. Closure

In social capital theory, closure is viewed as the extent to which the members of a group are connected through informal ties. This is typically represented by the social network measure of "density," which is defined as the total number of observed ties divided by the total number of possible ties. In this respect, closure can be viewed as the proportion of all possible ties that are actually connected, and a group's social network structure can be described as either "dense" if the proportion is high or "sparse" if the proportion is low.

Considering the information flow paradigm of social capital theory, closure reflects the pattern of information flows among and between the community members. In social capital theory, closure is generally portrayed as leading to positive social outcomes involving utilization of resources and group health and viability. However, some negative effects are sometimes noted, regarding groupthink and a reduced tendency to associate with outsiders. In work group effectiveness studies, closure has been generally associated with a positive impact on effectiveness, although at least one study suggested that the relationship is an inverted-U shape (Oh et. al. 2004).

The closure concept can be applied to the group as a whole, or it can be applied to any particular subgroup, in which case only the ties within the subgroup are considered. For the purposes of this research, the closure concept is extended to also consider the connections between one subgroup and the rest of the community. For this reason, the concept of "two-mode closure" is defined to consider only the ties between members of one subgroup (mode #1) and the other members of the community (mode #2). With two-mode closure, ties which are internal to either the subgroup or internal to the group of other community members are excluded.

Group Closure. As documented on Table 4, Group Closure is defined as the closure of the social network of informal ties within the total project community. Referring to social network studies of team performance, the Group Closure construct is analogous to the construct of team closure, and with this construct the "team" is viewed as consisting of all community members, regardless of whether they are core developers or peripheral developers. This is justified because it is recognized that peripheral developers contribute to the project in important ways, even though their total contribution is normally not as great as that of the core developers.

Core Closure. Applying the notion of closure to the core subgroup, the construct of Core Closure is defined as the closure of the social network of informal ties within the core subgroup of the project community. This construct views the "team" as consisting primarily of the core developers. This is an alternative view to considering the whole project community as a team. However, it is also a reasonable proposition considering that the core developers in an open source project are the most active and make the greatest total contribution to the production effort. A positive impact on the core subgroup should result in a positive impact on the entire project community.

Table 4
Social Network Constructs

Construct	*Definition*	*Relevant subgroup*
Group Closure	Extent (density) of informal ties considering all possible connections between members of the project community	None
Core Closure	Extent (density) of informal ties considering only the possible connections between members of the core developer subgroup, excluding all other possible ties	Core subgroup
Peripheral Two-Mode Closure	Extent (density) of informal ties considering only the possible connections between peripheral subgroup members and the rest of the project community, and excluding all other ties	Peripheral subgroup Core subgroup
Core Bridging	Extent of bridging ties, considering connections between members of the core subgroup and members of other project communities	Core subgroup
Administrator Bridging	Extent of bridging ties, considering connections between members of the administrator subgroup and members of other project communities	Administrator subgroup
Administrator Centrality	Central network position of the administrator or administrator subgroup in relation to the remainder of the project community	Administrator subgroup

Peripheral Two-Mode Closure. The two-mode closure concept is used to define the Peripheral Two-Mode Closure construct, which is the closure of the social network of informal ties, considering only the possible ties between the peripheral subgroup and the rest of the community[6]. This construct is defined based on the social capital notion of closure, as well as the assertions of open source literature which suggest that the involvement of peripheral members in core processes will help to fulfill their need for challenge and skills development, which will lead to an increase in their identification

[6] Considering that the "rest of the community" is equivalent to the core subgroup, this construct could just as easily be described as "Core Two-Mode Closure".

with the project community. It is expected that this will result in an increase in their participation level, thereby having a positive impact on community output and activity.

3.2.3. Bridging

Bridging is the extent to which project community members or subgroup members are connected to members of other open source software project communities. This is consistent with Burt's (1992) notion of brokerage, in the case where the brokers have a "tertius iungens" philosophy (Obstfeld 2005) which compels them to apply their positional advantage towards the benefit of the whole group, rather than using it primarily for their own personal gain. In the social capital literature, bridging is generally associated with improved access to resources and an associated increase in performance. This result has been observed in the team performance literature as well (Balkundi and Harrison 2006).

The bridging constructs are defined in relation to the bridging ties of core subgroup members and administrator subgroup members (Table 4). No bridging constructs are defined for the peripheral subgroup or the group as a whole, based on the premise that the group will not benefit from bridging ties that are held by peripheral developers who have a limited role in the project.

Core Bridging. The Core Bridging construct is the extensiveness of ties between members of the core subgroup and members of other project communities (excluding members of the focal project community). Comparing with the teams literature, this is analogous to the notion of team bridging or team centrality, where the core subgroup is considered to be "the team."

Administrator Bridging. The Administrator Bridging construct is also defined based on a more restrictive view of "team," in that it considers only the bridging ties of the administrator subgroup members to be important.

3.2.4. Leader Centrality

Leader centrality is the extent to which a team leader occupies a pivotal position within the network of information flows that are internal to the team. This central position is often associated with a perceived level of importance or prominence for an individual within the group (Wasserman and Faust 1994). In this context, a central structural position is typically represented by social network concepts such as degree centrality or betweenness centrality. Most applications of centrality involve individual nodes, although Everett and Borgatti (1999) have defined the concept of "class centrality," in which the centrality concept is extended from an individual within a network to a subgroup within a network.

Administrator Centrality. As shown on Table 4, the construct of Administrator Centrality is defined as the centrality of the administrator or administrator subgroup with respect to the total project community. In team literature, leader centrality is considered to have a positive relationship with team performance. In studies of open source projects, no works were identified which relate administrator centrality to community success. However, it is noted that the open source literature suggests that project community members are motivated by a sense of ownership in the project, and that heavy-handed control by administrators can reduce the motivation of both core developers and peripheral developers. The Administrator Centrality construct is an attempt to represent the team-related positive aspects of leader centrality with the implied negative aspects suggested by the open source literature.

3.2.5. Community Success

Community Success for an open source software project community is defined along the two dimensions of output and activity. The output level of a project community is the quantity of software that is produced by the community while the activity level is the quantity of participation by community members. These two dimensions of success include the elements of effort (reflected in the quantity of software produced) and performance (reflected in the acceptance of the community-market as evidenced by activity levels such as software downloads and page views). This is consistent with the work of Grewal, et. al. (2006) in which the authors measure "technical success" with the number of code commits and "commercial success" with the number of software downloads associated with the project.

This Community Success construct can be compared with the performance dimension of the group effectiveness construct commonly used in the team literture. Team performance is often aligned with the extent to which a team achieves its objectives and produces suitable output. An open source software project community which produces software that is widely downloaded and viewed can be said to have achieved its objectives. Therefore, the Community Success construct as defined above is generally equivalent to group performance in teams with regard to the accomplishment of task and group objectives.

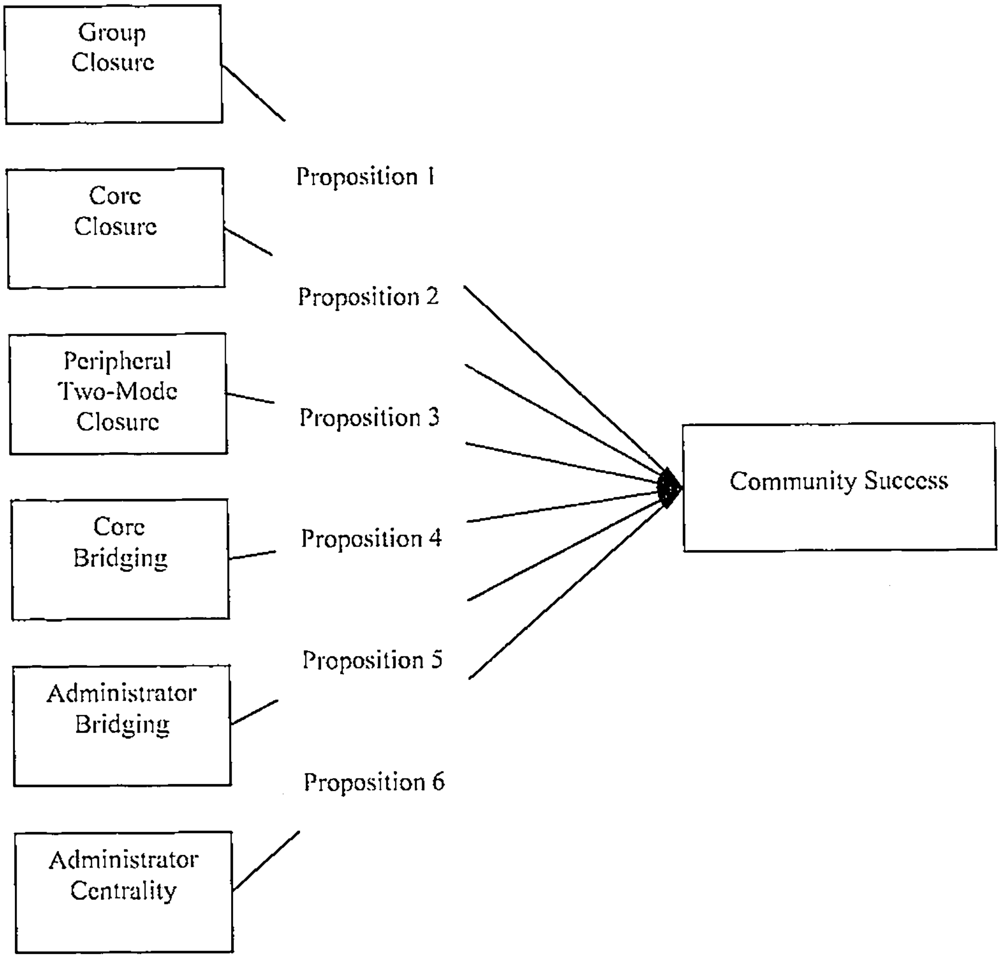
3.3. Social Network Model and Propositions

A social network model of success for open source software project communities is proposed as shown on Figure 6. The six social network constructs are shown on the left side of the figure, and the community success construct is shown on the right. A total of six propositions are derived. In the following sections, each of these propositions is described and the associated claims are justified based on the theoretical and conceptual foundations discussed in Chapter 2.

3.3.1. Group Closure

With respect to task performance, the design and production of software requires a certain level of interaction among the project community. Solving problems, integrating code contributions, and coordinating work require a certain extensiveness of discussion. However, the tools typically used by open source software project community members (e.g. version control systems, bug-trackers, etc.) act to reduce the level of direct interaction that is required. Further, the typical modular architecture of open source software projects is associated with a task design containing loosely coupled tasks and a limited need for interacting across modules. Therefore, a certain level of interaction is required, but only to a point.

Figure 6
Social Network Model of Community Success



The capacity of the project community to continue working together and to sustain itself depends upon the creation of a certain amount of trust among community members. One of the key positive effects that has been associated with closure is the facilitation of trust building (Coleman 1988). However, the open source software environment provides a relatively strong communal culture (Raymond 1999) and therefore a high level of closure may not be necessary because the members tend to share the strong values of the open source culture.

It is "costly" to develop and maintain ties. While the effort required to post a comment to a discussion forum seems to be trivial, consideration should be given to the time necessary to read and understand the content of previous forum posts and to start and maintain a dialogue with other members. Further, open source software projects, as defined in this research, involve volunteers who typically have a limited amount of time to contribute to the project. Thus, each additional tie that is established represents a cost to the actors involved and the group as a whole.

For the group as a whole, it is anticipated that a certain level of closure is required as described above. However, after a certain point, additional closure becomes a burden, it is distracting, and therefore it reduces the smooth functioning of the community. Closure above the required level will not incur further net benefits and so, the effect of the increasing cost of ties will be to reduce community success. Therefore, the relationship between Group Closure and Community Success is posited to be an inverted-U shape:

> *Proposition 1*
> The Group Closure of an open source software project community has an inverted-U relationship with Community Success. Community Success is maximized at a moderate level of Group Closure.

3.3.2. Core Closure

The subgroup of core developers is characterized by a higher level of activity than the peripheral developers. Most of the code for the project is created by these core developers. As a result, their need for interaction should be considerably greater than the needs of the group as a whole (which includes both the core and the periphery).

However, the cost-of-ties effect is still important and therefore, as with Proposition 1, an inverted-U shaped relationship is suggested:

> *Proposition 2*
> The Core Closure of an open source software project community has an inverted-U relationship with Community Success. Community Success is maximized at a moderate level of Core Closure.

3.3.3. Peripheral Two-Mode Closure

For peripheral developers, a greater level of two-mode closure between the peripheral subgroup and the rest of the group should lead to a greater sense of identification with the project, as well as feelings of satisfaction and challenge. As Raymond (1999) notes, it is important to "listen to the beta testers." Greater two-mode closure should translate into increased feelings of obligation and commitment to make contributions and to remain with the project. More connected peripheral developers are more likely to contribute code, bug reports, and assist with the production of the project software. These peripheral developers may be the source of new ideas and methods of development that could improve the group processes. Further, one or more may decide, at some point, to become core developers.

On the negative side, the cost-of-ties may become a significant factor as peripheral two-mode closure increases. Higher levels of connectedness with the core developers may become a distraction for these more active individuals, which may offset the benefits of having more motivated peripheral developers. Therefore, the relation between Peripheral Two-Mode Closure and Community Success is expected to have an inverted-U shaped relationship:

Proposition 3
The Peripheral Two-Mode Closure of an open source software project community has an inverted-U relationship with Community Success. Community Success is maximized at a moderate level of Peripheral Two-Mode Closure.

3.3.4. Core Bridging

There are various positive effects associated with bridging ties. The bridged members will have access to new ideas regarding production and design methods. In addition, these members may be able to bring in members from other projects on a one-time basis to solve particular problems and/or provide other special kinds of support. Bridging ties may also increase the likelihood of recruiting new project community members from other projects, as the focal actor utilizes his or her bridging ties to communicate the features of the focal project to potential members from other project communities. These effects result in additional resources which should help to improve task performance.

Bridging ties may also allow the focal actor to become aware of certain opportunities or threats to the focal project. For example, the bridged actor might identify a niche of related open source software projects which provides new opportunities for expanding the scope of the project to include interoperability with these other projects. Alternatively, the bridged actor might become aware of new or increased "competition" from other projects. In either event, the bridged actor may then be able to help guide the focal project through and around these environmental issues, which should lead to sustainable project community success.

In the case of bridging ties, the cost-of-ties effect is only a burden on the individual actor, because the ties are between that actor and the members of other project communities (not the focal community). However it is possible that too many bridging ties would result in a lack of time and attention given to the focal project by the bridging actor. Again, this is only one actor and the net negative effect of this on the overall project is expected to be minor. Therefore, the relationship between Core Bridging and Community Success is expected to be positive:

> *Proposition 4*
> The Core Bridging extent of an open source software project community is positively associated with Community Success.

3.3.5. Administrator Bridging

Considering the special influential position of administrators, it is possible that the bridging ties of administrators are the most important with regard to community success. In effect, the special position of administrators allows them to leverage the positive impact of their bridging ties. So, although the effect of administrator bridging may be stronger than for core bridging, a positive relationship is expected:

> *Proposition 5*
> The Administrator Bridging extent of an open source software project community is positively associated with Community Success.

3.3.6. Administrator Centrality

For administrators, a certain level of centrality is necessary in order for them to coordinate and integrate the work of the other developers. However, as the level of centrality increases, the administrators face the possibility of becoming overburdened and subject to "burn-out," which would have significant negative effects on both task performance and group viability. This is essentially the cost-of-ties effect observed at the individual level of the administrator. Pavlicek (2000) suggests that administrators should delegate as much as possible.

With regard to effects on the other (non-administrator) community members, again, a certain level of contact with administrators is valuable in that these members need to feel welcomed and accepted into the group. At a point, however, too much contact with the administrator subgroup can lead to a loss in the "feeling of ownership" that is apparently so important for open source software contributors. As noted by von Krogh (2003):

> Recent work by Karim Lakhani and Eric von Hippel and by Jae Yun Moon and Lee Sproull shows that contributors to open source software projects value a sense of ownership and control over the work product, something they do not experience in programming work carried out for hire. (von Krogh 2003)

Considering that an increase in administrator centrality will have a positive effect on success up to a certain point based on the involvement of administrators in task performance, and that further increases in centrality may have negative impacts with regard to excessive demands on the administrator and reduced motivation for the other members, the relationship between Administrator Centrality and Community Success is expected to be an inverted-U shape:

> *Proposition 6*
> The Administrator Centrality of an open source software project community has an inverted-U relationship with Community Success. Community Success is maximized at a moderate level of Administrator Centrality.

4. RESEARCH METHODOLOGY

This chapter includes a presentation of the study design and a description of the research setting. This is followed by a description and formal definition of the variables which operationalize the research constructs presented in Chapter 3. Finally, the sampling and data collection procedures are described and the resulting sample and research dataset is presented.

4.1. Study Design

A cross-sectional study design is chosen in which data are collected from a sample of open source software project communities. In the following sub-sections, the unit of analysis and study population are defined and the research method is discussed.

4.1.1. Unit of Analysis

The primary unit of analysis is the open source software project community. While it is possible to think of all open source developers as comprising a kind of community, the study definition of "project community" is limited to individuals associated with a particular project. Some data are collected at the lower level of community subgroup and even at the individual member level. However, social network analytical methods are then used to aggregate these results to the project community level.

4.1.2. Study Population

Considering the complex nature of open source software project communities and the various possible influencing factors (refer to section 3.1), a particular study population is defined in an attempt to control for some of these factors. In particular, the study population is limited to early-stage projects in order to control for project maturity, developer-targeted projects to control for project type and task structure, and corporate-sponsored projects are excluded to control for community norms and organizational environment.

Early-stage projects are identified by selecting those which have only two years of history following their first release of executable software. This study population definition results in a sample of projects that have similar age and developmental characteristics. In effect, these are all "start-up" or "early-stage" projects, which are viewed from a commonly defined starting point, regardless of the actual start date or current age of the project. The expected result is that the sample will be more homogeneous and represent a more focused group of projects, which will increase the likelihood of uncovering significant explanations of variance in the dependent variables.

With regard to developer-targeted projects, most prior studies of open source software projects have assumed the notion that "the user is also a developer," and have used the concept of "user-developer." However, many projects, such as Open Office, are targeted to end-users. While it is recognized that such projects exist, the developer-targeted project is accepted as the project type of focus in order to be consistent with prior studies and also to control for differences in project type. Therefore, end-user targeted projects are excluded from the study population.

Regarding community-founded projects, the salient view in most open source software research is the volunteer nature of the projects, and most of the motivational research has assumed this. However, it is increasingly recognized that many open source software contributors are paid by their companies to do the work (West and O'Mahoney 2005). It seems possible that this distinction would change the motivational factors and the underlying dynamics of the project. Therefore, the study definition of "open source software project" is limited to the traditional notion of a community-founded project. Corporate-sponsored projects and spin-off projects are excluded from the study population.

4.1.3. Research Method

The research method used is "analysis of existing statistics" (Babbie 2005). With this unobtrusive method, existing statistics and other types of historical records are the primary source of data. One advantage of this method is that there is no impact of the researcher on what is being studied. Another advantage is that data are not based on the perceptions of the research subjects, but the residuals of actual activity. A disadvantage is that certain reliability and validity problems are associated with this method (Webb et. al. 2000), as discussed further in section 7.3.

The other research method that could have been chosen to test the hypotheses is survey analysis. The analysis of existing statistics method was chosen for this study primarily because of the advantages of building social network variables from existing online discussion archives and project records. Such a method allows for a large number of networks to be sampled. This compares with laborious survey-based methods for creating social networks in which entire studies are typically devoted to studying one or a small number of networks. In addition, prior research has shown that subjects' perception of their social network is often quite different from their actual social network

(Krackhardt 1999). Calculation of social networks from archival data provides a representation of actual communications, and not merely perceived communications. Thus, having the ability to capture a large sample of objectively-created social networks is a relatively rare opportunity with respect to the study of groups.

In collecting existing statistics, a two-year observation window is utilized. The observation period begins with the date of first project release of executable software and ends at a point 24 months later. Even though data are collected over a period of time, a cross-sectional design is still utilized in that the entire two-year period is viewed as a single observational point. The two-year length of the window was chosen to provide a sufficiently long period for observing the formation of the relevant social networks and their effects on community success, without being so long as to be confounded by fundamental changes in the conditions in which the project operates. Open source software project life cycles on SourceForge are observed to range upwards of 7 or more years, and other projects, such as Linux, continue to mature after 15 years. During the project life cycle, various changes may occur in leadership or other conditions which change the nature of the social network structures. Also, the nature of the project undergoes a qualitative change as it grows substantially. Studies of the effect of social network structure on work group effectiveness have concluded that the intensity of the effect is reduced as the project matures and the group gains familiarity (Balkundi and Harrison 2006). Thus, this familiarity factor can mediate the relationship. Similar effects may occur in open source software project communities.

4.2. Research Setting

The research setting chosen for this study is the SourceForge hosting organization for open source software projects. On the SourceForge hosting site, individual projects are maintained and recognized based on a unique project name and a unique set of project web pages. Each project has at least one registered administrator who organizes and sets access privileges for the dedicated source code repository and public forum facilities which are made available by SourceForge. The project community members can be identified based on their registration with the project and/or by their participation in project forums.[7]

SourceForge is the largest and most diverse of the hosting platforms, with over 129,984 registered projects and 1,395,827 individual registered users[8]. Of these, 81,753 projects were registered with a valid "topic," and of these, a total of 35,231 were in a planning or pre-alpha stage, 39.145 were in an alpha or beta stage, 20,105 were in a production/stable or mature stage, and 1,968 were recorded as inactive, based on self-reported development status codes[9].

Other hosting platforms such as Savannah, Freshmeat, and others could have been selected. However, SourceForge was chosen in order to provide a uniform basis for sample selection and data collection, which has advantages both in terms of controlling for variations associated with the nature of the hosting platform and also in terms of

[7] It is recognized that individuals, sometimes referred to as "lurkers," may view the project pages and forum without posting to the forum or registering with the project. These individuals are not considered to be members of the project community for the purposes of this study.

[8] As of September 21, 2006

[9] Amounts do not total to 81,753 due to multiple codes being recorded for individual projects

logistical considerations. In effect, the choice of a single hosting organization may help to control for differences in community norms and organizational environment.

4.2.1. Data Sources

The SourceForge organization is the source of archival data. An intensive review of the SourceForge platform was performed to identify the availability of various data elements and to determine appropriate data extraction methods. Part of this review included the reading of SourceForge procedural documents and announcements to identify any situations or changes that might influence the integrity of the data on the site.

Data were acquired from SourceForge through two kinds of channels. One channel involves the direct capture of data (using cut-and-paste) from existing or archival project web sites[10]. The other channel involves acquiring access to and querying research databases which have been previously created by third parties based on data dumps from the SourceForge archives. The two research databases which were used in this study include the University of Notre Dame (UND) database and the Libresoft (LS) database.[11]

4.2.2. Data Element Selection

Based on a review of the various SourceForge data sources, various data elements were selected based on their availability and the extent to which they could be used in creating research variables to operationalize the previously defined research constructs. These variables, which are described in the following sections, were defined so as to logically and directly correspond with the associated constructs. Because an existing

[10] Selected web page screen images are contained in Appendix A.

[11] Descriptions of these research databases are contained in Appendices B and C.

statistics research method was selected, it was also necessary to consider both the availability and the integrity of the SourceForge data elements as these research variables were defined.

4.3. Dependent and Control Variables

In this section, the variables which operationalize the success construct dimensions of community output and community activity are defined and specified, along with the control variables that are used in the regression analyses.

4.3.1. Community Success

Six variables are defined for the community success dimensions of output and activity, all of which are calculated as the sum of the 24 monthly statistics which span the two-year observation window (Table 5). Three of these variables correspond with the output dimension and three correspond with the activity dimension. Each of these variables is described in the following paragraphs. Most of the community success variables are extracted from the UND research database, with the exception of the "code commits" variables which is extracted from the LS research database.

Community output variables. The community output variables consist of "code commits," "software releases" and "trackers closed." In producing software, developers normally work with a human-readable form known as "source code." Along with the first release of software, a production repository of the related source code is established and maintained on the host platform. As batches of new and/or improved source code are written and validated, these batches are entered (or "committed") into the source code repository. In creating the LS research database, the project source code repository records are examined and each commit is recorded along with its date. The variable "code commits" is a count of the number of these "commits" that are made over the two-year observation window.

Table 5
Community Success Variables

Variable Name	*Variable Description*	*Success Dimension*	*Data Source*	*Reference*
Code Commits	Number of source code commits	Ouput	SourceForge CVS records (LibreSoft database)	Healy and Schussman 2003
Software Releases	Number of software releases	Ouput	Project monthly statistical records (UND database)	Stewart and Ammeter 2002 Crowston, et. al. 2003
Trackers Opened	Number of closed trackers	Ouput	Project monthly statistical records (UND database)	Healy and Schussman 2003, Crowston, et. al. 2003
Software Downloads	Number of software downloads	Activity	Project monthly statistical records	Healy and Schussman 2003
Page Views	Number of page views	Activity	Project monthly statistical records	Healy and Schussman 2003
Trackers Closed	Number of opened trackers	Activity	Project monthly statistical records (UND database)	Healy and Schussman 2003, Crowston, et. al. 2003

At various points in time, based on the discretion of the administrators, the current production source code repository is "compiled" and a new release of executable software is made. This is essentially a working version of the software which can be used by developers or by non-technical users. Each release of this software is recorded in the project archives, and the variable "software releases" is a count of the number of such releases during the two-year window.

As community members identify the need for various kinds of changes to the software, the administrators may open a "tracker." These trackers are essentially work orders which specify requests from the community for development work, such as fixing a software bug or adding a functional feature. As the development work needed for a particular tracker is finished, the tracker is "closed." Each closed tracker is recorded in the project archives, and the variable "trackers closed" is a count of the number of trackers which are closed during the two-year window.

Community activity variables. The community activity variables consist of "software downloads," "page views," and "trackers opened." As software releases are made by the project administrators, new software versions are made available to the public. An individual who wishes to acquire and use this software is required to download the executable version from the project web site. Each such download is recorded in the project archives, and the variable "software downloads" is a count of the number of such download actions which occur during the two-year window.

The "page views" variable is measured by the number of times that any one of the project web pages are viewed. The project web pages include a home page, developer's page, and various other pages of interest to project developers and software users. The number of views which are made to these pages are recorded in the project archives, and the variable "page views" is a count of the number of such viewing actions which occur during the two-year window.

Finally, the variable "trackers opened" is defined as the count of the number of trackers which are opened during the two-year window (note "trackers closed" above).

The trackers opened variable is considered to be a measure of community activity because it reflects requests made by the entire project community and a greater level of downloading and page viewing should be associated with a greater level of tracker opening. As previously described, "trackers closed" is considered to be a measure of community output because the closing action occurs as the result of developmental work which is completed.

4.3.2. Controls

Previous studies have identified group size as having an effect on team effectiveness and this effect might also be expected in open source project communities. In addition, some social network variables, such as those involving density measurements, are sensitive to the total size of the group. Therefore, both group size and core size are used as controls. As noted on Table 6, "group size" and "core size" are defined as the number of project community members and the number of core subgroup members as of the midpoint in the two-year observation window.

Table 6
Control Variables

Variable Name	*Variable Description*	*Data Source*	*Calculation*
Group Size	Number of project community members	Project membership records (UND database)	Counted at mid-point of two-year observation window
Core Size	Number of core developer subgroup members	Project membership records (UND database)	Counted at mid-point of two-year observation window
Conversation Volume	Number of forum posts	Project monthly statistical records	Aggregated over two-year observation window

In addition, it is plausible that the success of the community could be related to the total volume of conversation, rather than the structure of the conversational network itself. Therefore, an additional control variable is defined to be "conversation volume," which is measured as the sum of the number of forum posts over the two-year observation window.

4.4. Social Network Variables

In this section, the networks and subgroups are defined and specified within the SourceForge research setting. A formal system of notation is defined and specified to include graph theoretic and sociometric notations. This notational system is used to define and formally specify the networks, subgroups, and the six social network variables which operationalize the six social network constructs described in Chapter 3.

4.4.1. Networks

The social network structural constructs defined in Chapter 3 are based on the information flow paradigm which is a fundamental premise of social capital theory. Therefore, an appropriate network definition for use in operationalizing these constructs would include links which are logically connected with information flow, as in a conversational connection or other form of communication.

Conversational network. Considering the availability of data from the SourceForge archives, a conversational network was defined based on data obtained from the project public forum records. Each project may have one or more public forums[12] on their SourceForge project site. Any SourceForge member can post an initial message to the forum. Individuals who view the forum can then respond with their own posts, resulting in a thread of discussion. While other forms of communication are recognized and certainly exist (direct emails, instant messaging, etc.), the norms of open source encourage the use of these transparent public forums and therefore the forum conversations were selected as a representative source of communicative connections between project members.

In defining the conversational network from public forum data, each node in the network is associated with a particular member of the project community, where a project community member is defined as an individual who has registered with the project or who has posted a comment to a project public forum. A link is then said to exist between two member-nodes if those two members participate in a single discussion thread on a project public forum during the two-year observation window. Crowston and Howison (2004) used a similar type of conversational network to study the social structural patterns of open source software projects by extracting textual data from bug report trackers.

Project membership network. The conversational network is adequate for calculating social network measures associated with the closure and leader centrality constructs because these constructs relate to conversations that occur within the project

[12] Public forums may be for general purposes (e.g. for "open discussion") or they may be designated for specific purposes (e.g. "user help").

community. However, the bridging constructs involve information flows that occur from inside the project community to individuals who are not part of the focal project community. Unfortunately, the SourceForge archives contained no public forums or other systematic data sources which could be used to calculate appropriate conversational measures for these external information flows. Therefore, cross-membership status was chosen as a proxy for such information flow and an appropriate project membership network was defined.

The defined project membership network consists of two types of nodes. One node type is specified to be a registered member of the focal project community. The other node type is defined to be a SourceForge project. A link between a member-node and a project-node is recognized if that particular individual is a member of that particular project. Therefore, the members of a focal project community will, by definition, have a link between their member-node and the focal project. However, if an individual is also a member of another SourceForge project, then a link is recognized between that individual and the other project. Gao, et. al. (2003) defined a similar type of project membership network in studying the connections between various open source software projects hosted by SourceForge. The key assumption in using this network for the calculation of bridging constructs is that membership in another project implies communication with members of that other project.

4.4.2. Subgroups

All of the social network structural constructs defined in Chapter 3, except for Group Closure, make reference to a particular subgroup[13] of the project community. Therefore, it is necessary to specify how subgroup membership is determined within the SourceForge research setting. The three subgroups of interest include core developers, peripheral developers, and administrators.

An individual is considered to be a core developer if that individual was formally registered with the focal project during the two-year observation window. An individual is recognized to be a peripheral developer if that individual posted a message to a project public forum during the two-year window (but was not formally registered with the project). Therefore, the core developer subgroup and the peripheral developer subgroup are mutually exclusive and exhaustive subsets of the set of members comprising the project community. An individual is considered to be an administrator if that individual is formally registered as an administrator with the focal project on the SourceForge records. Because registered administrators are also registered members, the administrator subgroup is a subset of the set of members comprising the core developer subgroup.

4.4.3. Formal Notation

In this subsection, the application of graph theory and sociometric notation to social network analysis is briefly reviewed, followed by a discussion of the basic concepts and notational systems that are relevant to the work. In general, the notational

[13] A "subgroup" is defined based on the a priori individual attributes of the subgroup members. This is in contrast to the typical notion of "subgroup" in social network analysis, in which the subgroup is defined by certain structural attributes using methods such as block modeling or hierarchical clustering (Wasserman and Faust, 1994).

conventions used by Wasserman and Faust (1994) are followed. In addition, definitions for one-mode and two-mode networks are provided as needed for this work.

Graph theory, a branch of mathematics, has been used extensively for modeling social systems including applications in anthropology, social psychology, communications, business, organizational research and geography (Wasserman and Faust 1994). For social network analysis, graph theory provides a useful vocabulary and a set of primitive concepts for representing social networks. It is also associated with visual representations which have proven to be valuable in helping to understand network concepts.

Sociometric notation was first introduced by Moreno (1934) and is perhaps the most widely used and practical notational system for social network analysis (Wasserman and Faust 1994). It can be used by itself or combined with graph-theoretic notation in describing social networks. In addition, most social network analysis software packages use a sociometric representation and take advantage of matrix algebra for network data manipulation and calculation of social network analytical measures. In the following formal network representations, both graph-theoretic and sociometric notations[14] are utilized.

One-mode[15] network. In Chapter 2, a social network was described as a network representation in which the nodes of the network are social entities and the links of the

[14] In defining the networks and graphs, we assume that there is only one relation in any given graph and that this relation is dichotomous and nondirectional. Consideration of multiple, valued, and/or directional relations is possible but is unnecessary for the purposes of this research.

[15] A "mode" is a type of node. Refer to "two-mode network".

network are relations between the social entities. Using graph theoretical notation, a social network can be more formally defined as consisting of a node set, a line set and a relation, whereby the node set includes all actors who are within the group of interest (e.g. the focal project community), and the line set includes all pairs of actors from the node set for which the relation applies (e.g. members who co-participate in a discussion thread).

An actor is denoted as "n" and the "node set" is defined as a set *N* which contains a total of g actors:

$N = \{n_1, n_2, ... n_g\}$.

A nondirectional relation is defined which may or may not exist between any two actors, whereby an unordered actor pair for which the relation exists is denoted as line "l", and the "line set" is defined as a set *L* which contains a total of L lines:

$L = \{l_1, l_2, ... l_L\}$.

Using the above graph theoretic notations, a complete specification for a nondirectional one-mode network (graph) can now be presented, as denoted by ***G***, where ***G*** contains set *N* and set *L*.

A sociometric notational definition of a social network begins with the same graph theoretic notation of a set *N* which contains g actors. However, instead of using the concept of a line set *L*, a sociometric approach is taken to define the actor pairs connected by a relation to be the cells of a matrix. Thus, a sociometric matrix **X** is defined on a single relation over the set of g actors in which the value of the matrix cell

x_{ij} is "1" if the relation exists between actor n_i and actor n_j, and "0" if the relation does not exist between actor n_i and actor n_j. For a nondirectional one-mode network, then, **X** is a symmetrical g x g matrix and it completely specifies the network.

Two-mode network. The above concepts are now extended to define a "two-mode network" in which two different node sets are permitted. For a two-mode network, a mode-1 actor is denoted as "n" and a mode-2 actor as "m", and two mutually exclusive node sets *N* and *M* are defined to contain a total of g and h actors respectively:

$N = \{n_1, n_2, ... n_g\}$, $M = \{m_1, m_2, ... m_h\}$.

The two sets *N* and *M* may contain actors which are of the same type, or set *N* may contain actors which are of a different type than those contained in set *M*. The associated sociometric matrix **X** is not square, but rather is rectangular and of dimension g x h, where each matrix row is associated with a unique actor "n" and each matrix column is associated with a unique actor "m". One special kind of two-mode network is an "affiliation network" in which *N* contains a set of actors and *M* contains a set of events or organizational entities, and the relation is defined by the affiliation of the actors with the event-organizations.

4.4.4. Formal Specification

As shown on Table 7, each social network structural construct defined in Chapter 3 is operationalized with a particular social network variable. The table indicates the construct name, variable name, data source, and reference in the social network analysis literature. Using the notational system defined in the previous subsection, definitions are first presented for the social network measures of "density," "nodal degree," "mean nodal degree," and "standardized actor degree centrality." These definitions are then used in defining the formal specification for each of the six social network variables below.

Table 7
Social Network Variables

Construct Name	*Variable Name*	*Data Source*	*Social Network Analysis Reference*
Group Closure	Group density	Public forums	Wasserman and Faust 1994
Core Closure	Core density	Public forums, Project membership records	Wasserman and Faust 1994
Peripheral Two-Mode Closure	Peripheral two-mode density	Public forums, Project membership records	Borgatti, et. al. 1998 Wasserman and Faust 1994
Core Bridging	Core membership degree	SourceForge membership records, Project membership records	Wasserman and Faust 1994
Administrator Bridging	Administrator membership degree	SourceForge membership records, Project membership records	Wasserman and Faust 1994
Administrator Centrality	Administrator class centrality	Public forums, Project membership records	Everett and Borgatti 1999

Density. The "density" of a graph is the actual number of lines in a graph as a proportion of the total possible number of lines in the graph. Denoting density as Δ, the calculation for density is specified by the formula:

$$\Delta = 2L / g(g-1).$$

Nodal degree. The "nodal degree" of a node n_i, denoted by $d(n_i)$, is the number of lines that are incident with the node n_i (Wasserman and Faust 1994). A node is incident with a line if that node is one of the unordered pair of nodes which defines the line (Wasserman and Faust 1994). Using sociometric notation, nodal degree is defined for a one-mode network as:

$$d(n_i) = \sum_{\text{all } j} x_{ij} = \sum_{\text{all } i} x_{ij}$$

The nodal degree for the mode-1 actors in an affiliation network is defined as:

$$d(n_i) = \sum_{\text{all } j} x_{ij}$$

Mean nodal degree. The "mean nodal degree" of a graph, denoted by $d^{\wedge}$, is the average nodal degree for all nodes in the network. Applied to the actors in an affiliation network, mean nodal degree is:

$$d^{\wedge} = \sum_{\text{from } i=1 \text{ to } g} d(n_i) / g = 2L / g.$$

Standardized actor degree centrality. The "standardized actor degree centrality" of a node n_i, denoted by $C'_D (n_i)$, is defined as:

$C'_D (n_i) = d(n_i) / (g-1)$. (Wasserman and Faust 1994)

The general social network measures defined above are now used in defining the specific social network variables to be used in this research.

Group density. The "group density" (GD) is the density of the "total conversation network," which is a one-mode network where actors are members of the focal project community and the relation is forum conversation.

Core density. The "core density" (CD) is the density of the "core conversation network," which is a one-mode network where the actors are members of the core subgroup of the focal project community and the relation is forum conversation[16].

Peripheral two-mode density. The "peripheral two-mode density" (PTD) is the density of the "periphery-core conversation network," which is a two-mode network where the mode-1 actors are members of the peripheral subgroup, the mode-2 actors are members of the core subgroup, and the relation is forum conversation which is only defined for actor pairs containing one core actor and one peripheral actor. Centralization of the total conversation network was considered as a candidate for operationalizing the Peripheral Two-Mode Closure construct. However, peripheral two-mode density was chosen instead because it takes advantage of the explicit definition of the core and peripheral subgroups, while centralization implicitly defines a core-periphery structure using network properties.

Core membership degree. The "core membership degree" (CMD) is the mean nodal degree (defined for an affiliation network) for all actors in the "core project membership network," which is an affiliation network where the actors are core subgroup

[16] This is a node-generated subgraph of the total conversation network graph (Wasserman and Faust 1994).

members of the focal project community, the events are SourceForge projects, and the relation is project membership. Class centrality measures (Everett and Borgatti, 1999) could also have been used to operationalize the bridging constructs. However, the decision was made not to process the entire SourceForge membership network and therefore the average degree measure was selected because it only requires the collection of project membership data for the focal project actors.

Administrator membership degree. The "administrator membership degree" (AMD) is the mean nodal degree (defined for an affiliation network) for all actors in the "administrator project membership network", which is an affiliation network where the actors are administrator subgroup members of the focal project community, the events are SourceForge projects, and the relation is project membership.

Administrator class centrality. The "administrator class centrality" (ACC) is the standardized actor degree centrality of the super-node in the "administrator-other conversation network[17]," which is a special type of two-mode network (Everett and Borgatti 1999) where the administrator subgroup members are represented as a single mode-1 "super-actor," the mode-2 actors are the other members of the focal project community, and the relation is forum conversation which is only defined for actor pairs containing the single super-actor and a mode-2 actor[18]. Degree centrality was chosen

[17] If the super-node contains only one actor, then administrator class centrality is equivalent to standardized nodal degree centrality for the one actor.

[18] In this definition, the effect of the mode-1 "super-actor" is that ties from a single mode-2 actor to multiple members of the administrator subgroup are counted only once.

over other possible centrality measures such as closeness or betweenness because it is a well-tested measure and there is no compelling reason to make other choices.

4.5. Sampling and Data Collection

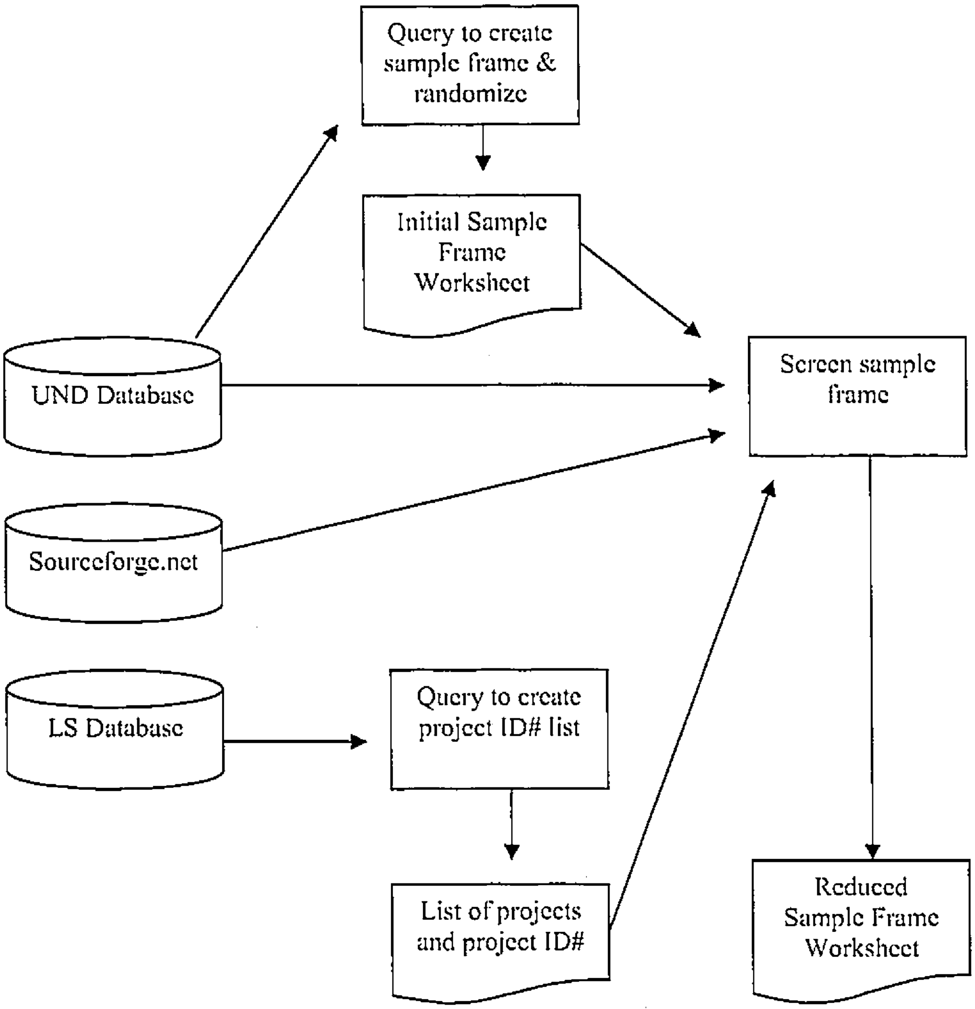
In this section, the overall sampling and data collection process is described. This process involves a series of data extraction, screening, and compilation procedures which were used to create a sample frame. This frame is then screened for conformance with study population, data availability and data integrity criteria. The screened sample frame is then used for selecting a sample of projects for which the appropriate data elements are extracted and research variables are computed, resulting in a research dataset to be used in the analysis phase (described in Chapter 5).

4.5.1. Sample Frame

The sampling strategy was to use the UND database to select either a probability sample or a complete sample (Babbie 2005). The probability sampling method is close to random sampling when the sample frame contains no systematic bias (Babbie 2005). The target sample size is 200 project communities.

As noted on Figure 7, an initial sample frame was created by querying the UND database for January 2006. This month was chosen because it was the most recent month for which data was also available from the LS database. The query script includes a selection for projects which conform to the study population definitions for target audience and project maturity (Table 8). This query also extracted certain data, such as open source license used, which were useful for profiling the selected sample. The initial sample frame was recorded on the Initial Sample Frame Worksheet, and it contained 934 project communities.

Figure 7
Sample Frame Development Workflow


Query to create sample frame & randomize
Initial Sample Frame Worksheet
UND Database
Screen sample frame
Sourceforge.net
LS Database
Query to create project ID# list
List of projects and project ID#
Reduced Sample Frame Worksheet

The initial sample frame was randomized in preparation for the probability sampling procedures included in the data compilation process. The Initial Sample Frame

Worksheet was sorted alphanumerically by project name, and this sorted list was used to apply a systematic sample with a random start (Babbie 2005). With this approach, an initial position is randomly chosen within the list and every nth project after that starting point is selected for possible inclusion in the sample.

Table 8
Project Selection Criteria

Criteria Category	*Test Criteria ("Reject if...")*	*Application Step*
Study Population	Evidence is found of corporate ownership or sponsorship Project type is not developer oriented First release date is less than 2 years prior to query date Only one core member is found	Screening Frame Query Frame Query Compilation
Data Availability	Administrators allow anonymous forum postings Public forums contain less than 50 posts during 2-year window Libresoft Project ID# not available All commit values are zero	Screening Screening Screening Compilation
Data Integrity	Evidence is found of ambiguity in date of first software release Evidence if found of data corruption in monthly statistics	Screening Screening

This initial frame is then screened for compliance with additional study population criteria, and is subjected to various tests for data availability and data integrity (Table 8). This resulted in a reduced sample frame which was recorded on the Reduced Sample Frame Worksheet, and it contained 257 project communities. The screening procedures were performed by the author and reliability was verified by a third party. The randomized and reduced sample frame was then passed to the data compilation process, which is described in the next sub-section.

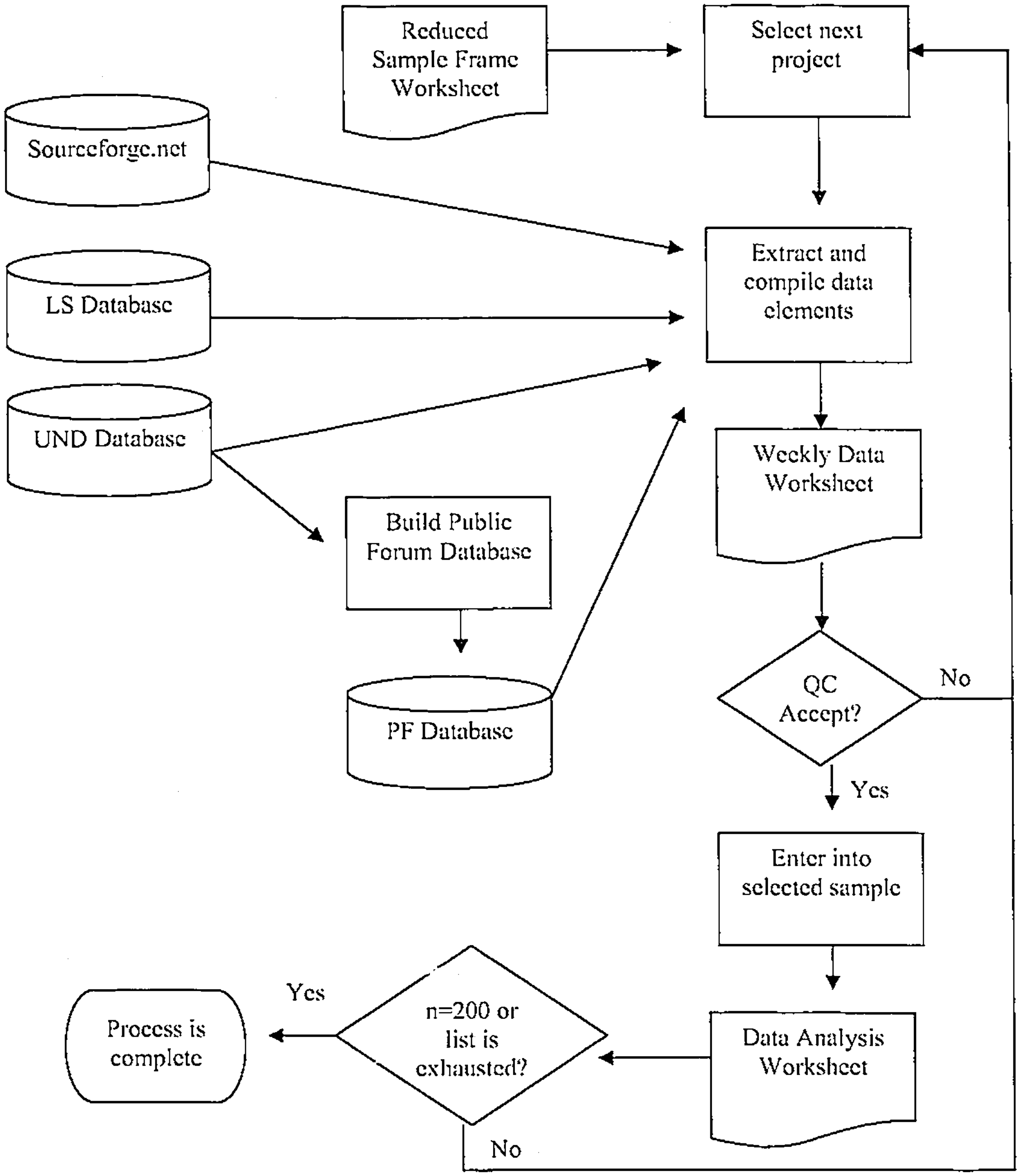
4.5.2. Data Compilation

In compiling the extracted data and computing the research variable values, Microsoft Excel was used for data manipulation and UCINET social network analysis software was used for the manipulation of network data and the calculation of social network measures. In the process, Microsoft Access was used to build a secondary database which contains the public forum data extracted from the UND database. The Access scripts to create and use this database were created by a third party and validated by the author using independent compilation methods. All other query scripts were created and validated by the author.

As noted on Figure 8, the compilation process begins with the randomized Reduced Sample Frame Worksheet (from Figure 7). The first project is selected at random from this worksheet and appropriate data items are extracted and compiled onto a Weekly Data Worksheet. This compilation process was performed by the author as well as by a third party who was closely supervised by the author. The results were checked by the author and/or by another third party for accuracy and compliance with compilation procedures. Cases for which errors were found were returned for reworking. Other cases for which no errors were found were entered into the final sample on the Data Analysis Worksheet. Another case was selected from the Reduced Sample Frame Worksheet and the process was repeated.

This process was to continue until either a total of 200 cases were included in the selected sample or the reduced sample frame was exhausted. Based on the 257 cases in the reduced sample frame and the application of additional study population and data availability tests (Table 8), the sample frame was exhausted, resulting in a complete sample consisting of 160 projects. The associated Data Analysis Worksheet represented the research dataset to be used for analysis.

Figure 8
Data Compilation Workflow



4.5.3. Sample Profile

Profile statistics for the selected sample are shown on Table 9. All statistics shown on the table were extracted at or near the midpoint of the two-year observation windows for each project. The year of first software release was spread fairly evenly across the sample, ranging from the Year 2000 to the Year 2003. A number of projects were initiated in 1998 and 1999. However, the data for these projects were corrupted or no longer available. The most frequently self-reported status levels of development were Beta and Production, accounting for over 70 percent of the project communities. The most common open source license used was the GPL (GNU General Public License) accounting for 58% of the total, followed by the LGPL (GNU Lesser General Public License) and the BSD (Berkeley Software Distribution) license which accounted for 13% and 9% respectively.

Table 9
Profile Statistics for Sampled Project Communities
(n = 160)

	No. projects	% of total
Year of first release:		
1999	1	0.6
2000	34	21.3
2001	45	28.1
2002	41	25.6
2003	39	24.4
Total	160	100.0
Project development status:		
1 Planning	2	1.3
2 Pre-alpha	11	6.9
3 Alpha	27	16.9
4 Beta	57	35.6
5 Production	58	36.3
6 Mature	5	3.1
7 Inactive	0	0.0
Total	160	100.0
Open source license used:		
GPL	93	58.1
LGPL	21	13.1
BSD	15	9.4
Apache	7	4.4
Other	24	15.0
Total	160	100.0

5. DATA ANALYSIS AND RESULTS

This chapter includes a description of the preliminary analyses which included normality testing and variable transformation, outlier assessment and removal, and factor analysis and variable reduction. Descriptive and correlation statistics are then presented. Finally, the regression procedures that were applied in testing the hypotheses are described, the hypotheses are listed, and the testing results are reported, including both linear and quadratic analyses[19].

5.1. Preliminary Analyses

Prior to performing regression analyses, a series of preliminary statistical and analytical procedures were applied to the research dataset associated with the sampled project communities. The distributions of the variables were first checked for normality and based on the findings, the dependent variables were log transformed. Outlier tests were then performed on the transformed variables including both univariate and multivariate procedures. This resulted in the removal of 17 cases. In the final preliminary step, possible reductions of the transformed dependent variables were considered based on the appliation of a factor analysis method. This step resulted in the removal of 2 of the 6 variables, resulting in a total of 4 community success variables to be used for hypothesis testing.

[19] The statistical data analyses presented in this chapter were performed using the software package: SPSS for Windcws, Version 14.0.

5.1.1. Transformation of Variables

An initial test of normality was performed for all research variables and high levels of skewness and non-normality were found in most of the variables. In order to rely on the results of a linear regression test, it is important that the standardized residuals resulting from the regression exhibit a normal distribution (Allison 1999), and a non-normal result is often associated with non-normality of the dependent variables. Preliminary linear regression analyses were performed between each dependent variable and each independent variable and, as expected, non-normality was noted in the standardized residuals.

The dependent variables were then transformed using a natural logarithmic function. Normality was then tested using a Kolmogorov-Smirnov statistic with a Lilliefors significance level, based on the null hypothesis that the standardized residuals are normally distributed. A significance level of less than .05 is taken as a rejection of the null hypothesis and an indication that the values have a non-normal distribution (Mertler and Vannatta 2005). The normal Q-Q plots were also inspected for each variable to check for a straight-line appearance which is an indication of normality (Mertler and Vannatta 2005). The results, which are shown on Table 10, indicate that the log transformation resulted in evidence of normality based on the Kolmogorov-Smirnov statistics and the normal Q-Q plots for all 6 variables.

The preliminary linear regression analyses for each dependent variable and each independent variable were repeated and it was observed that the standardized residuals passed the Kolmogorov-Smirnov test for normality for 4 of the 6 dependent variables. For the other 2 variables (Software Releases and Trackers Closed), the Kolmogorov-

Smirnov statistic was marginal but the normal Q-Q plots for these variables showed a reasonable straight line appearance. Therefore, the log transformed versions of the dependent variables were accepted for hypothesis testing. No further transformations of the independent or control variables were considered because normality of the standardized residuals was achieved with these variables in an untransformed state. In conducting the regression runs for the actual hypothesis testing, the normality of the standardized residuals was verified, as described in Section 5.3.

Table 10
Normality Tests of Dependent Variables

	Untransformed Variables			*Transformed Variables*		
Dependent Variable	*Kolmogorov-Smirnov Z-Statistic*	*Sig. Level*	*Shape of Normal Q-Q Plot*	*Kolmogorov-Smirnov Statistic*	*Sig. Level*	*Shape of Normal Q-Q Plot*
Code Commits	3.799	.000	Nonlinear	0.569	.902	Linear
Software Releases	3.538	.000	Nonlinear	1.065	.206	Linear
Trackers Closed	4.324	.000	Nonlinear	0.950	.328	Linear
Software Downloads	5.193	.000	Nonlinear	0.903	.389	Linear
Page Views	4.409	.000	Nonlinear	0.938	.343	Linear
Trackers Opened	4.145	.000	Nonlinear	0.691	.725	Linear

It was noted that in cases where the dependent variable ("y") had a zero value, the log transformed version of the variable - ln(y) - was undefined which resulted in a missing value for ln(y). Because the limit of ln(y) is zero as "y" approaches zero, it is reasonable to fill in the missing values for ln(y) with a "0". Therefore, the missing values were filled in accordance with this method.

5.1.2. Outlier Assessment

Outliers are cases which involve extreme values for one or more research variables. Generally, outliers are defined as values which are three or more standard deviations away from the mean value for the variable. This criterion was used for assessing univariate outliers in which the extreme values are tested for each variable individually. Based on this assessment, a total of 13 cases were identified in which this criterion was met for log transformations of the dependent and control variables. Mahalanobis distance measures were then used to check for multivariate outliers in which extreme values of the dependent variable are found for particular combinations of the independent variables. Multivariate outliers are determined if a chi-square statistic for the Mahalanobis distance is significant at $p < .001$ (Mertler and Vannatta 2005). An additional 4 cases were identified which met this criterion. Considering both univariate and multivariate situations, a total of 17 outlier cases were eliminated, resulting in an adjusted total of 143 cases.

5.1.3. Reduction of Variables

A factor analysis method can be used to test for measurement overlap among the dependent variables, and the results can be used for reducing the number of variables in total and for grouping them into dimensions or components. As described in Section 3.1, the research model defines success along the two dimensions of output and activity. As described in Section 4.3.1, the logical assessment of the 6 dependent variables led to the conclusion that Code Commits, Software Releases, and Trackers Closed are measures of the output dimension and that Software Downloads, Page Views, and Trackers Opened are measures of the activity dimension.

A factor analysis method was applied to assess the plausibility of the two-dimension model, as well as the logical assignments of the variables to the two dimensions. In addition, the factor analysis method was used to assess if any dependent variables can be removed in order to reduce the level of redundancy among these variables. The factor analysis was applied to the log transformed versions of the dependent variables because these were selected for inclusion in the linear regression testing. In performing the factor analysis, an exploratory approach was initially taken (Allison 1999). A principal component analysis was applied with a varimax rotation. Four criteria are used in determining the appropriate number of components to be retained, including eigenvalue, variance explained, screen plot, and residuals.

On the first factor analysis run, all 6 dependent variables were analyzed with component extraction based on eigenvalues greater than "1". This run produced two components with 4 variables loaded onto component #1 (Code Commits, Software Releases, Trackers Opened and Trackers Closed), and 2 variables onto component #2 (Software Downloads and Page Views) loaded. However, the variance explained was marginal (69.9%) and the scree plot and residual criteria suggested the need for an additional component.

A second run was then conducted in which a third component was forced. The result of this run was that Trackers Opened and Trackers Closed loaded onto component #1, Software Downloads and Page Views loaded onto component #2, and Code Commits and Software Releases loaded onto component #3. With this run, the eigenvalue criterion was not met. In addition, the component groupings are inconsistent with the output and activity dimensions. The Trackers Opened and Trackers Closed variables should logically be split between the two dimensions. Apparently their excessively high correlation (Pearson correlation = .86) which results from their logical connection (a tracker cannot be closed unless it is first opened) causes this inconsistent result.

The Trackers Opened and Trackers Closed variables were eliminated and a third run was performed which included the other 4 variables. An eigenvalue selection criterion was used which resulted in two components in which Software Downloads and Page Views loaded onto component #1 and Code Commits and Software Releases loaded onto component #2 (Table 11). All four criteria were met suggesting that no additional components were necessary. Also, the result is logical and intuitive. Therefore, Trackers Opened and Trackers Closed were eliminated from further consideration.

Table 11
Rotated Component Loadings for Accepted Dependent Variables
(Log Transformed Dependent Variables)

Dependent Variable	*Component #1*	*Component #2*
Code Commits	.170	.839
Software Releases	.074	.868
Software Downloads	.918	.129
Page Views	.923	.128

5.2. Descriptive and Correlation Statistics

Descriptive statistics were generated for each of the research variables. As noted on Table 12, the total size of the project communities ranged from 7 to 326 members with an average size of about 67 members. The total project community consisted of core

Table 12
Descriptive Statistics of Subgroups and Research Variables
(n = 143)

	Unit	Min.	Max.	Mean	S.D.
Subgroups:					
Peripheral developers	# members	2	313	61.8	62.6
Core developers	# members	2	21	5.4	3.9
Administrators	# members	1	8	2.0	1.4
Controls:					
GS Group Size	# members	7	326	67.3	63.0
CS Core Size	# members	2	21	5.4	3.9
CV Conversation Volume	# 2yr posts	50	3,258	326	451
Community Success:					
Code commits	# 2yr commits	50	43,594	2,336	4,511
Software releases	# 2yr releases	0	79	11.3	11.1
Software downloads	# 2yr downloads	758	222,510	23,893	35,910
Page views	# 2yr page views	4,825	1,243,073	165,180	227,992
Transformed Success Variables:					
CC Code Commits	Ln # 2yr commits	3.91	10.68	6.87	1.37
SR Software Releases	Ln # 2yr releases	0.00	4.37	2.01	0.97
SD Software Downloads	Ln # 2yr downloads	6.63	12.31	9.40	1.17
PV Page Views	Ln # 2yr page views	8.48	14.03	11.35	1.16
Community Social Network Structure:					
GD Group Density	0-to-1 index	.006	.429	.078	.074
CD Core Density	0-to-1 index	.000	1.000	.288	.357
PTD Periph. Two-Mode Density	0-to-1 index	.000	.642	.210	.155
CMD Core Membership Degree	# projects	1.00	7.20	2.03	0.99
AMD Admin. Membership Degree	# projects	1.00	11.50	2.49	1.82
ACC Admin. Class Centrality	0-to-1 index	.000	1.000	.554	.282

developers and peripheral developers, where the core subgroup ranged from 2 to 21 members with an average size of 5.4 members, while the peripheral subgroup ranged from 2 to 313 members with an average size of about 62 members. The average number of administrators per project community was 2.0, and 49 percent of the communities had only 1 administrator.

The average volume of public conversation in the two-year observation period was 326 posts, with a range from 50 to 3,258 posts. The average values for the community success variables (calculated over the two-year period) included 2,336 code commits, 11.3 software releases, about 24,000 downloads and about 165,000 page views. Of course, the mean values for the log transformed versions of these success variables were much lower, ranging from an average of 2.01 for the log of software releases to an average of 11.35 for the log of page views.

Four of the 6 social network structure variables are defined as "0-to-1 indexes." Of these, both Core Density and Administrator Class Centrality ranged from .000 to 1.000, with average values of .078 and .554 respectively. The Group Density variable ranged from .006 to .429 with an average value of .078, while Peripheral Two-Mode Density ranged from .000 to .642 with an average value of .210. For Core Membership Degree, the core subgroup members were found to be registered with an average of 2.03 projects per member, while the administrator subgroup members were registered with a slightly higher average of 2.49 projects per member. These values included the member link with the focal project.

It is interesting to note that the average size and range for the core subgroups was somewhat similar to the average size and range of the teams that were investigated in the

37 studies reviewed by Balkundi and Harrison (2006). Across the reviewed team studies, the average team size was 8, with a range of 3 to 15. This compares with the result for open source software project core subgroups which had an average size of 5.4, and a range of 2 to 21.

The matrix of Pearson correlation statistics for the research variables is presented on Table 13. As might be expected, the highest correlation value was noted between Core Membership Degree and Administrator Membership Degree (.828). High correlations were noted between the log transforms of the two community activity variables, Software Downloads and Page Views (.729), and between two of the density measures, Core Density and Peripheral Two-Mode Density (.714).

5.3. Hypothesis Testing

In this section, a set of testable hypotheses is derived followed by a presentation of the testing procedures that were performed.

5.3.1. Research Hypotheses

In this section, a set of four testable hypotheses is dervied for each proposition suggested in Chapter 3. Each hypothesis represents the relevant social network structural variable in combination with one of the four community success variables. The 24 resulting hypotheses are listed below:

Table 13
Correlation Matrix of Research Variables

	GS	CS	CV		GD	CD	PTD	CMD	AMD	ACC		Ln CC	Ln SR	Ln SD	Ln PV
GS															
CS	.132														
CV	.775	.025													
GD	-.520	-.112	-.280												
CD	.039	-.395	.267		.170										
PTD	-.168	-.551	.148		.331	.714									
CMD	-.092	-.116	-.116		-.072	-.023	.085								
AMD	-.067	-.047	-.105		-.032	-.058	-.002	.828							
ACC	-.194	-.294	.122		.320	.431	.692	-.037	-.067						
Ln CC	.002	.143	.102		-.099	-.134	-.026	.039	.007	.117					
Ln SR	.082	-.108	.157		-.140	-.058	.066	-.021	-.052	.274		.486			
Ln SD	.548	.101	.330		-.559	-.148	-.250	.052	.039	-.211		.215	.226		
Ln PV	.506	.219	.313		-.505	-.137	-.233	-.010	-.004	-.226		.284	.161	.729	

GS = Group Size
CS = Core Size
CV = Conversation Volume
GD = Group Density
CD = Core Density
PTD = Peripheral Two-Mode Density
CMD = Core Membership Degree
AMD = Administrator Membership Degree
ACC = Administrator Class Centrality
Ln_CC = Log Transform of Code Commits
Ln_SR = Log Transform of Software Releases
Ln_SD = Log Transform of Software Downloads
Ln_PV = Log Transform of Page Views

Hypothesis 1a
The Group Density of an open source software project community has an inverted-U relationship with Code Commits.

Hypothesis 1b
The Group Density of an open source software project community has an inverted-U relationship with Software Releases.

Hypothesis 1c
The Group Density of an open source software project community has an inverted-U relationship with Software Downloads.

Hypothesis 1d
The Group Density of an open source software project community has an inverted-U relationship with Page Views.

Hypothesis 2a
The Core Density of an open source software project community has an inverted-U relationship with Code Commits.

Hypothesis 2b
The Core Density of an open source software project community has an inverted-U relationship with Software Releases.

Hypothesis 2c
The Core Density of an open source software project community has an inverted-U relationship with Software Downloads.

Hypothesis 2d
The Core Density of an open source software project community has an inverted-U relationship with Page Views.

Hypothesis 3a
The Peripheral Two-Mode Density of an open source software project community has an inverted-U relationship with Code Commits.

Hypothesis 3b
The Peripheral Two-Mode Density of an open source software project community has an inverted-U relationship with Software Releases.

Hypothesis 3c
The Peripheral Two-Mode Density of an open source software project community has an inverted-U relationship with Software Downloads.

Hypothesis 3d
The Peripheral Two-Mode Density of an open source software project community has an inverted-U relationship with Page Views.

Hypothesis 4a
The Core Membership Degree of an open source software project community is positively associated with Code Commits.

Hypothesis 4b
The Core Membership Degree of an open source software project community is positively associated with Software Releases.

Hypothesis 4c
The Core Membership Degree of an open source software project community is positively associated with Software Downloads.

Hypothesis 4d
The Core Membership Degree of an open source software project community is positively associated with Page Views.

Hypothesis 5a
The Administrator Membership Degree extent of an open source software project community is positively associated with Code Commits.

Hypothesis 5b
The Administrator Membership Degree extent of an open source software project community is positively associated with Software Releases.

Hypothesis 5c
The Administrator Membership Degree of an open source software project community is positively associated with Software Downloads.

Hypothesis 5d
The Administrator Membership Degree of an open source software project community is positively associated with Page Views.

Hypothesis 6a
The Administrator Class Centrality of an open source software project community has an inverted-U relationship with Code Commits.

Hypothesis 6b
The Administrator Class Centrality of an open source software project community has an inverted-U relationship with Software Releases.

Hypothesis 6c
The Administrator Class Centrality of an open source software project community has an inverted-U relationship with Software Downloads.

Hypothesis 6d
The Administrator Class Centrality of an open source software project community has an inverted-U relationship with Page Views.

5.3.2. Regression Methods

A multiple linear regression with ordinary least squares (Tabachnick and Fidell 2007) was used as the primary statistical testing method. For each hypothesis, the relevant DV is regressed on the relevant IV. Control variables are included and tests are performed for both linear and quadratic (inverted-U or U-shaped) relationships. The quadratic test involves a transformation of the IV in which the IV is mean-centered and squared (Allison 1999).

Because it is plausible that group size, core size, and/or conversational volume may be positively related to community success, associated variables were defined and applied as controls in every regression (refer to Section 4.3.2 for definitions). The purpose of this approach is to isolate the effects of the independent variable from the effects of the control variables. In this way, the resulting explanation of variance in the dependent variable is incremental and does not reflect effects associated with control variables.

A single three-step hierarchical regression test is applied which incorporates the control variables, the linear testing, and the quadratic testing. The first step is a regression of DV on the three control variables ("model 1"). The second step is the regression of the DV on the three control variables and the relevant IV ("model 2"). The third step is the regression of the DV on the three control variables, the relevant IV, and the relevant transformed (mean-centered and squared) IV.

To support an inverted-U relationship, the coefficient estimates for the untransformed IV (in model 2) should be positive and the coefficient estimates for the transformed IV (in model 3) should be negative and have a significant p-value. In addition, model 3 should result in a significant change in the level of explained variance, as measured by a significant F statistic for the change in R-squared from model 2 to model 3. This quadratic method may also support a U-shaped relationship based on the same criteria as described above except that the coefficient signs are reversed (i.e. the model 2 coefficient is negative and the model 3 coefficient is positive).

The appropriate application of multiple linear regression requires the satisfaction of certain assumptions. The testable assumptions include normality, homoscedasticity, and linearity. It is also appropriate to look for multicollinearity among the IVs. In the following paragraphs, the procedures that were used to test for these situations are described and the results of this application are reported.

Normality. The normality of all variables was tested and a necessary transformation of the DVs was made as reported in Section 5.1.2. In addition, the normality of the standardized residuals in each regression run was tested using a Kolmogorov-Smirnov statistic with a Lilliefors significance level, based on the null hypothesis that the standardized residuals are normally distributed. A significance level of less than .05 is taken as a rejection of the null hypothesis and an indication that the values have a non-normal distribution (Mertler and Vannatta 2005). No Lilliefors significance levels were less than .05, and therefore no indication of non-normality in the standardized residuals was found for any of the 24 regression runs.

Homoscedasticity. The extent to which a DV exhibits equal levels of variance across the entire range of variation of the IVs is referred to as homoscedasticity. To check for homoscedasticity, a scatterplot of the predicted values of the DV (as the x-axis) against the standardized residuals (as the y-axis) was inspected for the presence of an uneven spread in the vertical scatter from left to right (Mertler and Vannatta 2005). No visual evidence was found for an uneven spread in any of the 24 regression runs.

Linearity. Linearity is the extent to which the relationship between the DV and the IVs follows a straight-line shape. To check for linearity, a scatterplot of the predicted values of the DV (as the x-axis) against the standardized residuals (as the y-axis) was inspected for the presence of a non-linear pattern which deviated from a straight left to right pattern (Mertler and Vannatta 2005). No visual evidence was found for a significant deviation from linearity in any of the 24 regression runs.

Multicollinearity. For each regression run, multicollinearity among the control variables and the IV was tested with a Tolerance statistic, which is a measure of the collinearity among the tested variables. A Tolerance value of .10 or less is considered to be a serious problem (Mertler and Vannatta 2005). No Tolerance values were found below the 0.10 threshold, and therefore the multicollinearity test was satisfied for all 24 regression runs.

5.4. Testing Results

In each of the following sub-sections, the results of each hypothesis test are contained in a table which shows both the linear test results and the quadratic test results. For the linear regressions and the quadratic regressions, the tables include the unstandardized coefficient, the standard error, the standardized beta, the adjusted R-squared and the change in R-squared from the first step to the second step for the linear regressions and from the second step to the third step for the quadratic regressions. For each regression which produced a significant result for the IV or transformed IV coefficient ($p < .05$), the detailed results of all three models are shown in Appendix D.

In general, the predictive values of the models were relatively consistent across the 24 regressions. Including the effect of the control variables, the explanation of variance was highest for the regressions of Software Downloads with adjusted R-squared values ranging from .306 to .393 for the linear regressions and from .302 to .400 for the quadratic regressions. The predictive values for the regressions of Page Views were nearly as high. The least predictive regressions were for Software Releases, where adjusted R-squared values range from .011 to .070 for the linear versions and from .006 to .065 for the quadratic versions. The predictive values for the regressions Code Commits were only slightly higher than these values.

5.4.1. Group Density

The four Group Density hypotheses (H1a through H1d) were tested and a summary of the results are shown on Table 14. For the linear regressions on Group Density, a significant negative relationship was found for both Software Downloads and

Page Views (both at $p < .001$). For both of these regressions, the effect of an increase in Group Density from the average value of .078 to a value of .178 would be to reduce Software Downloads and Page Views by about 40 percent. Details for these two regressions are contained in Tables D-1 and D-2 in Appendix D. Negative relationships were also found for Code Commits and Software Releases, although at less significant p-values of .066 and .063 respectively. For the quadratic testing, a near-significant result was noted for the Software Downloads model and the Page Views model in support of a U-shaped relationship.

5.4.2. Core Density

The four Core Density hypotheses (H2a through H2d) were tested and a summary of the results are shown on Table 15. For the linear regressions on Core Density, a significant negative relationship was found for Software Releases (at $p < .05$). Further details of this regression are contained on Table D-3 in Appendix D. Near-significant negative relationships were also found for Code Commits ($p = .057$) and Software Downloads ($p = .067$). For the quadratic testing, a significant result was noted for the Page Views model ($p < .05$) in support of a U-shaped relationship. Further details of this regression are contained on Table D-4 in Appendix D.

5.4.3. Peripheral Two-Mode Density

The four Peripheral Two-Mode Density hypotheses (H3a through H3d) were tested and a summary of the results are shown on Table 16. For the linear regressions on Peripheral Two-Mode Density, a weak negative relationship was noted for only one of the IVs: Software Downloads (at p = .092). For the quadratic testing, no significant or near-significant relationships were found.

5.4.4. Core Membership Degree

The four Core Membership Degree hypotheses (H4a through H4d) were tested and a summary of the results are shown on Table 17. No significant or near-significant relationships were found for the linear regressions on Core Membership Degree. However, for the quadratic regressions, one very weak result was found for Software Downloads (p = .099) in support of an inverted-U shaped relationship.

5.4.5. Administrator Membership Degree

The four Administrator Membership Degree hypotheses (H5a through H5d) were tested and a summary of the results are shown on Table 18. For the linear regressions on Administrator Membership Degree, no significant or near-significant relationships were found. However, for the quadratic regressions, significant support (at $p < .05$) was noted for an inverted-U shaped relationship with Code Commits. Details of this regression are contained on Table D-5 in Appendix D.

5.4.6. Administrator Class Centrality

The four Administrator Class Centrality hypotheses (H6a through H6d) were tested and a summary of the results are shown on Table 19. For the linear regressions on Administrator Class Centrality, a significant positive relationship was found for Software Releases ($p < .01$). Details of this regression are contained on Table D-6 in Appendix D. For the quadratic regressions, significant support was also found for a U-shaped relationship with Page Views ($p < .05$). Details of this regression are shown on Table D-7 in Appendix D.

Table 14
Summary of Regressions on Group Density,
Controlling for Group Size, Core Size and Conversation Volume
(Log-Transformed Dependent Variables)

	Unstandardized Coefficient	Standard Error	Standardized Beta	Adj. R^2	ΔR^2
Linear regressions:					
H1a: Code Commits	-3.374†	1.822	-.182	.049	.023
H1b: Software Releases	-2.427†	1.295	-.186	.036	.024
H1c: Software Downloads	-5.547***	1.237	-.353	.393	.086
H1d: Page Views	-4.871***	1.285	-.311	.339	.067
Quadratic‡ regressions:					
H1a: Code Commits	16.026	14.560	.175	.050	.008
H1b: Software Releases	9.643	10.359	.149	.035	.006
H1c: Software Downloads	16.375†	9.827	.210	.400	.012
H1d: Page Views	17.097†	10.203	.221	.348	.013

* $p < .05$; ** $p < .01$; *** $p < .001$; n = 143 groups

† p = .066 (Code Commits Linear), .063 (Software Releases Linear)

† p = .098 (Software Downloads Quadratic), .096 (Page Views Quadratic)

‡ First regressed on independent variable and then regressed on mean-centered and squared independent variable

Table 15
Summary of Regressions on Core Density,
Controlling for Group Size, Core Size and Conversation Volume
(Log-Transformed Dependent Variables)

	Unstandardized Coefficient	Standard Error	Standardized Beta	Adj. R^2	ΔR^2
Linear regressions:					
H2a: Code Commits	-.707†	.368	-.184	.050	.025
H2b: Software Releases	-.570*	.261	-.210	.044	.032
H2c: Software Downloads	-.489†	.265	-.150	.321	.016
H2d: Page Views	-.267	.272	-.082	.276	.005
Quadratic‡ regressions:					
H2a: Code Commits	1.552	1.201	.185	.055	.011
H2b: Software Releases	.121	.855	.020	.037	.000
H2c: Software Downloads	.596	.866	.084	.318	.002
H2d: Page Views	1.910*	.877	.269	.295	.024

* $p < .05$; ** $p < .01$; *** $p < .001$; n = 143 groups

† p = .057 (Code Commits Linear), .067 (Software Downloads Linear)

‡ First regressed on independent variable and then regressed on mean-centered and squared independent variable

Table 16
Summary of Regressions on Peripheral Two-Mode Density,
Controlling for Group Size, Core Size and Conversation Volume
(Log-Transformed Dependent Variables)

	Unstandardized Coefficient	Standard Error	Standardized Beta	Adj. R^2	ΔR^2
Linear regressions:					
H3a: Code Commits	-.249	.996	-.028	.025	.000
H3b: Software Releases	-.399	.707	-.064	.013	.002
H3c: Software Downloads	-1.200†	.708	-.159	.318	.014
H3d: Page Views	-.350	.728	-.047	.272	.001
Quadratic‡ regressions:					
H3a: Code Commits	.108	4.714	.002	.018	.000
H3b: Software Releases	-2.829	3.339	-.091	.011	.005
H3c: Software Downloads	1.941	3.345	.052	.315	.002
H3d: Page Views	4.522	3.425	.121	.276	.009

* $p < .05$; ** $p < .01$; *** $p < .001$; n = 143 groups

† $p = .092$

‡ First regressed on independent variable and then regressed on mean-centered and squared independent variable

Table 17
Summary of Regressions on Core Membership Degree,
Controlling for Group Size, Core Size and Conversation Volume
(Log-Transformed Dependent Variables)

	Unstandardized Coefficient	Standard Error	Standardized Beta	Adj. R^2	ΔR^2
Linear regressions:					
H4a: Code Commits	.099	.116	.071	.030	.005
H4b: Software Releases	-.015	.083	-.015	.011	.000
H4c: Software Downloads	.113	.083	.096	.313	.009
H4d: Page Views	.052	.085	.045	.272	.002
Quadratic‡ regressions:					
H4a: Code Commits	-.068	.061	-.132	.032	.008
H4b: Software Releases	.023	.044	.064	.006	.002
H4c: Software Downloads	-.073†	.044	-.167	.322	.013
H4d: Page Views	-.022	.045	-.051	.268	.001

* $p < .05$; ** $p < .01$; *** $p < .001$; n = 143 groups

† $p = .099$

‡ First regressed on independent variable and then regressed on mean-centered and squared independent variable

Table 18
Summary of Regressions on Administrator Membership Degree, Controlling for Group Size, Core Size and Conversation Volume (Log-Transformed Dependent Variables)

	Unstandardized Coefficient	Standard Error	Standardized Beta	Adj. R^2	ΔR^2
Linear regressions:					
H5a: Code Commits	.022	.063	.029	.026	.001
H5b: Software Releases	-.021	.045	-.040	.013	.002
H5c: Software Downloads	.041	.045	.065	.308	.004
H5d: Page Views	.017	.046	.027	.271	.001
Quadratic‡ regressions:					
H5a: Code Commits	-.040*	.019	-.303	.049	.029
H5b: Software Releases	.012	.014	.129	.011	.005
H5c: Software Downloads	-.016	.014	-.138	.310	.006
H5d: Page Views	-.007	.014	-.065	.267	.001

* $p < .05$; ** $p < .01$; *** $p < .001$; n = 143 groups

‡ First regressed on independent variable and then regressed on mean-centered and squared independent variable

Table 19
Summary of Regressions on Administrator Class Centrality,
Controlling for Group Size, Core Size and Conversation Volume
(Log-Transformed Dependent Variables)

	Unstandardized Coefficient	Standard Error	Standardized Beta	Adj. R^2	ΔR^2
Linear regressions:					
H6a: Code Commits	.573	.471	.118	.035	.010
H6b: Software Releases	.963**	.326	.280	.070	.057
H6c: Software Downloads	-.211	.339	-.051	.306	.002
H6d: Page Views	-.247	.346	-.060	.273	.003
Quadratic‡ regressions:					
H6a: Code Commits	1.709	1.474	.105	.038	.009
H6b: Software Releases	-.515	1.026	-.045	.065	.002
H6c: Software Downloads	.488	1.066	.035	.302	.001
H6d: Page Views	2.347*	1.069	.170	.293	.024

* $p < .05$; ** $p < .01$; *** $p < .001$; n = 143 groups

‡ First regressed on independent variable and then regressed on mean-centered and squared independent variable

6. DISCUSSION

This chapter begins with a summary and discussion of the results in relation to the hypotheses and in comparison with the limited empirical findings that have been reported in the open source software literature. This is followed by a set of conjectures which suggest plausible explanations for the alternative relationships that were implied by the hypothesis testing results. In order to further interpret the meaning of the results, these conjectures are then assessed with respect to their implications regarding the likely direction of causality between social network structure and community success. Finally, the unexpected lack of effect of structure on success is discussed and possible explanations are offered.

6.1. Summary of Findings

This section presents a summary and discussion of the results of hypothesis testing which were presented in Chapter 5. Each of the following sub-sections contains a review of the results for the closure, bridging, and leader centrality hypotheses along with an associated results summary table.

6.1.1. Closure

The results for the 12 regressions associated with closure are presented in Table 20. The table summarizes the results of regressions on Group Density, Core Density, and Peripheral Two-Mode Density (as previously referenced on Tables 14, 15 and 16) and shows each hypothesized relation in comparison with an alternative relation suggested by the regression result, if applicable. All of the closure hypotheses posited an inverted-U relationship, reflecting the expectation of a positive slope for lower levels of closure,

Table 20
Summary of Test Results for Closure Hypotheses

Hyp#	*Independent Variable*	*Dependent Variable*	*Success Dimension*	*Hypothesized Relation*	*Suggested Alternative Relation*	*Detail Results Table*
H1a	Group Density	Code Commits	Output	Inverted-U	Negative (p=.066)	
H1b	Group Density	Software Releases	Output	Inverted-U	Negative (p=.063)	
H1c	Group Density	Software Downloads	Activity	Inverted-U	Negative ***	Table D-1
H1d	Group Density	Page Views	Activity	Inverted-U	Negative ***	Table D-2
H2a	Core Density	Code Commits	Output	Inverted-U	Negative (p=.057)	
H2b	Core Density	Software Releases	Output	Inverted-U	Negative *	Table D-3
H2c	Core Density	Software Downloads	Activity	Inverted-U	Negative (p=.067)	
H2d	Core Density	Page Views	Activity	Inverted-U	U-Shaped *	Table D-4
H3a	Peripheral TM Density	Code Commits	Output	Inverted-U	None	
H3b	Peripheral TM Density	Software Releases	Output	Inverted-U	None	
H3c	Peripheral TM Density	Software Downloads	Activity	Inverted-U	Negative (p=.092)	
H3d	Peripheral TM Density	Page Views	Activity	Inverted-U	None	

* p < .05 ; ** p < .01 ; *** p < .001; n = 143 groups

a negative slope for higher levels of closure, and a maximal point occurring at a moderate level of closure. In effect, the positive segment of the hypothesized relationship reflects the expected benefits associated with at least some level of density among the conversations, while the negative segment reflects the prediction that additional connections would be counterproductive and that the "cost of ties" would become dominant, as discussed in Chapter 3.

For Group Density, the results did not support an inverted-U shape for any of the hypotheses. Rather, a negative relationship was found. The strongest negative relationship was found between Group Density and the two community activity variables, Software Downloads and Page Views (at p-values < .001). There is also evidence of a negative relationship between Group Density and the community output variables, although the relationship is not as strong (with p-values of .066 and .063). With reference to the results for the H1c and H1d hypotheses, it is noted that these regressions showed both linear relationships and U-shaped relationships. Because the linear relationships had a more significant p-value (< .001) than the U-shaped relationships (.098 and .096), they were considered to be dominant and only the linear results are shown in Table 20.

For Core Density, an inverted-U relationship was also expected but with a less extensive negatively sloped segment, considering the additional positive benefits associated with the needs of the core subgroup to be more interactive in creating the software. For these hypotheses, a mostly negative relationship with community success was observed, with three of four regressions showing a negative result. The negative relationship was stronger and more consistent for the output variables than for the activity variables. The strongest result was between Core Density and Software Releases ($p < .05$). In the case of the activity variables, one of the two relationships (with Page Views) was found to be a U-shape (at $p < .05$). A U-shaped relationship involves a negative slope for lower levels of the independent variable and then a positive slope for higher levels of the independent variable, with a minimum occurring at a moderate level of the independent variable.

For Peripheral Two-Mode Density, an inverted-U relationship was expected but with less emphasis on the negative side because of the additional benefits associated with the positive psychological effects of including the peripheral developers in core discussions. The results of these regressions did not support the hypotheses, but rather contained only one weak negative relationship ($p = .092$) on just one of the four success variables – Software Downloads - with no effect seen on the other three variables.

While it was generally expected that the closure-success relationship would be an inverted-U in which a segment of the curve is negatively sloped, it was surprising to find a negative slope for the entire length of the curve in 8 of the 12 closure hypotheses. In effect, these results suggest that there is essentially no benefit to closure within an open source software project community.

The strongest negative relationships for Group Density were noted for the activity variables, while the strongest negative relationships for Core Density were observed for the output variables. Comparing the Group Density results with the results for Core Density, it is noted that the negative relationships were less pronounced for the core subgroup than for the group as a whole. This may be an indication that the expected benefits associated with the needs of the core subgroup are influencing the result.

However, it is still surprising to consider that density among the core subgroup seems to produce no benefit with respect to community output. It is interesting to note that no significant negative relationship was seen for the Peripheral Two-Mode Density hypotheses which may indicate that the expected benefits of the peripheral-core connectivity are acting to offset the otherwise negative aspects of closure as noted above.

It is difficult to compare these findings with reports in the open source software literature because most of the prior social network studies of open source have been descriptive and have not attempted to relate social network structure to success at the level of the project community. Healy and Schussman (2003) study the statistical characteristics of the entire set of projects on SourceForge but they do not address social network structures at the project level. Krishnamurthy (2002) notes the surprisingly low volume of conversations in open source projects but the author does not calculate conversational density. Volume and density are distinct concepts and a finding of low volume does not necessarily imply a finding of low density, although the two are not inconsistent.

One recent paper by Crowston and Howison (2006) reported the results of an empirical study of bug report forums. Their method of collecting data and defining the conversational network was similar to the method used in this dissertation, except that they focused their data collection efforts on bug report forums rather than general forums. The authors calculated and reported density of the conversation networks and found a negative relationship between conversational density and group size. This result corresponds with the findings of the dissertation that group density and group size are negatively correlated (Pearson correlation value of -.52, see Table 13). However, the

Crowston and Howison (2006) study did not consider a success variable in their regression. They regressed density on group size, while the dissertation study regressed success on density while controlling for group size. Thus, the dissertation study controlled for the relationship between density and group size, and still found a negative relationship between density and success. Crowston and Howison did not perform such an analysis.

6.1.2. Bridging

The results for the 8 hypotheses associated with bridging are presented on Table 21. A positive relationship was expected for these hypotheses, which includes Core Membership Degree and Administrator Membership Degree. As discussed in Chapter 3, there were a number of expected benefits associated with bridging ties such as providing access to new ideas, obtaining help to solve problems, and increasing the likelihood of recruiting new members to the focal project. While some cost-of-ties effect was recognized, it was noted that this cost was not compounded as with intragroup ties and therefore an overall positive relationship was expected.

The results for the bridging regressions did not support a positive relationship for any of the hypotheses. For Core Membership Degree, only one of the four runs showed an inverted-U result – Software Downloads - and that result was very weak (p=.099). The other three runs showed no significant effect. Considering that a positive relationship was expected, it was surprising to find that the extensiveness of bridging ties did not have an effect on success, implying that such bridging ties are not an important

Table 21
Summary of Test Results for Bridging Hypotheses

Hyp#	*Independent Variable*	*Dependent Variable*	*Success Dimension*	*Hypothesized Relation*	*Suggested Alternative Relation*	*Detail Results Table*
H4a	Core Member. Degree	Code Commits	Output	Positive	None	
H4b	Core Member. Degree	Software Releases	Output	Positive	None	
H4c	Core Member. Degree	Software Downloads	Activity	Positive	Inverted-U (p=.099)	
H4d	Core Member. Degree	Page Views	Activity	Positive	None	
H5a	Admin. Member. Degree	Code Commits	Output	Positive	Inverted-U *	Table D-5
H5b	Admin. Member. Degree	Software Releases	Output	Positive	None	
H5c	Admin. Member. Degree	Software Downloads	Activity	Positive	None	
H5d	Admin. Member. Degree	Page Views	Activity	Positive	None	

* p < .05 ;   ** p < .01 ;   *** p < .001;   n = 143 groups

factor in open source software project communities. For Administrator Membership Degree, again only one of the four runs showed an inverted-U result – Code Commits - although in this case, the result was significant at $p < .05$. Again, the lack of an effect of administrator bridging on three of the four success variables was surprising.

In a recent study by Grewal et. al. (2006), the authors collected data from 108 open source software project communities on SourceForge and related various measures of bridging (which they refer to as "network embeddedness") with the number of code commits and the number of downloads (used as measures of project success). Overall, the authors obtained a mixed set of positive, negative, and "no-effect" relationships between bridging and success. Their conclusion that the impact of bridging was greater on code commits than on downloads is consistent with the dissertation results. Their suggestion that bridging has "powerful but subtle effects on project success" is generally inconsistent with the dissertation finding that bridging had only a minor effect on success. However, due to methodological differences, the comparability of the two studies is questionable. For example, Grewal et. al. (2006) used many different bridging measures which were not comparable to the measures used in the dissertation. In addition, their study utilized a nominalist sampling approach in which 10 projects were selected based on their common platform technology and then other projects were selected based on known bridging ties with these original 10 projects. This is in contrast with the dissertation study in which a random sampling strategy was used. It is possible that the bridging results for a sample of projects with known bridging connections may be different than the results for a randomly selected sample of projects.

6.1.3. Leader Centrality

The results for the 4 hypotheses associated with leader centrality are presented on Table 22. As discussed in Chapter 3, some positive relation was expected between leader centrality and success in that a certain level of connectedness between the leaders and the rest of the group would seem to be necessary to integrate the code contributions of the members and to coordinate some activities as needed. However, at higher levels of leader centrality, a cost-of-ties effect was expected in which too much centrality becomes burdensome on the administrators, resulting in a negative curve at higher levels of centrality. Therefore, the hypotheses linking Administrator Class Centrality with community success posited an inverted-U relationship.

The results presented in Table 22 did not support an inverted-U shaped relationship for any of the four leader centrality hypotheses. However, the suggestion of an alternative relationship shape was inconclusive. In the case of Software Releases, an alternative positive relationship is suggested ($p < .01$). Yet, in the case of Page Views, an alternative U-shaped relationship is suggested ($p < .05$). For the other two hypotheses, no significant effect was noted.

With regard to open source software literature, no studies were identified in which leader centrality measures are investigated. However, the literature does suggest that open source administrators tend to operate in low key roles, avoiding power relationships and delegating as much as possible. These observations are not inconsistent with the finding that leader centrality had a mixed relationship with success.

Table 22
Summary of Test Results for Leader Centrality Hypotheses

Hyp#	*Independent Variable*	*Dependent Variable*	*Success Dimension*	*Hypothesized Relation*	*Suggested Alternative Relation*	*Detail Results Table*
H6a	Admin. Class Centrality	Code Commits	Output	Inverted-U	None	
H6b	Admin. Class Centrality	Software Releases	Output	Inverted-U	Positive **	Table D-6
H6c	Admin. Class Centrality	Software Downloads	Activity	Inverted-U	None	
H6d	Admin. Class Centrality	Page Views	Activity	Inverted-U	U-Shaped *	Table D-7

* p < .05 ; ** p < .01 ; *** p < .001; n = 143 groups

In summary, of the 24 hypotheses that were tested, a total of 7 produced results which were significant at $p < .05$ (see Tables D-1 through D-7), 6 produced results which were significant at $p < .10$, and the remaining 11 hypothesis tests showed no significant effects. While none of the hypothesized relationships were supported, the alternative relationships that were suggested are summarized below:

1. In general, a negative relationship was observed between the closure variables and the success variables (mainly considering the activity variables regressed on Group Density, and the output variables regressed on Core Density).

2. U-shaped relationships were observed for Page Views (considering the regressions on Core Density and Administrator Class Centrality).

3. An inverted-U relationship was observed between Administrator Membership Degree and Code Commits.

4. A positive relationship was observed between Administrator Class Centrality and Software Releases.

6.2. Conjectures and Causality

As discussed in the previous section, the results broadly deviated from expectations. Considering that this was one of the first large-scale empirical studies of the relationship between social network structure and success in open source software project communities, it seemed likely that some surprising results would be found. However, the extent of the deviation that was observed was dramatic considering that the hypotheses were formulated based on well-established social network theories of team effectiveness with plausible adjustments made to reflect expected differences between teams and open source software project communities. In addition, even though the expected relationships were not found, a number of other relational shapes were implied.

In this section, conjectures are offered which attempt to explain each of the four significant findings noted at the end of the previous section. These conjectures consist of explanatory arguments which are plausible but which are not empirically tested in the current study. Considering the extent of deviation from expectations, it is also appropriate to reassess the causality assumptions which were inherent in the study's conceptual research model (Figure 4). Therefore, each conjecture is further reviewed with respect to its implications for the most likely direction of the causal arrow between social network structure and community success. In the remainder of this section, each finding is stated, followed by one or more conjectures which are related to that finding.

Finding #1: in general, a negative relationship was observed between the closure variables and the success variables. The closure of a network is essentially the proportion of the total possible links in a network that are actually connected. Therefore, a higher closure value indicates more connected links while a lower closure value indicates fewer connected links. If the causal arrow is assumed to point from structure to outcome, then the observed negative relationship between closure and success would imply that a lack of network links can somehow cause or logically lead to success. No plausible conjectures were identified which could explain such a relationship. Therefore, the possibility of a spurious relationship was considered whereby a third factor is identified which affects both closure and success.

Three conjectures were formulated which, if valid, imply that the negative relationship between closure and success is spurious. All of these conjectures involve a third factor which is associated with the attributes of certain project artifacts. One of these factors is the modularity of the software architecture, which is a technological artifact. The other two factors include the quality of the software documentation and the appropriateness of the project rules, both of which are informational artifacts of the project. These three factors and their suggested impact on closure and success are discussed below.

Software architecture. The modularity of the software architecture is recognized as an important success factor for open source software projects (MacCormack et. al. 2006). Modular software architecture permits changes to source code within one module without significant effects on code contained in other modules. An ineffective modular design will tend to increase coding interdependencies in which the coding work of one developer is more likely to affect the work of other developers.

As a result, ineffective modularity will tend to increase the closure level as multi-person conversations are needed to discuss the impact of code changes and to investigate complex bugs which are more likely to arise. At the same time, this may lead to a reduction in developer productivity as efforts are shifted from coding to conversation, and may also demotivate the developers who are focused on writing code and view conversation as a distraction. The need for dense discussions may frustrate these developers which may cause them to reduce their effort level and in some cases they may even choose to abandon the project. The combined impact of reduced productivity and reduced effort is to decrease the output dimension of success.

With regard to the activity dimension of success, ineffective modularity can directly reduce the quality of the software that is produced, because of the increased likelihood of complex bugs and their negative impact on software usability. In addition, the reduction in productivity and coding effort that was mentioned above will have an indirect negative effect on software quality. A lower level of software quality will tend to reduce the interest level of the community which will translate into a decrease in the number of downloads and the number of page views, both of which are measures of the activity dimension of success.

In summary, ineffective software modularity will tend to increase closure as a result of the increase in coding interdependencies, and at the same time, it will tend to decrease output due to losses in productivity and effort, and will decrease activity due to negative impacts on software quality. The suggested positive relationship between modularity ineffectiveness and closure and the suggested negative relationship between modularity ineffectiveness and success will result in a negative correlation between closure and success. However, because this negative correlation arises from the effects of a third variable (software modularity ineffectiveness), the closure-success relationship would be viewed as spurious and no causal relationship would be suggested between closure and success.

Software documentation. In a software development project, the software documentation contains a description of the overall architecture and modular structure of the software, specific descriptions of the functionality of various procedures, data definitions, and other important information about the software. High quality documentation is clear and complete and it makes the overall software architecture explicit. Poor or incomplete documentation can increase the level of closure as questions and discussions are necessary in order to clarify features of the software that are useful and/or necessary to know as a developer writes source code. As with ineffective software modularity, a low quality of software documentation will decrease output success as effort is shifted from coding to conversation and as frustrated developers reduce their overall level of effort.

Poor quality software documentation can directly reduce the quality of the software that is produced, because of the increased likelihood that coding efforts will be based on incorrect assumptions and missing information. In addition, the reduction in productivity and coding effort will have an indirect negative effect on software quality, which as was the case with ineffective software modularity, will translate into a decrease in the activity levels of the project community.

Thus, poor quality software documentation will tend to increase closure as questions and discussions are necessary to clarify knowledge needed for coding tasks. At the same time, it will decrease output and activity as described above. As with the software modularity conjecture, this suggests that the negative relationship between closure and success is spurious and arises as a result of the positive relationship between poor software documentation and closure and the negative relationship between poor software documentation and success.

Project rules. Open source software projects are less reliant on hierarchy and supervision than software development teams, and therefore the project rules play an important role in guiding the behavior of the independent contributors. These rules may be formally stated in a document or they may be informally stated in various public forum postings. The open source license that is chosen is also part of the project rules. In effect, these rules provide guidelines regarding the rights and responsibilities of the community members, and they specify certain types of behaviors that are either encouraged or discouraged. Rules which are inappropriate or understated will tend to lead to complaints, disputes and controversies that require multi-person discussions, thus resulting in an increase in closure. As with the software architecture and software documentation artifacts, this increase in closure will tend to reduce the output levels, and the resulting indirect negative impact on software quality will tend to reduce the activity levels. Therefore, this conjecture also implies that the closure-success relationship is spurious, based on arguments that are similar to the two previous conjectures.

Finding #2: U-shaped relationships were observed for Page Views. As previously noted, U-shaped relationships were observed between Core Density and Page Views and between Administrator Class Centrality and Page Views. This suggests that a negative relationship exists for lower levels of the independent variable and that a positive relationship exists for higher levels of the independent variable. No conjecture which assumes a homogeneous study population could be identified to explain this result. However, if it is assumed that a subset of the study population has different characteristics that would lead to a positive relationship with Page Views, then the combination of this situation with a negative relationship for the remainder of the population (as was seen in other regression tests) would result in a U-shaped relationship.

In particular, it is possible that certain project communities consist of individuals who know each other in an off-line context and who choose to utilize the resources of

SourceForge to collaboratively develop software. These groups may utilize planning and control approaches that are associated with teams and that are not commonly used in open source software project communities. In effect, these may be de facto software development teams that use the SourceForge facilities to conduct their work. If this were true, then these de facto teams would likely exhibit positive relationships between closure and success and leader centrality and success, similar to the relationships that have been observed for other kinds of teams.

If this conjecture is true, then the study population actually consisted of two different regimes which would tend to dilute the results and reduce the significance of all of the regression results. However, it is noted that only 2 of the 24 regressions resulted in a significant U-shape and that various other regressions did show significant linear and inverted-U results. In addition, a significant regime split can often be detected by an obvious bimodal or multimodal distribution of the research variables, and no such distribution pattern was noted. Therefore, it is suggested that the impact of the U-shaped finding is secondary and that there are no important implications regarding the direction of causality.

Finding #3: an inverted-U relationship was observed between Administrator Membership Degree and Code Commits. This relationship involves a positive slope for the lower values of bridging and a negative slope for the higher values of bridging. The most plausible conjecture for this result is that the expected positive effects of bridging are in fact being observed for the lower values of the bridging variable. However, at the higher values of the variable, it is possible that a "cost-of-ties" effect is being seen, in which too many bridging ties become burdensome on the administrators and the effect on community success is negative. This cost-of-ties effect was expected for closure and leader centrality but was not expected for bridging because the tie only affected one member of the community (the administrator) and the level of expected benefits was extensive. However, because of the importance of the administrator, the cost-of-ties effect may in fact be important. If this conjecture is true, then the implication is that the causal arrow does point from social network structure (bridging) to output (Code Commits) in reference to this finding.

Finding #4: a positive relationship was observed between Administrator Class Centrality and Software Releases. The decision to make a software release is typically made by the administrator. While a high level of coding activity (Code Commits) is logically associated with frequent releases (Software Releases), it is possible for an administrator to make frequent releases even if there is a relatively low volume of code commits. In effect, the decision to release is somewhat arbitrary and it is possible that certain administrators are biased towards frequent releases and therefore they have a higher "propensity to release" than others. If this were the case, then those administrators with high propensity to release would make frequent releases resulting in a high level of Software Releases. In this situation, the frequent releases would tend to generate questions and comments from developers who download the releases and these conversations would tend to dominate the forums and would be directed to the releasing administrator, resulting in high levels of Administrator Class Centrality. In effect, these administrators would be generating their own centrality. If this conjecture were true, then the implication would be that the causal arrow points in a reverse direction from the assumption of the research model – that is, it would point from outcome (Software Releases) to social network structure (Administrator Class Centrality).

Summary. Of all the conjectures offered in this section, only the one for finding #3 implies that the causal arrow points from social network structure to success. Otherwise, all of the other conjectures imply spurious results, reverse causality, or the presence of a qualitatively different subset of communities. Taken together with the various other "no effect" results that were observed, the general implication is as follows:

> The social network structure of an open source software project community has no important effect on community success.

In addition, the three conjectures associated with the negative relationship between closure and success (finding #1) imply that:

> The closure of an open source software project community is a condition or indicator of community success, but is not a driver or cause of such success.

6.3. The Insignificance of Structure

In the previous section, it was concluded that the social network structure of the open source software project communities that were studied had no important effect on community success. In this section, this insignificance of structure with respect to success is further discussed. In particular, explanations are offered regarding how it could be that social network structure has no important effect on community success, even though social network theory, supported by numerous empirical studies, suggests that structure should be important with respect to group performance.

As previously discussed, social network theory is based on the notion that a social network acts as a conduit for the flow of resources such as knowledge and the tangible resources that can be accessed based on that knowledge. Social capital theory suggests that a structure with high closure within a group will improve the performance of tasks which require the utilization of the knowledge of the group, while a structure with many bridging ties between group actors and non-group actors will improve the performance of tasks which require access to knowledge which is beyond the boundaries of the group. In effect, social network structure is predicted to be important to success in work groups because it can enable or impede the transfers of knowledge, where such transfers are needed to support activities such as learning, problem-solving, coordination and task completion, all of which are necessary for successful group outcomes.

Considering this knowledge transfer view as a frame of reference, there are two general reasons that can be offered to explain the insignificance of social network structure with respect to community success. One possibility is that knowledge transfers are somehow being mediated without the involvement of the social network. In effect, other mechanisms may substitute for the social network as a mode of knowledge transfer. The other possibility is that there may simply be less need for knowledge transfers in successfully completing the work associated with open source software projects. Ultimately, both of these reasons may contribute to the explanation of the counterintuitive findings that were previously described. In the remainder of this section, various conjectures are offered which expand upon these two possibilities.

6.3.1. Substitutes for the Social Network

While it may be possible to imagine knowledge transfers that are mediated through shared cognition and/or strong culture, the most tangible possibility seems to be that knowledge could be transferred indirectly through artifacts rather than directly through the social network. Open source software developers operate in a network-mediated computing environment involving many types of tools and other technical artifacts such as source code repositories, programming languages, project web pages, and others (Scacchi 2002). The scenario in which artifacts can successfully mediate knowledge transfer is feasible to the extent that the artifacts can be inscribed with knowledge and that the task can be structured to allow for workflows from person to artifact to person, rather than from person to person. In this case, the artifacts become the mediators of knowledge transfer and they act as a substitute for the social network in this regard. This is somewhat similar to the "knowledge ecology" view offered by Lanzara and Morner (2003).

For example, the source code is an artifact of the project. The statement sequence, algorithmic logic, and general organization of the code can be viewed as a kind of inscription of knowledge. When a developer checks out a batch of code from the source code repository, the knowledge that was inscribed by all of the previous contributors to that code becomes available to that developer. In a sense, these prior developers are "speaking" to the new developer through the code. As this developer makes changes to the code, he or she is inscribing their own knowledge into the code, and this new knowledge becomes available to other developers as soon as the new code is committed into the repository.

An example of artifact mediation as a substitute for social network structure may be found in the use of outside project records by teams versus open source software project communities. In the case of teams, the detail and accessibility of these outside records is relatively limited compared with the transparency and accessibility of open source software project records. Team members commonly use their bridging ties in order to obtain this outside information and therefore the bridging structure of their social network is important for successful outcomes. In the case of open source software developers, however, it is possible to obtain a great deal of information about outside projects from the publicly accessible work records in the form of source code repositories, public forums, and other informational artifacts which are posted on the project web site, all of which can be located with the use of an efficient search engine. These records can be used by developers to learn about other projects and to obtain useful artifacts such as source code fragments and even problem solutions which are noted in public forums. Therefore, the importance of the bridging ties is reduced and the public record artifacts act to substitute for the social network structure with regard to mediating these knowledge flows. The use of open source software project records in this manner was noted by von Krogh et. al. (2005) who found that developers often reported reading the mailing lists of other projects:

> The barriers between open source projects seem to be less distinct as one might assume. Since developers stated that they tend to read several projects' mailing lists, it is difficult if not impossible to track 'silent' and uncredited knowledge transfer in the form of ideas between projects as there is no formal system for recording these kind of transfers.

In the case of social network mediated knowledge transfers, an ineffective social network structure can act to impede the knowledge flow (for example, as where low closure limits the interpersonal flow of knowledge). In a similar way, an ineffective design for a knowledge-mediating artifact may act to impede the flows of knowledge. For example, if the software documentation artifact is of high quality, then it can be relied upon to facilitate knowledge transfers. If however it is of low quality, then it can impede such transfers and require that the social network be used in its place. If the overall task structure is designed for artifact mediated transfers, as may be the case in open source software projects which must operate in a geographically dispersed and asynchronous environment (Yamauchi et. al. 2000), then this can represent an inefficiency which is reflected in a lower level of success.

6.3.2. Reduced Need for Knowledge Transfer

Various possible explanations can be offered regarding why there may be less need for knowledge transfer in open source software project communities, when compared with the needs of traditional teams. These explanations are listed and described below.

Modular software architecture. Modular software architecture permits changes to source code within one module without significant effects on code contained in other modules. This reduces the need for knowledge transfer between developers who are working on different modules.

Accepted standards and tools. The use of well-known coding standards, design approaches, and programming languages may act to reduce the need for knowledge transfer because developers will already be familiar with these tools and will not require additional knowledge in order to use them.

Highly skilled developers. Project community members may be so highly skilled and experienced that knowledge transfer is not very important for learning and problem solving. These experienced individuals may not need direction from a central leader but rather are self-directed such that their choice of task and work method productively contributes to the overall software development task. They may also not need or want help from other members of the project community or from individuals outside of the project community.

Familiarity. It has been observed that familiarity among the members of teams can act to weaken the relationship between social network structure and team performance, implying a reduced need for knowledge transfers (Balkundi and Harrison 2006). This may also be observed in open source software projects. However, the study population involved the two-year period following the first release of software, and therefore the familiarity effect may not be so important in this study as compared with the familiarity that develops in teams over the span of many years. In open source projects, it is also possible that the core developers become familiar with the source code itself to the extent that they have contributed to its growth from an early seed stage. This kind of familiarity may also reduce the need for knowledge transfer.

Developer as user. In developer-targeted software projects, the developer is also the user and therefore the communication that would normally occur between user and developer is not necessary. This would result in a reduction in the need for knowledge transfer, based on a comparison with a traditional team-based approach in which external users are usually consulted in developing software requirements and in evaluating the project output.

Open source culture. The culture of the broader open source software community is characterized as a kind of meritocracy in which a rational approach is favored over other approaches which resort to hierarchical position or relationships of power and influence (Raymond 1999). Such a culture may result in limited exchanges of knowledge compared with hierarchical cultures which require more protracted and extensive knowledge transfers as may be seen within a bureaucratic structure (Yamauchi 2000).

Shared mental models. To the extent that participants have shared mental models, it is possible that these shared models may reduce the need for knowledge flows associated with coordination and other development activities (Scozzi et. al. 2008). In some respects, this may be related to the notion of familiarity as described above. In addition, shared mental models can also be viewed as an aspect of the open source culture.

7. CONCLUSIONS

The objective of this dissertation research was to investigate the social network structural conditions that are associated with success in open source software project communities. In pursuing this goal, a set of propositions were developed based on social network theories of teams and other relevant theoretical and empirical literature. These propositions were operationalized in the form of 24 hypotheses which were then tested using data obtained from open source software project archives. The results deviated broadly from the expectations and an alternative set of relationships was observed.

Plausible explanations for the alternative relationships were suggested and analyzed and the two primary implications were that 1) the social network structure of an open source software project community has no important effect on community success, and 2) the closure of an open source software project community is a condition or indicator of community success, but is not a driver or cause of such success. This "insignificance of structure" was examined and a series of explanations were offered which suggested that artifacts may be substituting for the social network as a knowledge transfer medium, and that the overall need for knowledge transfer within an open source software project may be lower than in a traditional team-based project.

In this final chapter, the implications of these surprising results are further explored. This begins with the suggestion that the observed anomalies may represent a paradigm disruption which triggers the need for theory building. Some requirements for such a theory building effort are offered along with two propositions which are suggested as extensions of explanations offered in Section 6.3. This is followed by a discussion of the implications for research and practice, the contributions of the work, a discussion of research limitations, and a presentation of future research directions.

7.1. Implications

The arguments presented in Chapter 6 suggest that the findings of this work represent an anomaly with respect to currently accepted theories of team effectiveness and social capital. More broadly, this work suggests that what is referred to as an "open source software project community" is actually neither "team" nor "community" but is a new kind of social entity which is built upon a socio-technical development process involving extensive interactions between humans and technical artifacts. In this section, these suggestions are further explored regarding the possibility that open source software may represent a disruption to the team development paradigm. This is followed by a discussion of requirements for building this new theory. Finally, the implications of these conclusions with respect to research and practice are considered.

7.1.1. Paradigm Disruption

A paradigm is characterized by well-accepted theories and ways of thinking (Kuhn 1996). The disruption to an existing paradigm is often identified by observations which are counterintuitive and by the failure of existing theories and paradigmatic thinking to account for these observations (Kuhn 1996). In addition, Kuhn notes that technology changes will often lead to paradigm disruptions: "... technology has often played a vital role in the emergence of new sciences." (Kuhn 1996)

It is argued that the concept of teams and the social network theory of team effectiveness are aspects of a team development paradigm. In particular, the notion that teams are the fundamental means for developing knowledge products is certainly well accepted in research and practice. In addition, the assertions of social capital theory regarding the importance of closure and bridging structures for work group outcomes are well-tested and broadly applied throughout the social network theoretical literature.

In the case of open source software project communities, it is noted that open source is a relatively new phenomenon which has emerged along a track which is generally parallel to the developmental track of the internet. Further, open source projects are highly dependent on the internet and advanced information technology tools which have only recently become available. Therefore, it is certainly possible that a technology as pervasive and disruptive as the internet could be leading to the emergence of a new form of collaborative development which might represent a disruption to the team paradigm.

The findings of this research that the social network structures of an open source software project community have no important effect on its success are certainly counterintuitive. How could social networks not be important for developing software in these communities when they are so important in teams? In particular, it is difficult to fathom how a knowledge-based product as complex as computer software could be developed without the need for dense interactions to facilitate knowledge flows between and among the participating developers.

In Chapter 6, the results of this work were analyzed in depth with reference to the current social network theories and it was apparent that these theories offer little or no predictive value regarding the success of open source software project communities. Taken together with the presence of counterintuitive findings and the possibility that the internet has spawned a new kind of collaborative development process, these arguments suggest that:

> The open source software project community may represent a disruption to the team development paradigm.

A paradigm disruption triggers the need for theory building. If open source is in fact a paradigm disruption, then the need for new theories is apparent. However, even if open source does not qualify as a "full blown" paradigm disruption as defined by Kuhn (1996), the results of this study, if confirmed by future studies, would certainly suggest that a significant anomaly has been found and a confirmed anomaly is a reason for theory building (Weick 1989).

7.1.2. Requirements for a New Theory

Kuhn (1996) describes the typical theory building process that is associated with a paradigm disruption:

> Discovery commences with the awareness of anomaly, i.e., with the recognition that nature has somehow violated the paradigm-induced expectations that govern normal science. It then continues with a more or less extended exploration of the area of anomaly. And it closes only when the paradigm theory has been adjusted so that the anomalous has become the expected. (Kuhn 1996)

The scope of a new theory which addresses the disruption of the team paradigm could possibly encompass all forms of collaborative development involving the structures and behaviors of teams, virtual development communities such as open source software project communities, and similar forms of organization and activity. However, in the short-run, an important starting point would be to build and test theories which are focused on explaining the anomalies of open source software development.

The overall problem to be addressed by the new theory is explaining how open source software project communities can successfully develop complex artifacts such as software without being impacted by the social network structures of closure, bridging or leader centrality. In particular, the theory should explain why social network structure is not important for learning, problem-solving, coordination and task completion in open source software project communities, even though it is important for the successful performance of these activities in teams.

Based on the discussions and possible explanations that were offered in Section 6.3, the following two propositions are suggested as a foundation for future theory building:

> *Proposition A*
> Compared with software development teams and teams in general, open source software project communities substitute artifact mediation for social networks as a mechanism for knowledge transfer.
>
> *Proposition B*
> Compared with software development teams and teams in general, open source software project communities have less need for knowledge transfer in achieving successful outcomes.

The conjectures and explanations offered in Chapter 6 may provide a starting point for further elaborating these propositions and developing testable hypotheses. For example, in expanding on Proposition A, it may be useful to consider the source code repository, software documentation and project rules as artifacts which may be substituting for social networks. In this case, the theory would need to specify how these types of artifacts are mediating knowledge flows and also how the overall task structure and workflow patterns could be organized to permit such flows to lead towards successful task completion. Such a theory might incorporate the notions of self-organization and evolutionary mechanisms. In expanding on Proposition B, the various explanations offered in Section 6.3.2 may provide the basis for defining various hypotheses. Again, the theory would need to specify the manner in which successful task completion can occur without the related knowledge flows taking place.

7.1.3. Research Implications

In many respects, the new theory building process has already begun as evidenced by the significant level of research interest in developing new frameworks and mechanisms for describing and explaining the unique aspects of open source software projects. In a recent article by von Krogh and von Hippel (2006), the authors organize their review of the current status of open source software research into three categories: 1) motivations of open source software contributors, 2) governance, organization, and the process of innovation in open source software projects, and 3) competitive dynamics enforced by open source software. The propositions suggested in Section 7.1.2 involve aspects which are part of von Krogh and von Hippel's second category of research. With regard to other open source software research efforts, the works of Lanzara and Morner (2003) and Lee and Cole (2003) may be especially relevant to the suggested new theory in that these authors discuss the importance of evolutionary mechanisms in the open source development process, and these mechanisms may help to explain how artifact-mediation can substitute for social network structure and still provide adequate support for successful group outcomes.

With respect to organizational theories, even though it is suggested that open source software project communities are not teams, they are still collective forms of work production and therefore organizational theories should be relevant. In particular, some of the earlier organizational research works in the areas of substitutes for leadership (Howell, et. al. 1986), self-regulating teams and socio-technical systems (Cummings 1978), and centralization versus decentralization (Carley 1995) may be productive areas for further investigation. As an example, Kerr and others (Kerr and Jermier 1978) have proposed a substitutes for leadership theory which suggests that highly structured tasks may require lower levels of leadership. In effect, the greater the task structure, the less the requirement for direction. This implies a certain reduction in the required knowledge transfers between the leader and the other team members. As a result, this theory may help to explain the reduced need for knowledge transfer in open source software project communities based upon the structure of the open source tasks. This may be especially applicable for explaining the lack of effect of leader centrality on community success.

In a broader sense, the possible presence of a paradigm disruption should alert researchers in the fields of open source software, team effectiveness and social capital theory to reconsider and more explicitly state their assumptions. In general, the presence of a paradigm can cause a kind of "blindness" to other possibilities and the resistance to paradigm changes is well-established (Kuhn 1996). As a result, researchers in these domains should recognize the possibility that their paradigmatic perspective may be limiting their choice of research phenomena to be studied. In particular, it is possible that existing open source software researchers have been unduly influenced by the team paradigm and it may be appropriate to step back and consider the possibility that open source software communities may be a fundamentally new form of collaborative development. This might involve taking a more grounded approach which explicitly identifies and isolates the team-oriented concepts. In the domain of social network theory, researchers should reconsider their basic assumptions about the social network as a conduit for knowledge flow and consider alternative perspectives in which artifacts may play a key role in knowledge transfer. This may be especially relevant in the study of socio-technical systems.

7.1.4. Practical Implications

One practical implication of the study relates to the finding that administrator bridging has an inverted-U relationship with code commits. This implies that a project community can benefit from the membership ties of the administrator and therefore connections with other projects should be pursued. However, too many ties can be counterproductive and administrators should be aware of how their other memberships and commitments may be having a negative impact on the success of their projects.

In terms of artifact design, the study results imply that certain project artifacts including software architecture, software documentation, and project rules may be important factors of success. Administrators and host platform designers should be aware of the importance of these artifacts and should take actions to ensure that they are properly designed. If problems arise, these artifacts should be carefully evaluated to see if there are any deficiencies that can be corrected.

In more general terms, perhaps the most important implication for practice is the recognition that open source may represent a fundamentally new form of collaborative development. Practitioners should expand their perspectives and reconsider their assumptions that a team is the only organizational form which can be used for collaboratively developing a knowledge product. Open source methods have been shown to be a useful and interesting alternative to team-based software development methods. However, practitioners should be aware that other possible applications of open source methods may be feasible in areas such as the development of innovative product designs, knowledge repositories, and other kinds of knowledge-based products.

7.2. Contributions

Overall, this was one of the first large-scale empirical studies of the relationship between social network structure and success in open source software project communities. In particular, it is the first known study to relate closure and leader centrality to success, and the second known study (after Grewal, et. al. 2006) to relate bridging to success in open source project communities. In the remainder of this section, the specific contributions to theory, methodology, and practice are described.

7.2.1. Theory

This work contributes a social network perspective to the emerging theories of open source software with respect to governance, organization, and development processes. In particular, the anomalous results point towards the consideration of artifact-mediation and knowledge transfer reductions as possible elements which may ultimately be synthesized with these new open source theories. Further, the work has connected open source software research with team effectiveness research in terms of social capital theory and leader centrality.

For team effectiveness researchers and social network theorists, this work provides an interesting counterpoint to well-tested concepts and theories. The results suggest the presence of a paradigm disruption which may require the re-evaluation of assumptions and new theory building efforts with regard to theories of workgroups and the roles and effects of social network structures. In the domain of social network research, the dissertation has extended the application of social network theory to a new form of socio-technical activity and has applied the concept of core and peripheral subgroups within the context of social capital theory.

Ultimately, though, the most significant theoretical contribution of this research may not be in adding to any existing theory but rather in tracing the outlines for a new theory - one which suggests that artifacts may substitute for social networks as mediators of knowledge transfer. As noted by Weick (1989):

> ... the contribution of social science does not lie in validated knowledge, but rather in the suggestion of relationships and connections that had previously not been suspected, relationships that change actions and perspectives.

7.2.2. Methodology

The use of a two-year observation window following first software release date is a methodological contribution which provides for a more controlled study population with respect to project maturity. The study has also demonstrated the use of archival statistics for defining and measuring social network structural variables, and has made a connection between two important research databases which were not previously used in tandem.

Further contributions to social network analytical methodology include the definition of two-mode density in the context of a priori subgroups. Even though two-mode density is a basic social network concept that is often used in practice, it is not commonly used in research and there appears to be potential for further similar applications. Also, the study applies the relatively new concept of class centrality in a unique way, by using it to measure the centrality of a subgroup (administrators) as an independent variable.

7.2.3. Practice

With regard to practice, the study will be useful to individuals and firms who sponsor, manage, and/or participate in open source software projects. In a pragmatic sense, the results of this work may provide practical measurement tools which can be efficiently applied to pre-existing digital archives such as email, instant messaging and online forums (Hinds and Lee Forthcoming). Even though social network structures were not established as likely causes of success, the closure structure was noted to be an important indicator of success, which makes it a useful evaluation metric. Open source software project administrators can use such measures to assess their own communities and to determine if they have the right kinds of structures or if changes might be necessary.

7.3. Limitations

It is recognized that the study population was limited to early-stage projects which were targeted to developers and not sponsored by corporations. The results may not be generalizable to more mature projects and/or projects which are user-targeted or corporate-sponsored.

With regard to the variable selection, it is noted that the choice of bridging variables was limited by the availability of data, and that more appropriate variables may produce different results. In addition, the conversational networks are built from online public forum records, and it is possible that there were other offline conversations among project members which were not captured in the data. However, the norms of open source software promote a high level of openness and transparency which may limit the extent to which these offline conversations actually take place.

The choice of SourceForge as the sole research setting is a limitation in that it is possible that the projects hosted by SourceForge are not representative of the broader population of projects which may be found on other hosting sites and/or which may have their own hosting platform. Also, the extensive transparency associated with SourceForge may not be representative of other hosting sites. However, SourceForge is, by far, the largest of the available hosting platforms and SourceForge projects include a wide variety of software types, application domains, and open source licenses.

With regard to the choice of research method, it is recognized that the use of historical statistics may result in reliability issues (Babbie 2005). Existing statistical records are usually kept for purposes other than research, and various changes can occur in record-keeping methods, information processing systems, definition of fields, and so forth. These matters are addressed by taking proactive steps to identify changes in recording method and other changes which might affect data reliability. Fortunately, the SourceForge foundation is well aware that they are the source of considerable research efforts and, along with their open policy, they appear to be conscientious about publishing their record-keeping methods and announcing any changes. These announcements are carefully reviewed to determine the impact on data reliability and other steps are taken to check the integrity of the data.

Finally, a cross-sectional study design normally results in ambiguity with respect to the direction of the causal arrow between independent and dependent variables, since time precedence cannot be established. Various conjectures were offered and their implications regarding causal direction were discussed. However, as noted in that section, these conjectures are not tested in this study and would require longitudinal studies to more strongly support an argument of causality.

7.4. Future Research Directions

A number of future research directions can be envisioned. In the short-term, attempts to generalize the results of this work to other types of open source software projects would be worthwhile. This would involve relaxing some of the restrictions imposed by the study population definition and re-testing the hypotheses for projects of different maturity levels, projects involving user-targeted software, and projects which are corporate-sponsored rather than community-based. Projects from host organizations other than SourceForge should also be considered.

Because of the anomalous nature of the results, it is important that alternative research methods be used to either confirm or refute the observed deviation from theories of teams and social capital theories. This might involve more intensive field studies in which a small number of project communities are investigated in order to evaluate some of the conjectures that have been offered but have not been empirically tested. These studies can search for the presence of alternative forms of communication among project developers. Also important is to further investigate the possible existence of two different types of project communities, which may be the basis for the U-shaped relationships that were observed.

With regard to theory building, the propositions suggested in Section 7.1.2 should be further developed and elaborated into testable hypotheses. Various kinds of research methods might be applied depending upon the nature of the hypotheses that are suggested. In the short-term, these efforts would be focused on explaining the anomalous results that were seen in open source software project communities. In the longer term, it is possible that these efforts could be expanded to consider other types of virtual development communities that may utilize open source methods and principles in building a more general theory of collaborative development.

Finally, there appears to be significant potential in considering the role and impact of technical artifacts with regard to the open source development process. Ongoing work in socio-technical design research is associated with this type of study. Initially, this work might involve comparative studies of artifacts and their roles in the development process, for example as in comparing a prominent open source software project with the development of a non-software product such as the Wikipedia. More generally, there is the potential to conduct design research studies which use laboratory and field experimental methods to test the impact of different design strategies on the nature and success of the development community that emerges.

LIST OF REFERENCES

Adamic, L. A. and B. A. Huberman (2000). "The nature of markets in the world wide web." Quarterly Journal of Electronic Commerce 1: 5-12.

Ahuja, M. K. and K. M. Carley (1999). "Network structure in virtual organizations." Organization Science 10(6): 741-757.

Allison, P. D. (1999). Multiple regression: a primer. Thousand Oaks, CA, Pine Forge Press.

Almarzouk, M., L. Zheng, et al. (2005). "Open source: concepts, benefits and challenges." Communications of the AIS 16: 756-784.

Axelsson, B. and G. Easton, Eds. (1992). Industrial networks: a new view of reality. London, Routledge.

Babbie, E. (2005). The basics of social research. Belmont, CA, Thomson Wadsworth.

Balkundi, P. and D. A. Harrison (2006). "Ties, leaders, and time in teams: Strong inference about network structure's effects on team viability and performance." Academy of Management Journal 49(1): 49-68.

Barabasi, A.-L. (2002). Linked: the new science of networks. Cambridge, MA, Perseus Publishing.

Barry, B. and R. Hardin, Eds. (1982). Rational man and irrational society. Beverly Hills, CA, Sage.

Beal, D. J., R. R. Cohen, et al. (2003). "Cohesion and performance in groups: A meta-analytic clarification of construct relations." Journal of Applied Psychology 88: 989-1004.

Benkler, Y. (2002). "Coase's penguin, or, Linux and the nature of the firm." The Yale Law Journal 112(3): 369-446.

Benkler, Y. (2006). The wealth of networks: How social production transforms markets and freedom. New Haven, Yale University Press.

Bessen, J. (2005). Open source software: free provision of complex public goods. MIT open source working papers.

Bordieu, R. (1986). The forms of capital. Handbook of theory and research for the sociology of education. J. G. Richardson. New York, Greenwood Press: 241-258.

Borgatti, S. P., C. Jones, et al. (1998). "Network measures of social capital." Connections 21(2): 25-36.

Brooks, F. P. (1975). The mythical man-month: essays on software engineering. Reading, MA, Addison-Wesley.

Brown, J. S. (1998). "Internet technology in support of the concept of "communities of practice." Accounting, Management, and Information Technologies 8: 227-236.

Brown, J. S. and P. Duguid (1991). "Organizational learning and communities of practice." Organization Science 2(1): 40-57.

Brown, J. S. and P. Duguid (2000). The social life of information. Boston, MA, Harvard Business School Press.

Buchanan, M. (2002). Nexus: small worlds and the groundbreaking science of networks. New York, W. W. Norton & Company.

Burt, R. S. (1992). Structural holes: The social structure of competition. Cambridge, Mass., Harvard University Press.

Burt, R. S. (2001). Structural holes versus network closure as social capital. Social capital: theory and research. N. Lin, K. Cook and R. S. Burt. New York, Aldine De Gruyter: 31-56.

Capiluppi, A., P. Lago, et al. (2003). Evidences in the evolution of OS projects through changelog analysis.

Carley, K. M. (1995). Computational and mathematical organization theory: perspective and directions. 1995 Informs meetings in Los Angeles, CA.

Carrington, P. J., J. Scott, et al., Eds. (2005). Models and methods in social network analysis. Cambridge, Mass., Cambridge University Press.

Chengalur-Smith, S. and A. Sidorova (2003). Survival of open-source projects: a population ecology perspective. Twenty-Fourth International Conference on Information Systems.

Chesbrough, H. W. (2003). Open innovation: the new imperative for creating and profiting from technology. Boston, Mass., Harvard Business School Press.

Christensen, C. M. (1997). The innovator's dilemma: when new technologies cause great firms to fail. Boston, Mass., Harvard Business School Press.

Coleman, J. S. (1988). "Social capital in the creation of human capital." American Journal of Sociology 94: S95-S120.

Conklin, M., J. Howison, et al. (2005). Collaboration using OSSmole: A repository of FLOSS data and analyses. International Conference on Software Engineering Workshop on Mining Software Repositories, St. Louis, MO.

Crowston, K., H. Annabi, et al. (2004). Towards a portfolio of FLOSS project success measures. The 4th Workshop on Open Source Software Engineering, Edinburgh, Scotland.

Crowston, K., H. Annabi, et al. (2005). Effective work practices for FLOSS development: A model and propositions. 38th Hawaii International Conference on System Sciences - 2005, Hawaii.

Crowston, K. and J. Howison (2004). The social structure of free and open source software development. Syracuse FLOSS Working Paper.

Crowston, K. and J. Howison (2006). "Hierarchy and centralization in free and open source software team communications." Knowledge, Technology, & Policy 18(4): 65-85.

Crowston, K. and B. Scozzi (2002). "Open source software projects as virtual organizations: competency rallying for software development." IEE Proceedings Software 149(1): 3-17.

Cummings, T. G. (1978). Self-regulating work groups: A socio-technical synthesis." The Academy of Management Review 3(3): 625-634.

Davila, T., M. J. Epstein, et al. (2006). Making innovation work: how to manage it, measure it, and profit from it. Upper Saddle River, NJ, Wharton School Publishing.

Dawes, R. (1980). "Social dilemmas." Annual Review of Psychology 31: 169-193.

Evans, D. S. and B. J. Reddy (2003). "Government preferences for promoting open source software: a solution in search of a problem." Michigan Telecommunications and Technology Law Review 9: 313-394.

Everett, M. G. and S. P. Borgatti (1999). "The centrality of groups and classes." Journal of Mathematical Sociology 23(3): 181-201.

Freeman, L. C. (2004). The development of social network analysis: a study in the sociology of science. Vancouver, BC, Empirical Press.

Fukuyama, F. (1995). The social virtues and the creation of prosperity. London, Hamish Hamilton.

Gao, Y., V. Freeh, et al. (2003). Analysis and modeling of open source software community. North American Association for Computational Social and Organizational Science (NAACSOS) Conference 2003.

German, D. and A. Mockus (2003). Automating the measurement of open source projects. The 3rd Workshop on Open Source Software Engineering, Portland, OR.

Gongla, P. and C. R. Rizzuto (2001). "Evolving communities of practice: IBM global services experience." IBM Systems Journal 40(4): 842-862.

Granovetter, M. (1973). "The strength of weak ties." American Journal of Sociology 78: 1360-1380.

Granovetter, M. (1985). "Economic action and social structure: the problem of embeddedness." American Journal of Sociology 91(3): 481-510.

Grewal, R, G.L. Lilien, et. al. (2006). "Location, location, location: How network embeddedness affects project success in open source systems." Management Science 52(7): 1043-1056.

Hackman, J. R. (1986). The design of work teams. The handbook of organizational behavior. J. W. Lorsch. Englewood Cliffs, NJ, Prentice-Hall: 315-342.

Hahsler, M. and S. Koch (2005). Discussion of a large-scale open source data collection methodology. 38th Hawaii International Conference on System Sciences - 2005, Hawaii.

Hardin, G. (1968). "The tragedy of the commons." Science 162: 1243-1248.

Hardin, R. (1982). Collective action. Baltimore, The John Hopkins University Press.

Healy, K. and A. Schussman (2003). The ecology of open-source software development. Working Paper: Department of Sociology, University of Arizona. URL: http://www.opensource.mit.edu/papers/healyschussman.pdf.

Hinds, D. and Lee, R.M. (Forthcoming). Assessing the social network health of virtual communities. Handbook of Research on Socio-Technical Design and Social Networking Systems . B. Whitworth and A. de Moor, to appear.

Howell, J.P., Dorfman, P.W., S. Kerr (1986). "Moderator variables in leadership research." The Academy of Management Review 11(1): 88-102.

Howison, J. and K. Crowston (2004). The perils and pitfalls of mining SourceForge. Mining Software Repositories Workshop, International Conference on Software Engineering - 2004, Edinburgh, Scotland.

Hunt, F. and P. Johnson (2002). On the Pareto distribution of SourceForge projects. Open Source Software Development Workshop, Newcastle.

Huysman, M., E. Wenger, et al. (2003). Communities and technologies: proceedings of the first international conference on communities and technologies; C&T 2003. First international conference on communities and technologies; C&T 2003, Kluwer Academic Publishers.

Iannacci, F. (2003). "The Linux managing model." First Monday 8(12).

Katzy, B. R. and K. Crowston (2000). "A process theory of competency rallying in engineering projects." Submitted to IEEE Transactions on Engineering Management.

Kauffman, S. (1993). The origins of order. Oxford, UK, Oxford University Press.

Kerr, S. and J. Jermier (1978). "Substitutes for leadership: Their meaning and measurement." Organizational Behavior and Human Performance 22: 375-403.

Kogut, B. and A. Metiu (2001). "Open-source software development and distributed innovation." Oxford Review of Economic Policy 17(2): 248-264.

Kozlowski, S. W. J. and B. S. Bell (2003). Work groups and teams in organizations. Handbook of psychology: Industrial and organizational psychology. W. C. Borman, D. R. Ilgen and r. Klimoski. New York, Wiley: 333-375.

Krackhardt, D. (1999). "The ties that torture: Simmelian tie analysis in organizations." Research in the Sociology of Organizations 16: 183-210.

Krishnamurthy, S. (2002). "Cave or community? An empirical examination of 100 mature open source projects." First Monday 7(6).

Kuhn, T. S. (1996). The structure of scientific revolutions (third edition). Chicago, IL. The University of Chicago Press.

Lacy, S. (2005). Open source - now it's an ecosystem. BusinessWeek Online.

Lakhani, K. R., B. Wolf, et al. (2002). Hacker Survey (release 0.3), Boston Consulting Group.

Lakhani, K. R. and R. G. Wolf (2005). Why hackers do what they do: understanding motivation and effort in free/open source software projects. Perspectives on free and open source software. J. Feller, B. Fitzgerald, S. Hissam and K. R. Lakhani. Cambridge, MA, MIT Press.

Lanzara, G. F. and M. Morner (2003). The knowledge ecology of open-source software projects. 19th EGOS Colloquium (European Group of Organizational Studies), Copenhagen.

Lave, J. and E. Wenger (1991). Situated learning: legitimate peripheral participation. Cambridge, MA, Cambridge University Press.

Lee, G.K. and R.E. Cole (2003). "From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development." Organization Science 14(6): 633-649.

Lerner, J. and J. Tirole (2002). "The simple economics of open source." Journal of Industrial Economics 52: 197-234.

Lessig, L. (2001). The future of ideas. New York, Random House, Inc.

Lin, N. (2001). Building a network theory of social capital. Social capital: theory and research. N. Lin, K. Cook and R. S. Burt. New York, Aldine De Gruyter: 3-29.

Lomi, A. and P. Pattison (2004). "Introduction to the CMOT special issue on mathematical representations and models for the analysis of social networks within and between organizations." Computational and Mathematical Organization Theory 10: 5-15.

Luhmann, N. (1984 (translated 1995)). Social Systems. Stanford, CA, Stanford University Press.

Luri, J.S. and M.S. Raisinghani (2001). "An empirical study of best practices in virtual teams." Information & Management 38: 523-544.

MacCormack, A., J. Rusnak, et. al. (2006). "Exploring the structure of complex software designs: An empirical study of open source and proprietary code." Management Science 52(7): 1015-1030.

Madey, G., V. Freeh, et al. (2002). The open source software development phenomenon: an analysis based on social network theory. Eighth Americas Conference on Information Systems, Dallas, Texas.

Madey, G., V. Freeh, et al. (2004). Modeling the free/open source software community: a quantitative investigation. Free/open source software development. S. Koch. Hershey, PA, Idea Group: 203-220.

Malone, M. and W. Davidow (1992). Virtual corporation. Forbes 150: 102-107.

March, J. G. (1991). "Exploration and exploitation in organizational learning." Organization Science 2(1): 71-87.

Markus, M. L., B. Manville, et al. (2000). "What makes a virtual organization work?" Sloan Management Review 42: 13-26.

Mertler, C. A. and R. A. Vannatta (2005). Advanced and multivariate statistical methods. Los Angeles, CA, Pyrczak Publishing.

Moreno, J. (1934). Who shall survive? New York, Beacon Press.

Morner, M. (2003). The emergence of open-source software projects: how to stabilize self-organizing processes in emergent systems. Autopoietic organization theory: drawing on Niklas Luhmann's social system perspective. T. Bakken and T. Hernes. Oslo, Copenhagen Business School Press: 259-271.

Mowshowitz, A. (2002). Virtual organization: toward a theory of societal transformation stimulated by information technology, Quorum Books.

Muffatto, M. and M. Faldani (2003). "Open source as a complex adaptive system." Emergence 5(3): 83-100.

Nahapiet, J. and S. Ghoshal (1998). "Social capital, intellectual capital, and organizational advantage." Academy of Management Review 23(2): 242-266.

Obstfeld, D. (2005). "Social networks, the tertius iungens orientation, and involvement in innovation." Administrative Science Quarterly 50: 100-130.

Oh, H., M.-H. Chung, et al. (2004). "Group social capital and group effectiveness: The role of informal socializing ties." Academy of Management Journal 47: 860-875.

Olson, M. J. (1965). The logic of collective action. Cambridge, MA, Harvard University Press.

O'Reilly, T. (1999). "Lessons from open-source software development." Communications of the ACM 42(4): 32-37.

OSI (2004). The open source definition, Open Source Initiative; URL: http://www.opensource.org/docs/definition.html.

Pavlicek, R. C. (2000). Embracing insanity: open source software development. Indianapolis, IN, Sams.

Putnam, R. D. (2000). Bowling alone: the collapse and revival of American community. New York, Simon and Schuster.

Raymond, E. S. (1998). "Homesteading the noosphere." First Monday 3(10).

Raymond, E. S. (1999). The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary. Sebastopol, CA, O'Reilly & Associates, Inc.

Rothfuss, G. J. (2002). A framework for open source projects. Department of Information Technology. Zurich, Switzerland, University of Zurich: 157.

Savannah (2005). URL: http://savannah.gnu.org/.

Scacchi, W. (2002). "Understanding the requirements for developing open source software systems." IEE Software Proc. 149(1): 24-39.

Schenkel, A., R. Teigland, et al. (2000). Theorizing communities of practice: a social network approach. Academy of Management: Organization and Management Theory Division.

Scott, J. (2000). Social network analysis: a handbook (second edition). London, Sage Publications.

Scozzi, B., K. Crowston, et. al. (2008). Shared mental models among open source software developers. 41st Hawaii International Conference on System Sciences - 2008, Hawaii.

Simon, H. A. (1976). Administrative behavior (3rd edition). New York, Free Press.

SourceForge (2005). URL: http://www.sourceforge.net.

Stallman, R. (1985). The GNU Manifesto. URL: http://www.gnu.org/gnu/manifesto.html.

Stewart, K. J. and T. Ammeter (2002). An exploratory study of factors influencing the level of vitality and popularity of open source projects. Twenty-Third International Conference on Information Systems.

Strader, T. J., F.-R. Lin, et al. (1998). "Information infrastructure for electronic virtual organization management." Decision Support Systems 23: 75-94.

Sturmer, M. (2005). Open source community building. MIT Open Source collection.

Tabachnick, B.G. and Fidell, L.S. (2007). Using multivariate statistics. Boston, MA, Pearson.

Teigland, R. (2003). Knowledge networking: structure and performance in networks of practice. Institute of International Business. Stockholm, Stockholm School of Economics.

Trombly, M. (2005). Open source may help China curb software piracy. CIO Insight.

UCINET (2005). URL: http://www.analytictech.com/ucinet.htm.

von Hippel, E. (2001). "Innovation by user communities: learning from open-source software." MIT Sloan Management Review 42(4): 82-86.

von Hippel, E. and G. von Krogh (2003). "Open source software and the "private-collective" innovation model: issues for organization science." Organization Science 14(2): 209-223.

von Krogh, G. (2003). "Open-source software development." Sloan Management Review: 14-18.

von Krogh, G., S. Spaeth, et. al. (2005). Knowledge reuse in open source software: An exploratory study of 15 open source projects. 38th Hawaii International Conference on System Sciences - 2005, Hawaii.

von Krogh, G and E. von Hippel (2006). "The promise of research on open source software." Management Science 52(7): 975-983.

Wagstrom, P. A. (2004). Toward a simulation model of open source software development. NAACSOS, Pittsburgh, PA.

Wagstrom, P. A., J. D. Herbsleb, et al. (2005). A social network approach to free/open source software simulation. First International Conference on Open Source Systems, Genova.

Wasko, M. M. and R. Teigland (2002). The provision of online public goods: examining social structure in a network of practice. Twenty-Third International Conference on Information Systems - 2002.

Wasserman, S. and K. Faust (1994). Social network analysis: methods and applications. Cambridge, UK, Cambridge University Press.

Watts, D. J. (2003). Six degrees: the science of a connected age. New York, W. W. Norton & Company.

Webb, E. J., D. T. Campbell, et al. (2000). Unobtrusive measures. Thousand Oaks, CA, Sage.

Weber, S. (2003). Open source software in developing economies. Social Science Research Council Working Paper.

Weber, S. (2004). The success of open source. Cambridge, Mass., Harvard University Press.

Weick, K. E. (1989). "Theory construction as disciplined imagination." Academy of Management Review 14(4): 516-531.

Wenger, E. (1998). Communities of practice: learning, meaning, and identity. Cambridge, UK, Cambridge University Press.

Wenger, E., R. McDermott, et al. (2002). Cultivating communities of practice: a guide to managing knowledge. Boston, Mass, Harvard Business School Press.

West, J. and S. O'Mahony (2005). Contrasting community building in sponsored and community founded open source projects. 38th Hawaii International Conference on System Sciences - 2005, Hawaii.

Xu, J., Y. Gao, et al. (2005). A topological analysis of the open source software development community. 38th Hawaii International Conference on System Sciences - 2005, Hawaii.

Yamauchi, Y., M. Yokozawa, et. al. (2000). Collaboration with lead media: How open source software succeeds. ACM 2000 Conf. Comput. Supported Cooperative Work, Philadelphia.

Yang, H. and J. Tang (2004). "Team structure and team performance in IS development: A social network perspective." Information & Management 41: 335-349.

Ye, Y., K. Nakakoji, et al. (2005). The co-evolution of systems and communities in free and open source software development. Free/open source software development. S. Koch. Hershey, PA, Idea Group Inc (IGI): 59-82.

APPENDICES

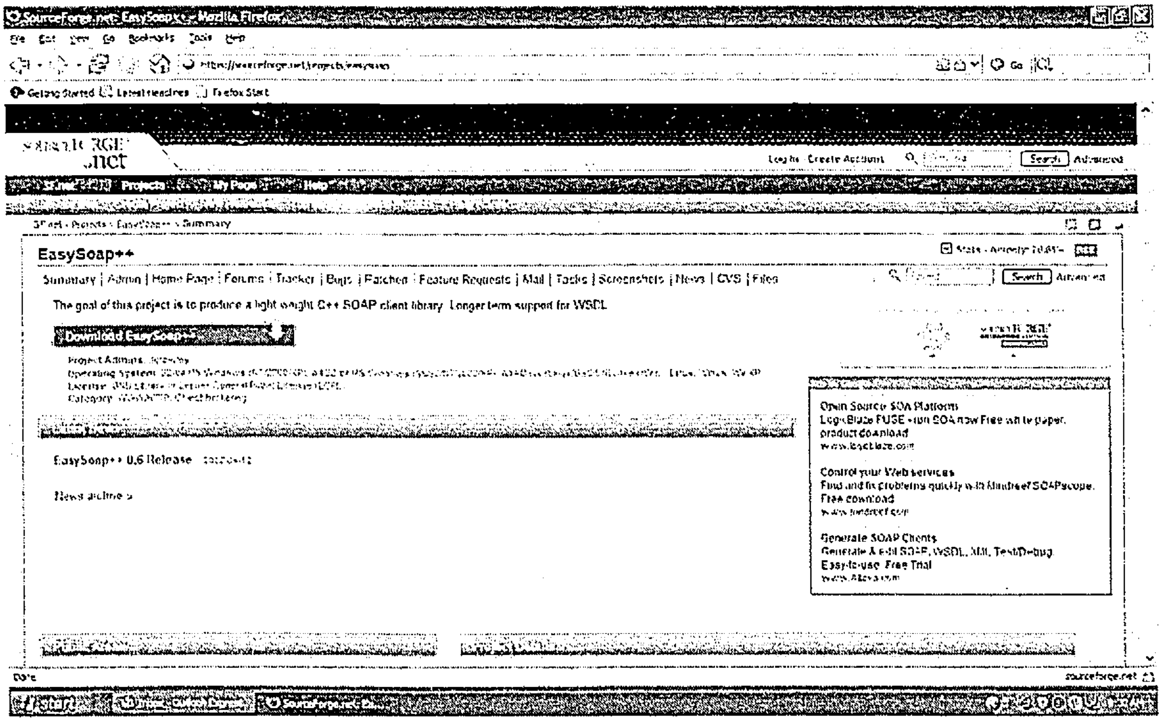
Appendix A

Sourceforge.net screen images

This appendix contains screen images obtained from the SourceForge.net web site.

Sourceforge.net screen images

Figure A-1
SourceForge Project Home Page Summary Screen



SourceForge.net: EasySoap++ - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Getting Started Latest Headlines Firefox Start
Log in Create Account Search Advanced
SF.net Projects My Page Help
SF.net › Projects › EasySoap++ › Summary
Summary | Admin | Home Page | Forums | Tracker | Bugs | Patches | Feature Requests | Mail | Tasks | Screenshots | News | CVS | Files
Search Advanced
Download EasySoap++
Done
sourceforge.net

EasySoap++

The goal of this project is to produce a light weight C++ SOAP client library. Longer term support for WSDL

Project Admins:
Operating System:
License:
Category:

EasySoap++ 0.6 Release

News archive »

Open Source SOA Platform
LogicBlaze FUSE - run SOA now Free white paper, product download
www.logicblaze.com

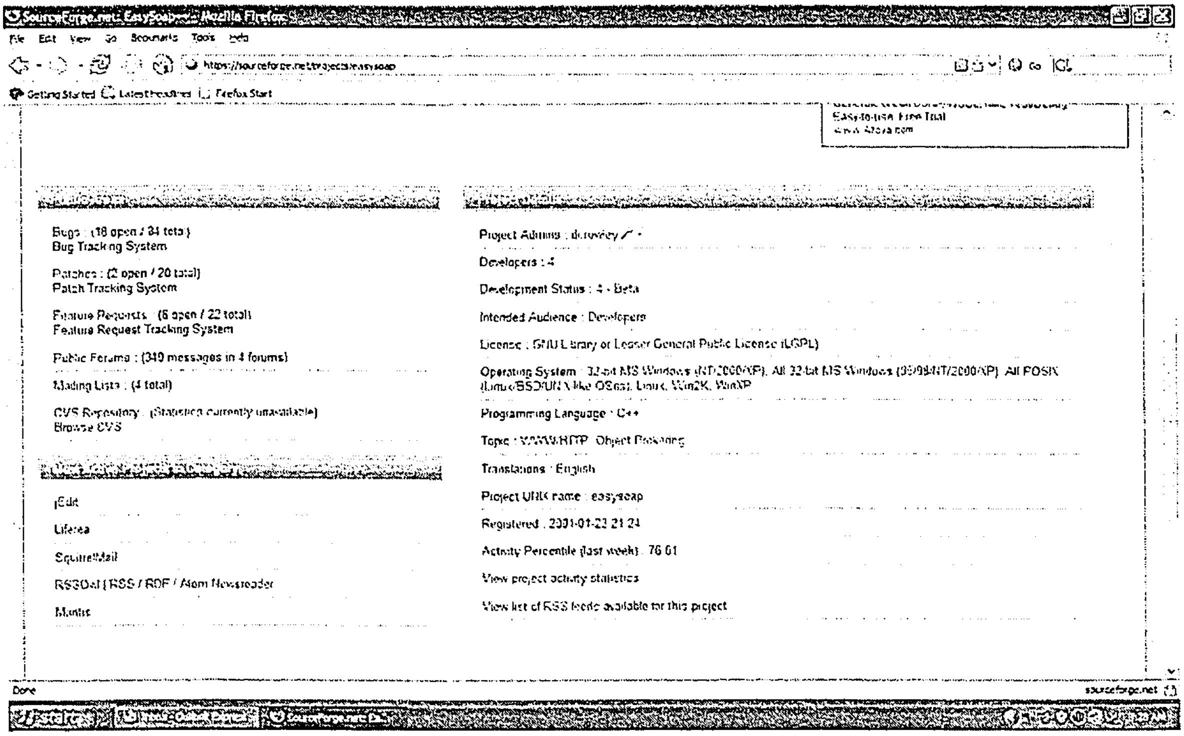
Control your Web services
Find and fix problems quickly with Mindreef SOAPscope. Free download
www.mindreef.com

Generate SOAP Clients
Generate & edit SOAP, WSDL, XML, Test/Debug. Easy-to-use. Free Trial
www.Altova.com

Source URL: https://sourceforge.net/projects/easysoap

Sourceforge.net screen images

Figure A-2
SourceForge Project Home Page Project Details and Public Areas


SourceForge.net: EasySoap++ - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://sourceforge.net/projects/easysoap
Getting Started Latest Headlines Firefox Start
Easy-to-use. Free Trial
www.Altova.com
Done
sourceforge.net

Bugs : (18 open / 34 total)
Bug Tracking System

Patches : (2 open / 20 total)
Patch Tracking System

Feature Requests : (6 open / 22 total)
Feature Request Tracking System

Public Forums : (349 messages in 4 forums)

Mailing Lists : (4 total)

CVS Repository : (Statistics currently unavailable)
Browse CVS

[Edit

Lifetea

SquirrelMail

RSSOwl | RSS / RDF / Atom Newsreader

Mantis

Project Admins : dcrowley

Developers : 4

Development Status : 4 - Beta

Intended Audience : Developers

License : GNU Library or Lesser General Public License (LGPL)

Operating System : 32-bit MS Windows (NT/2000/XP), All 32-bit MS Windows (95/98/NT/2000/XP), All POSIX (Linux/BSD/UNIX-like OSes), Linux, Win2K, WinXP

Programming Language : C++

Topic : WWW/HTTP, Object Brokering

Translations : English

Project UNIX name : easysoap

Registered : 2001-01-23 21:24

Activity Percentile (last week) : 76.01

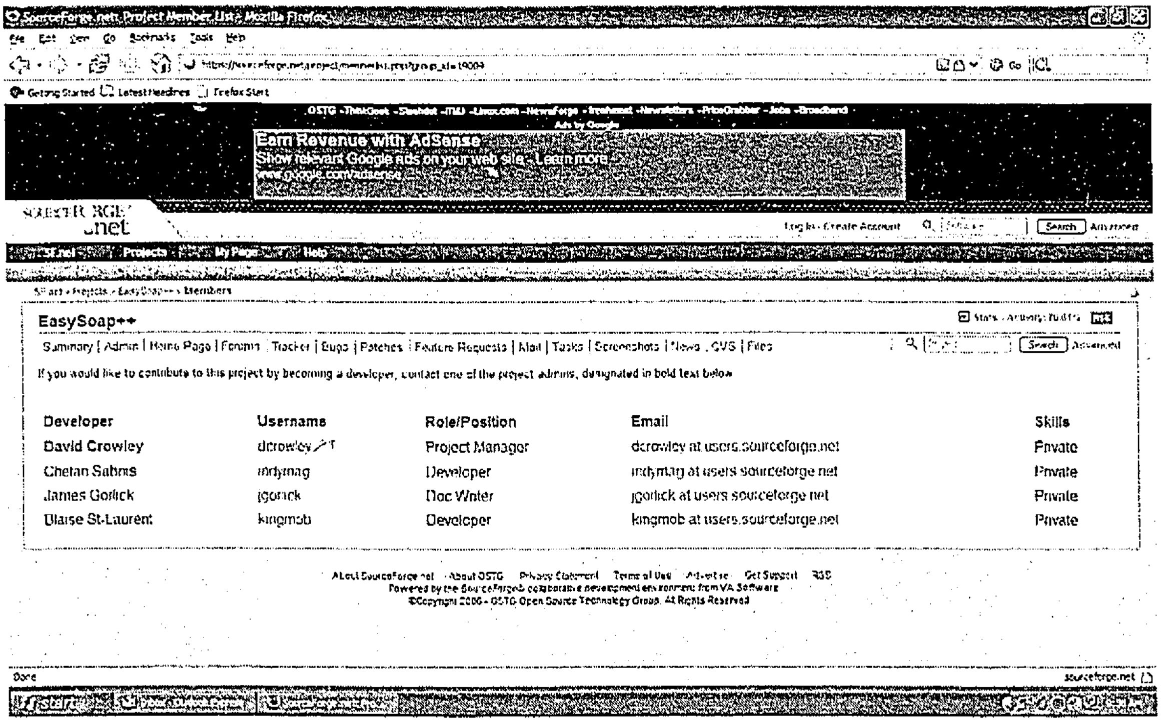
View project activity statistics

View list of RSS feeds available for this project

Source URL: https://sourceforge.net/projects/easysoap

Sourceforge.net screen images

Figure A-3
SourceForge Project Member Page


SourceForge.net: Project Member List - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Getting Started Latest Headlines Firefox Start
OSTG - ThinkGeek - Slashdot - ITMJ - Linux.com - NewsForge - freshmeat - Newsletters - PriceGrabber - Jobs - Broadband
Ads by Google
Earn Revenue with AdSense
Show relevant Google ads on your web site - Learn more
www.google.com/adsense
SOURCEFORGE.net
Log In - Create Account
Search
Advanced
SF.net
Projects
My Page
Help
SF.net » Projects » EasySoap++ » Members
Summary | Admin | Home Page | Forums | Tracker | Bugs | Patches | Feature Requests | Mail | Tasks | Screenshots | News | CVS | Files
Search
Advanced
About SourceForge.net · About OSTG · Privacy Statement · Terms of Use · Advertise · Get Support · RSS
Done
sourceforge.net

EasySoap++

If you would like to contribute to this project by becoming a developer, contact one of the project admins, designated in bold text below.

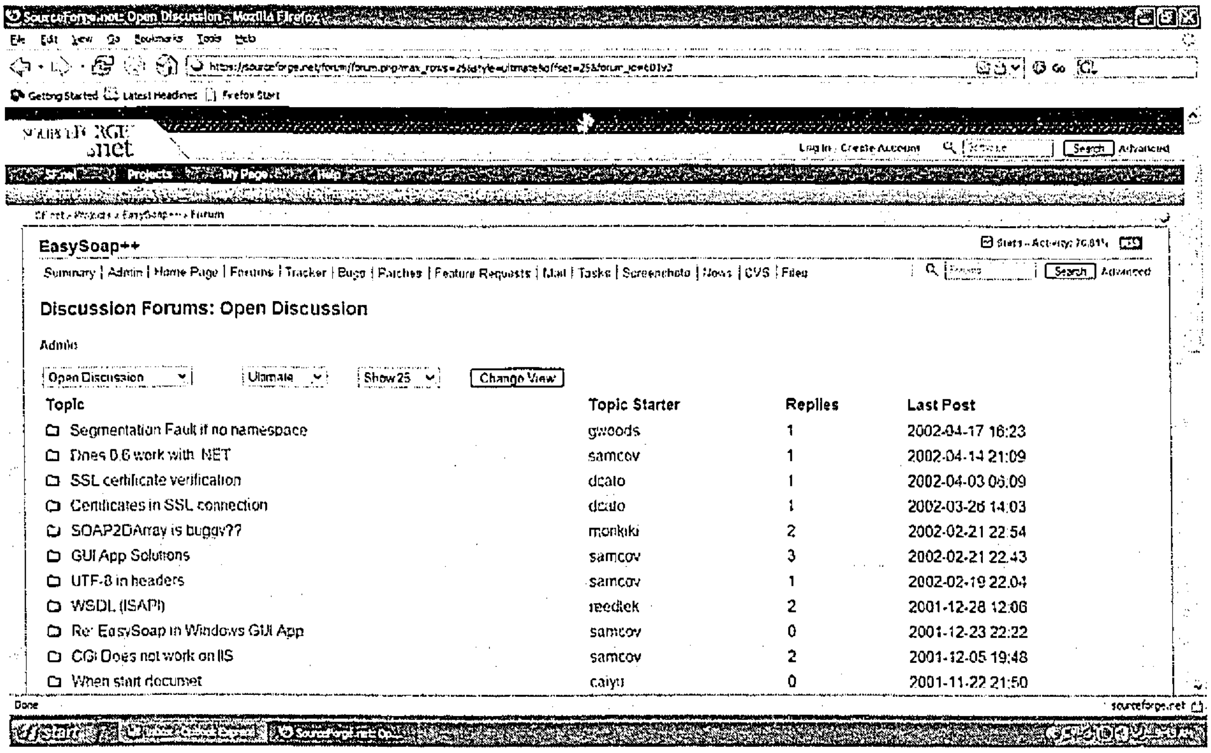
Developer	Username	Role/Position	Email	Skills
David Crowley	dcrowley	Project Manager	dcrowley at users.sourceforge.net	Private
Chetan Sabnis	indymag	Developer	indymag at users.sourceforge.net	Private
James Gorlick	jgorlick	Doc Writer	jgorlick at users.sourceforge.net	Private
Blaise St-Laurent	kingmob	Developer	kingmob at users.sourceforge.net	Private

Powered by the SourceForge® collaborative development environment from VA Software
©Copyright 2006 - OSTG Open Source Technology Group. All Rights Reserved

Source URL: https://sourceforge.net/project/memberlist.php?group_id=19009

Sourceforge.net screen images

Figure A-4
SourceForge Project Forum Page Topic Listing


SourceForge.net: Open Discussion - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Getting Started Latest Headlines Firefox Start
Log In - Create Account
Search Advanced
SF.net Projects My Page Help
SF.net » Projects » EasySoap++ » Forum
Stats - Activity: 76.81%
Summary | Admin | Home Page | Forums | Tracker | Bugs | Patches | Feature Requests | Mail | Tasks | Screenshots | News | CVS | Files
Search Advanced
Done
sourceforge.net
Start

EasySoap++

Discussion Forums: Open Discussion

Admin

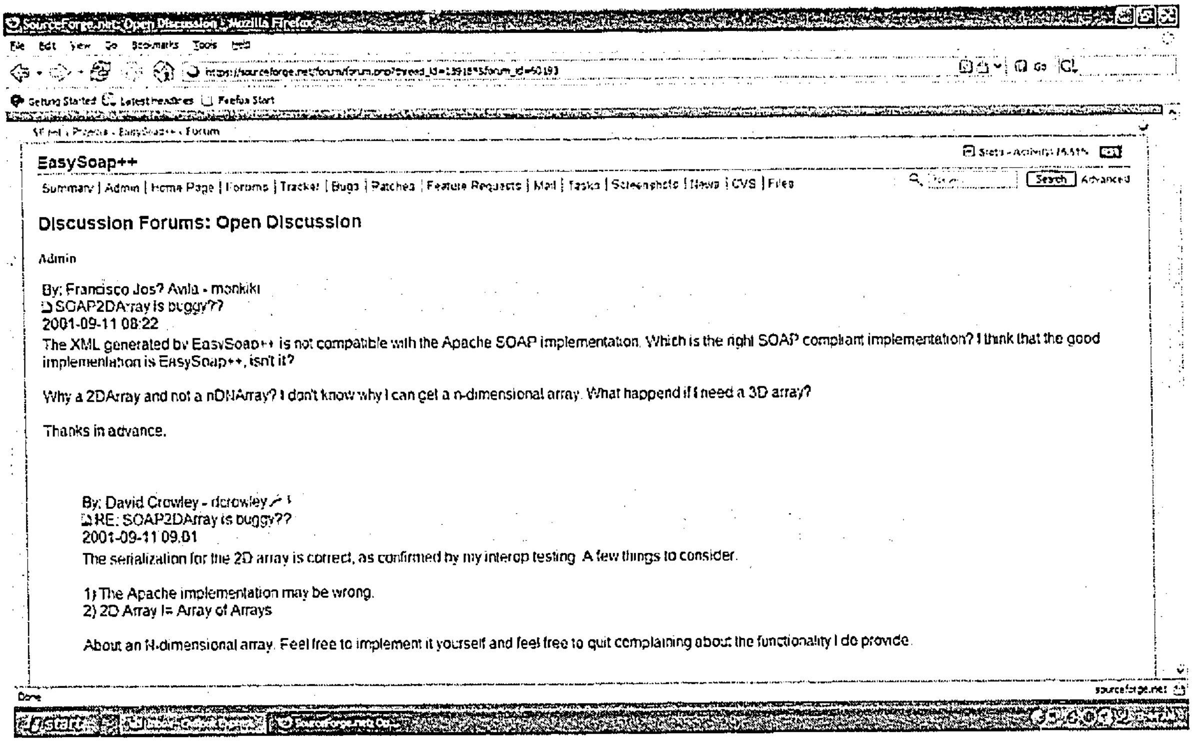
Open Discussion | Ultimate | Show 25 | Change View

Topic	Topic Starter	Replies	Last Post
Segmentation Fault if no namespace	gwoods	1	2002-04-17 16:23
Does 0.6 work with .NET	samcov	1	2002-04-14 21:09
SSL certificate verification	dcato	1	2002-04-03 06:09
Certificates in SSL connection	dcato	1	2002-03-26 14:03
SOAP2DArray is buggy??	monkiki	2	2002-02-21 22:54
GUI App Solutions	samcov	3	2002-02-21 22:43
UTF-8 in headers	samcov	1	2002-02-19 22:04
WSDL (ISAPI)	reedtek	2	2001-12-28 12:06
Re: EasySoap in Windows GUI App	samcov	0	2001-12-23 22:22
CGI Does not work on IIS	samcov	2	2001-12-05 19:48
When start documet	caiyu	0	2001-11-22 21:50

Source URL: https://sourceforge.net/forum/forum.php?forum_id=60193

Sourceforge.net screen images

Figure A-5
SourceForge Project Forum Page Discussion Text


SourceForge.net: Open Discussion - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Getting Started Latest Headlines Firefox Start
Summary | Admin | Home Page | Forums | Tracker | Bugs | Patches | Feature Requests | Mail | Tasks | Screenshots | News | CVS | Files
Search Advanced
Done
sourceforge.net
start

EasySoap++

Discussion Forums: Open Discussion

Admin

By: Francisco Jos? Avila - monkiki
SOAP2DArray is buggy??
2001-09-11 08:22
The XML generated by EasySoap++ is not compatible with the Apache SOAP implementation. Which is the right SOAP compliant implementation? I think that the good implementation is EasySoap++, isn't it?

Why a 2DArray and not a nDNArray? I don't know why I can get a n-dimensional array. What happend if I need a 3D array?

Thanks in advance.

By: David Crowley - dcrowley
RE: SOAP2DArray is buggy??
2001-09-11 09:01
The serialization for the 2D array is correct, as confirmed by my interop testing. A few things to consider.

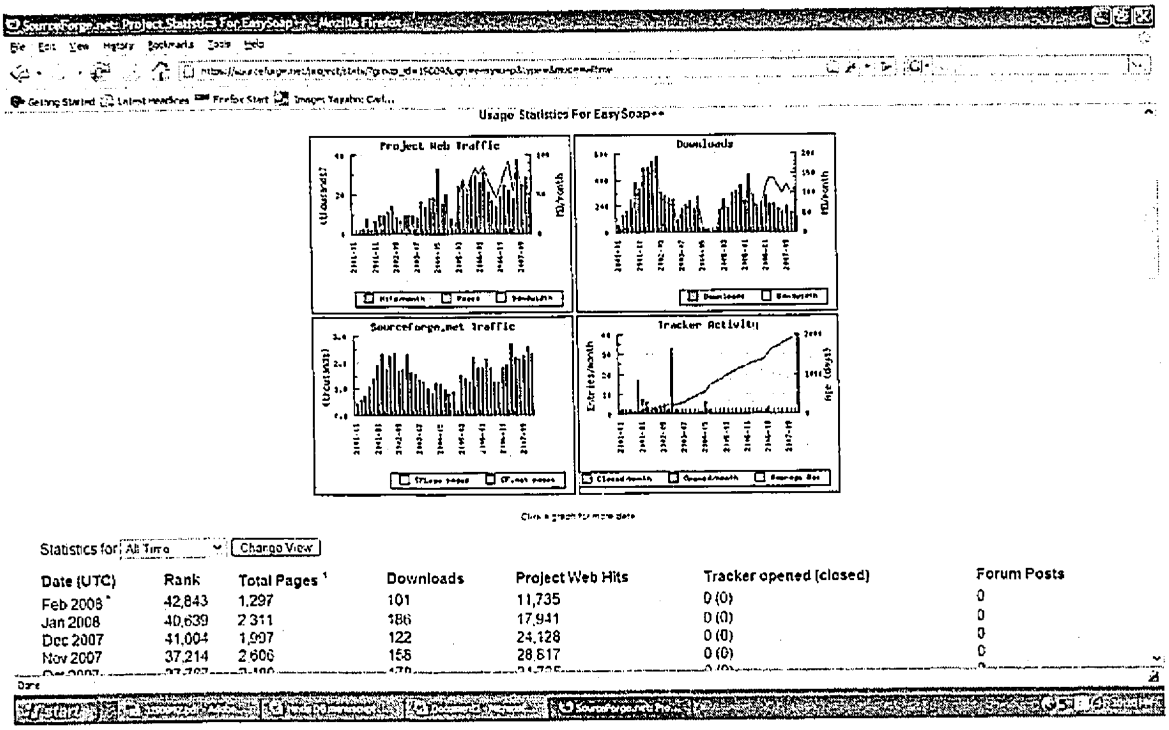
1) The Apache implementation may be wrong.
2) 2D Array != Array of Arrays

About an N-dimensional array. Feel free to implement it yourself and feel free to quit complaining about the functionality I do provide.

Source URL:
https://sourceforge.net/forum/forum.php?thread_id=1254140&forum_id=60193

<u>Sourceforge.net screen images</u>

Figure A-6
SourceForge Project Statistics Page



SourceForge.net: Project Statistics For EasySoap++ - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Getting Started Latest Headlines Firefox Start
Usage Statistics For EasySoap++
Project Web Traffic
Downloads
Sourceforge.net Traffic
Tracker Activity
Click a graph for more data
Done
start

Statistics for All Time Change View

Date (UTC)	Rank	Total Pages [1]	Downloads	Project Web Hits	Tracker opened (closed)	Forum Posts
Feb 2008 *	42,843	1,297	101	11,735	0 (0)	0
Jan 2008	40,639	2 311	186	17,941	0 (0)	0
Dec 2007	41,004	1,997	122	24,128	0 (0)	0
Nov 2007	37,214	2 606	158	28 817	0 (0)	0

Source URL:
<u>https://sourceforge.net/project/stats/?group_id=19009&ugn=easysoap&type=&mode=alltime</u>

Appendix B

University of Notre Dame Research Database

SourceForge.net Research Data

SourceForge.net is the world's largest Open Source software development web site, with the largest repository of Open Source code and applications available on the Internet. Owned and operated by OSTG, Inc. ("OSTG"), SourceForge.net provides free services to Open Source developers. The SourceForge.net web site is database driven and the supporting database includes historic and status statistics on over 140,000 projects and over 1.5 million registered users' activities at the project management web site. OSTG has shared certain SourceForge.net data with the University of Notre Dame for the sole purpose of supporting academic and scholarly research on the Free/Open Source Software phenomenon. OSTG has given Notre Dame permission to in turn share this data with other academic researchers studying the Free/Open Source Software phenomenon.

Source URL: http://www.nd.edu/~oss/Data/data.html

Release of the SourceForge.net Research Data

To advance the understanding of, and research on, the Free/Open Source Software phenomenon, portions of the data that may support such research, will be made available to academic or scholarly researchers. All requests for data must be submitted in writing (e-mail) to the Notre Dame PI, (Greg Madey). Only academic and scholarly researchers are eligible to receive the data. To receive the data, a short questionnaire and agreement must be completed, signed and returned. A wiki for users of the research data is available here.

Source URL: http://www.nd.edu/~oss/Data/data.html

University of Notre Dame Research Database

Description of Data Available

SourceForge.net uses relational databases to store project management activity and statistics. There are over 100 relations (tables) in the data dumps provided to Notre Dame. Some of the data have been removed for security and privacy reasons. SourceForge.net cleanses the data of personal information and strips out all OSTG specific and site functionality specific information. On a monthly basis, a complete dump of the databases (minus the data dropped for privacy and security reasons) is shared with Notre Dame. The Notre Dame researchers have built a data warehouse comprised of these monthly dumps, with each stored in a separate schema. Thus, each monthly dump is a snapshot of the status of all the SourceForge.net projects at that point in time. As of March 2007, the data warehouse was almost 500 GBytes in size, and is growing at about 25 GBytes per month. Much of the data is duplicated among the monthly dumps, but trends or changes in project activity and structure can be discovered by comparing data from the monthly dumps. Queries across the monthly schema may be used to discover when changes took place, to estimate trends in project activity and participation, or even that no activity, events or changes have taken place. To help researchers determine what data is available, an ER-diagram and the definitions of tables and views in the data warehouse are provided.

Source URL: http://www.nd.edu/~oss/Data/data.html

Appendix C

Libresoft Project Research Database

Libre Software Engineering

Welcome to the Libre Software[1] Engineering web site at the Grupo de Sistemas y Comunicaciones (System and Communication Group, GSyC) at the Universidad Rey Juan Carlos located in Móstoles, near Madrid (Spain).

Libre Software offers Software Engineering scientists the possibility not only of having a closer look at the product that is being created, but also of studying in detail the whole development process and its technical, social and economic consequences.

The main research topic at the Universidad Rey Juan Carlos is the **quantitative measurement of libre software** development patterns and characteristics in order to gain knowledge on the process, mainly by studying the different agents that participate in it, the use of the different development and development-supporting tools as well as the methods that have been followed. The main focus is technically oriented having principally an engineering perspective of the research area in contrast to other research groups which are primarily centered on social and economic aspects.

NEWS: We also drive the **FLOSS Research Planet** which syndicates other research blogs from researchers who investigate libre software.

Source URL: http://libresoft.es/description

Appendix D

Detailed Regression Results

This appendix contains tables with detailed results of regressions which produced a significant result ($p < .05$). These regressions are referred to in Tables 20, 21, and 22, and in the corresponding subsections of Section 5.4.

Detailed Regression Results

Table D-1
Log-Transformed Software Downloads Regressed on Group Density, Controlling for Group Size, Core Size and Conversation Volume
(Unstandardized Coefficients)

Variables	Model 1		Model 2		Model 3	
Group Size	.014***	(.002)	.009***	(.002)	.007**	(.003)
Core Size	.003	(.021)	.001	(.020)	-.001	(.020)
Conversation Volume	-.001*	(.000)	.000	(.000)	.000	(.000)
Group Density			-5.547***	(1.237)	-8.881***	(2.349)
Group Density mean-centered and squared					16.375†	(9.827)
R^2	.324		.410		.421	
F-Statistic	22.184***		23.952***		19.963***	
Adjusted R^2	.309		.393		.400	
ΔR^2			.086		.012	
ΔF-Statistic			20.106		2.777	

Standard errors are in parentheses

* p < .05 ; ** p < .01 ; *** p < .001; n = 143 groups

† p = .098

Detailed Regression Results

Table D-2
Log-Transformed Page Views Regressed on Group Density,
Controlling for Group Size, Core Size and Conversation Volume
(Unstandardized Coefficients)

Variables	Model 1		Model 2		Model 3	
Group Size	.011***	(.002)	.007**	(.002)	.005†	(.003)
Core Size	.042†	(.021)	.040†	(.020)	.038†	(.020)
Conversation Volume	.000	(.000)	.000	(.000)	.000	(.000)
Group Density			-4.871***	(1.285)	-8.353**	(2.439)
Group Density mean-centered and squared					17.097†	(10.203)
R^2	.291		.358		.371	
F-Statistic	19.019***		19.233***		16.150***	
Adjusted R^2	.276		.339		.348	
ΔR^2			.067		.013	
ΔF-Statistic			14.382		2.808	

Standard errors are in parentheses

* $p < .05$; ** $p < .01$; *** $p < .001$; n = 143 groups

† $p = .053$ (Model 1 Core Size), .053 (Model 2 Core Size), .057 (Model 3 Group Size)

† $p = .064$ (Model 3 Core Size), .096 (Model 3 Group Density mean-centered and squared)

Detailed Regression Results

Table D-3
Log-Transformed Software Releases Regressed on Core Density,
Controlling for Group Size, Core Size and Conversation Volume
(Unstandardized Coefficients)

Variables	Model 1		Model 2		Model 3	
Group Size	-.001	(.002)	-.002	(.002)	-.002	(.002)
Core Size	-.026	(.021)	-.044*	(.022)	-.044†	(.022)
Conversation Volume	.000	(.000)	.001*	(.000)	.001*	(.000)
Core Density			-.570*	(.261)	-.615	(.412)
Core Density mean-centered and squared					.121	(.855)
R^2	.039		.071		.071	
F-Statistic	1.876		2.641*		2.102†	
Adjusted R^2	.018		.044		.037	
ΔR^2			.032		.000	
ΔF-Statistic			4.781		.020	

Standard errors are in parentheses

* p < .05 ; ** p < .01 ; *** p < .001; n = 143 groups

† p = .050 (Model 3 Core Size), .069 (Model 3 F-Statistic)

Detailed Regression Results

Table D-4
Log-Transformed Page Views Regressed on Core Density,
Controlling for Group Size, Core Size and Conversation Volume
(Unstandardized Coefficients)

Variables	Model 1		Model 2		Model 3	
Group Size	.011***	(.002)	.011***	(.002)	.011***	(.002)
Core Size	.042†	(.021)	.033	(.023)	.036	(.023)
Conversation Volume	.000	(.000)	.000	(.000)	.000	(.000)
Core Density			-.267	(.272)	-.977*	(.422)
Core Density mean-centered and squared					1.910*	(.877)
R^2	.291		.296		.319	
F-Statistic	19.019***		14.502***		12.864***	
Adjusted R^2	.276		.276		.295	
ΔR^2			.005		.024	
ΔF-Statistic			.964		4.741	

Standard errors are in parentheses

$* p < .05$; $** p < .01$; $*** p < .001$;   n = 143 groups

† p = .053

Detailed Regression Results

Table D-5
Log-Transformed Code Commits Regressed on Administrator Membership Degree, Controlling for Group Size, Core Size and Conversation Volume (Unstandardized Coefficients)

Variables	Model 1		Model 2		Model 3	
Group Size	-.005†	(.003)	-.005†	(.003)	-.006*	(.003)
Core Size	.059*	(.029)	.059*	(.029)	.060*	(.029)
Conversation Volume	.001*	(.000)	.001*	(.000)	.001*	(.000)
Administrator Membership Degree			.022	(.063)	.212†	(.110)
Administrator Membership Degree mean-centered and squared					-.040*	(.019)
R^2	.052		.053		.082	
F-Statistic	2.564†		1.941		2.455*	
Adjusted R^2	.032		.026		.049	
ΔR^2			.001		.029	
ΔF-Statistic			.120		4.324	

Standard errors are in parentheses

* p < .05 ;   ** p < .01 ;   *** p < .001;   n = 143 groups

† p = .073 (Model 1 Group Size), .073 (Model 2 Group Size)

† p = .057 (Model 3 Administrator Membership Degree), .057 (Model 1 F-Statistic)

Detailed Regression Results

Table D-6
Log-Transformed Software Releases Regressed on Administrator Class Centrality, Controlling for Group Size, Core Size and Conversation Volume
(Unstandardized Coefficients)

Variables	Model 1		Model 2		Model 3	
Group Size	-.001	(.002)	.002	(.002)	.002	(.002)
Core Size	-.026	(.021)	-.010	(.021)	-.011	(.021)
Conversation Volume	.000	(.000)	.000	(.000)	.000	(.000)
Administrator Class Centrality			.963**	(.326)	.890*	(.358)
Administrator Class Centrality mean-centered and squared					-.515	(1.026)
R^2	.039		.096		.098	
F-Statistic	1.876		3.660**		2.963*	
Adjusted R^2	.018		.070		.065	
ΔR^2			.057		.002	
ΔF-Statistic			8.701		.252	

Standard errors are in parentheses

* p < .05 ; ** p < .01 ; *** p < .001; n = 143 groups

Detailed Regression Results

Table D-7
Log-Transformed Page Views Regressed on Administrator Class Centrality, Controlling for Group Size, Core Size and Conversation Volume
(Unstandardized Coefficients)

Variables	Model 1		Model 2		Model 3	
Group Size	.011***	(.002)	.011***	(.002)	.011***	(.002)
Core Size	.042†	(.021)	.038†	(.022)	.042†	(.022)
Conversation Volume	.000	(.000)	.000	(.000)	.000	(.000)
Administrator Class Centrality			-.247	(.346)	.084	(.373)
Administrator Class Centrality mean-centered and squared					2.347*	(1.069)
R^2	.291		.294		.318	
F-Statistic	19.019***		14.342***		12.756***	
Adjusted R^2	.276		.273		.293	
ΔR^2			.003		.024	
ΔF-Statistic			.511		4.822	

Standard errors are in parentheses

* p < .05 ; ** p < .01 ; *** p < .001; n = 143 groups

† p = .053 (Model 1 Core Size), .090 (Model 2 Core Size), .059 (Model 3 Core Size)

VITA

DAVID HINDS

1972	B.S., Engineering Science University of Miami Miami, Florida
1973	M.S., Management Science University of Miami Miami, Florida
1974 – 1980	Metro Dade County Transportation
1980 – 1983	Cordis Corporation
1983	M.B.A. Florida International University Miami, Florida
1983 – 1985	Cordis Bio-Synthetics, Inc.
1985 – 1988	Deloitte Haskins and Sells
1988 – 1998	Trend Distributors
1998 – 2002	The Wurth Group
2003 – 2008	Doctoral Candidate Business Administration Florida International University Miami, Florida

PUBLICATIONS AND PRESENTATIONS

Franceschi, K., Lee, R. M. and Hinds, D. (January 2008). "Engaging e-learning in virtual worlds: supporting group collaboration." Presented at the *41st Hawaii International Conference on System Sciences.*

Hinds, D., Roark, A., Schimpeler, C., and Corradino, J. (1978). "Transportation modeling in a changing world: a Miami case study." *Transportation Planning and Technology* 4: 125-135.

Hinds, D. (1979). "RUCUS scheduling software: A comprehensive status report and assessment." *Transit Journal* 5(1): 17-34.

Hinds, D. (2004). "Micropayments: a technology with a promising but uncertain future." (short note included in "Mobile banking services" by Niina Mallat, Matti Rossi and Virpi Tuunainen). *Communications of the ACM* 47(5): 44.

Hinds, D. (2004). "Critical mass behavior and transaction costs in open source and open content projects." *North American Association for Computational Social and Organizational Science (NAACSOS) Conference 2004*, Pittsburgh, PA (CMU).

Hinds, D. (2005). "Open web learning - achieving creative synergy in the open development and use of e-learning resources." *7th International Conference on Enterprise Information Systems - Doctoral Consortium*, Miami, Florida.

Hinds, D. and Lee, R. M. (2006). "Why do some open source software projects succeed while others fail? Group centrality constructs as predictors of project outcome." *International Sunbelt Social Network Conference XXVI*, Vancouver, British Columbia.

Hinds, D. and Lee, R. M. (January 2008). "Social network structure as a critical success condition for virtual communities." Presented at the *41st Hawaii International Conference on System Sciences.*

Hinds, D. and Pasztor, A. (August 2008). "What's wrong with our concept of knowledge? A case of semantic pathology." To be presented at the *2008 Academy of Management Annual Meeting*, Anaheim, California.

Hinds, D. and Lee, R.M. (Forthcoming). Assessing the social network health of virtual communities. *Handbook of Research on Socio-Technical Design and Social Networking Systems*, Edited by Whitworth, B. and de Moor, A., to appear.

Lee, R. M., Dominguez, C. E., Franceschi, K. and Hinds, D. (2006). "Mitigating culture shock: e-learning cultural affordances." *2nd Workshop on Tourism and ICT: Dynamic and Intelligent Configuration of Tourism Services*, University of Twente, Enschede, The Netherlands.

O'Neil, B. F., Catanese, A. J. and Hinds, D. (1978). Transportation systems. *Handbook of Operations Research: Models and Applications.* J. J. Moder and S. E. Elmaghraby. New York, Van Nostrand Reinhold Company. 2: 477-502.