ESSAYS ON EXPLOITATION AND EXPLORATION IN SOFTWARE DEVELOPMENT

by

Orcun Temizkan

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2012

Approved by:

_____
Dr. Ram Kumar

_____
Dr. Chandrasekar Subramaniam

_____
Dr. Sungjune Park

_____
Dr. Cem Saydam

_____
Dr. Jaya Bishwal

ABSTRACT

ORCUN TEMIZKAN. Essays on exploitation and exploration in software development (Under direction of DR. RAM KUMAR)

Software development includes two types of activities: software improvement activities by correcting faults and software enhancement activities by adding new features. Based on organizational theory, we propose that these activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration). In the context of open source software (OSS) development, developing a patch would be an example of an exploitation activity. Requesting a new software feature would be an example of an exploration activity. This dissertation consists of three essays which examine exploitation and exploration in software development.

The first essay analyzes software patch development (exploitation) in the context of software vulnerabilities which could be exploited by hackers. There is a need for software vendors to make software patches available in a timely manner for vulnerabilities in their products. We develop a survival analysis model of the patch release behavior of software vendors based on a cost-based framework of software vendors. We test this model using a data set compiled from the National Vulnerability Database (NVD), United States Computer Emergency Readiness Team (US-CERT), and vendor web sites. Our results indicate that vulnerabilities with high confidentiality impact or high integrity impact are patched faster than vulnerabilities with high availability impact. Interesting differences in the patch release behavior of software vendors based on software type (new release vs. update) and type of vendor (open source vs. proprietary) are found.

The second essay studies exploitation and exploration in the content of OSS development. We empirically examine the differences between exploitation (patch development) and exploration (feature request) networks of developers in OSS projects in terms of their social network structure, using a data set collected from the SourceForge database. We identify a new category of developers (ambidextrous developers) in OSS projects who contribute to patch development as well as feature request activities. Our results indicate that a patch development network has greater internal cohesion and network centrality than a feature request network. In contrast, a feature request network has greater external connectivity than a patch development network.

The third essay explores ambidexterity and ambidextrous developers in the context of OSS project performance. Recent research on OSS development has studied the social network structure of software developers as a determinant of project success. However, this stream of research has focused on the project level, and has not recognized the fact that software projects could consist of different types of activities, each of which could require different types of expertise and network structures. We develop a theoretical construct for ambidexterity based on the concept of ambidextrous developers. We empirically illustrate the effects of ambidexterity and network characteristics on OSS project performance. Our results indicate that a moderate level of ambidexterity, external cohesion, and technological diversity are desirable for project success. Project success is also positively related to internal cohesion and network centrality. We illustrate the roles of ambidextrous developers on project performance and their differences compared to other developers.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my dissertation advisors, Dr. Ram Kumar, for his continuous support and excellent guidance through my dissertation. I would like to thank him for providing me with an excellent atmosphere for doing research. He constantly encouraged me and generously devoted extensive amount of time in refining key points of my work. He also provided me with the unique opportunity to gain a broad and extensive range of experience in doing research. I would also like to thank my dissertation committee members, Dr. Chandrasekar Subramaniam, Dr. Sungjune Park, Dr. Cem Saydam, and Dr. Jaya Bishwal, for their excellent guidance, helpful feedbacks and invaluable suggestions to my research. I also owe a special thanks to Dr. Cem Saydam for his continuous support and guidance through my doctorate program.

Finally, I would like to express my deepest gratitude to my mother (Huriye Temizkan), my father (Osman Temizkan) and my brother (Erhan Temizkan) for their continuous and unconditional supports through my doctorate program as well as through my life. Their continuous and unconditional supports provided me enormous strength and allowed me to overcome all obstacles during my doctorate program. I would also like to express my deepest gratitude to my grandmother (Gul Ayse Tufan). I wish she could see my graduation. Successful completion of the dissertation would not have been possible without the love and patience of my family. I love all of you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1. Background

Software development is an organizational process that software vendors use to develop and maintain software (Hoffer et al. 2008). Software development primarily depends on the economic behavior and organizational structure of software vendors. In general, software vendors focus on effective management of software development. As shown in Figure 1, one dimension of software development is an external environment in which external entities such as government and interest groups (i.e., customers) impose costs on a firm (Baron 2001) since software vendors internalize the external effects of their decisions (Baron 2001). Other software vendors also affect the software development from the competition perspective (Arora et al. 2010b, Cavusoglu et al. 2007). Therefore, the economic activity of software vendors is affected by government, customers, and other software vendors. Moreover, the changing needs of customers also affect software development since a software product is modified after delivery to correct faults, to improve performance or other attributes, and to enhance the product by adapting it to a modified environment (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). As shown in Figure 1, another dimension of software development is an internal environment in which organizational structure of software vendors affects software development.

Other internal factors specific to software vendors, software products, patches, and vulnerabilities also affect software development based on the cost structure of software vendors. In this dissertation, we analyze the effects of external and internal factors on software development.

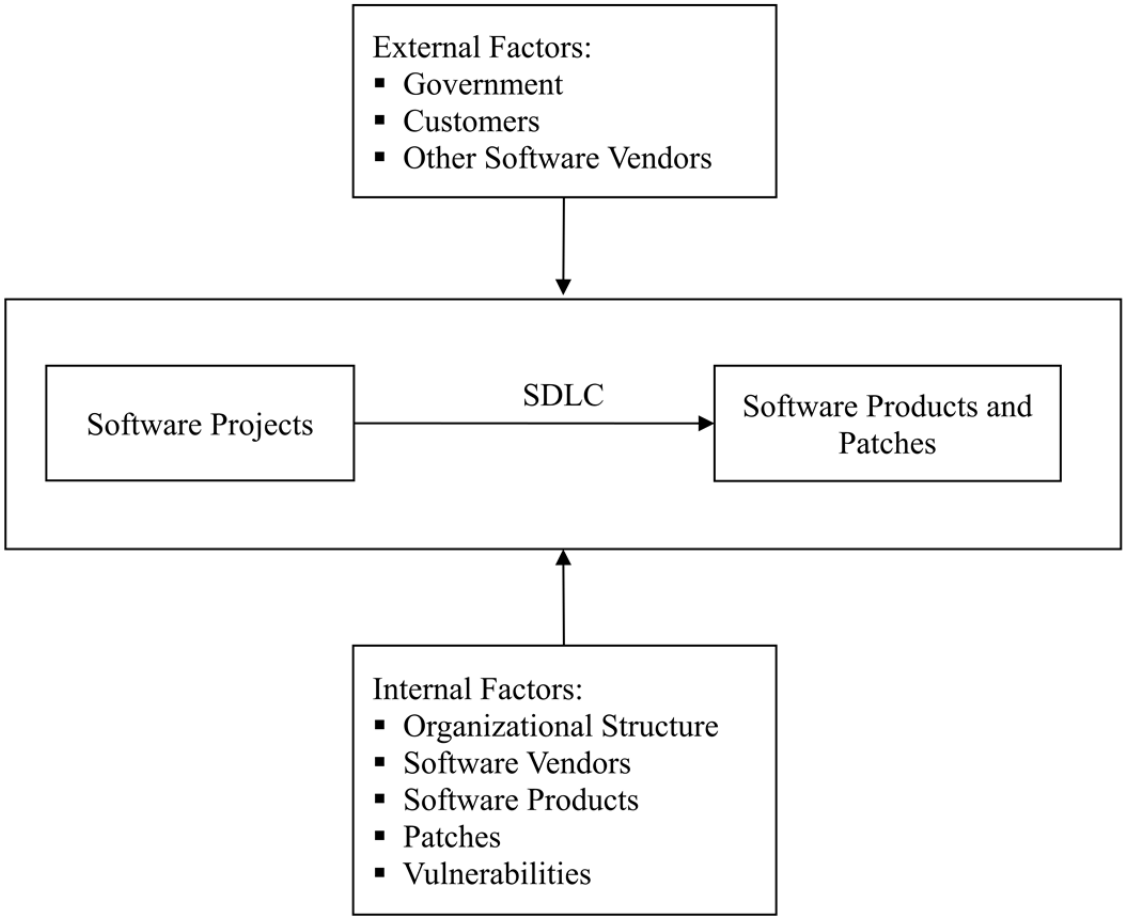

FIGURE 1: Research Overview

Software development can be modeled with a systems development life cycle (SDLC) which provides sequential activities such as planning, analysis, design, implementation, and maintenance for software developers to follow. Software maintenance is the last phase of a systems development life cycle. Software maintenance

is defined as the modification of a software product after delivery to correct faults and to enhance the product by adding new features based on user requirements or by adapting it to a modified environment (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). Although maintenance is the last phase, actually it covers all previous phases to be performed (Hoffer et al. 2008), and thereby it resembles a systems development life cycle itself (Hoffer et al. 2008). The total cost of system maintenance is estimated to comprise at least 50% of total software life cycle costs (Van Vliet 2000, Kemerer and Slaughter 1999, Kemerer 1995). Thus, software maintenance is one of the major phases of a software development. In software maintenance, there are two important types of OSS project activities: software improvement activities by correcting faults (i.e., vulnerabilities) and software enhancement activities by adding new features.

In an organizational context, exploitation and exploration have been identified as two types of activities for the development and use of knowledge in organizations (March 1991). Prior research indicates that different types of tasks require different communication patterns and different amount of communication based on characteristics and nature of a task (Katz and Tushman 1979). A task can differ along several dimensions including time span, specific vs. general problem orientation, and the generation of new knowledge vs. using existing knowledge (Katz and Tushman 1979). March (1991) has suggested that exploitation and exploration represent fundamentally incompatible and inconsistent activities. For example, exploitation represents activities that improve existing organizational competencies and build on the existing technological trajectory. Therefore, exploitation broadens existing knowledge and skills, improves established designs, and expands existing products and services. In contrast, exploration

represents activities that changes the organizational competencies and build on a different technological trajectory. Therefore, exploration requires new knowledge, offers new designs, and creates new products and services. In addition, exploitation is related to efficiency, centralization, and tight cultures while exploration is associated with flexibility, decentralization, and loose cultures (Benner and Tushman 2003). Therefore, exploitation and exploration require different organizational structures (Benner and Tushman 2003, Levinthal and March 1993). We propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration) based on organizational theory (March 1991). In the context of OSS development, developing a patch would be an example of an exploitation activity. Requesting a new software feature would be an example of an exploration activity. To the best of our knowledge, this is the first research to study OSS development at the activity level.

While exploitation and exploration represent fundamentally incompatible and inconsistent activities (March 1991), recent research on organizational literature has stressed the importance of a balance between exploitation and exploration for organizational survival (Benner and Tushman 2003, Tushman and O'Reilly 1996). Structural differentiation is a proposed mechanism for organizations to build an ambidextrous organization (Benner and Tushman 2003, Tushman and O'Reilly 1996). Structural differentiation refers to the subdivision of organizational tasks into distinct organizational units that develop appropriate contexts for exploitation and exploration activities. However, the coordination and integration of exploitative and exploratory activities is a necessary step in achieving ambidexterity (Jansen et al. 2009, Gilbert 2006,

Smith and Tushman 2005, Tushman and O'Reilly 1996). We identified a new category of developers (ambidextrous developers) in OSS projects who contribute to exploitative activities (patch development) and exploratory activities (feature request). We propose that ambidextrous developers are an integration mechanism between patch development and feature request activities. We develop a theoretical construct for project ambidexterity based on the concept of ambidextrous developers. We develop a theoretical construct of ambidexterity as a measure of the ability of OSS projects to pursue both exploitative and exploratory activities concurrently.

1.2. Research Objectives and Organization of the Dissertation

This dissertation examines exploitation and exploration in software development. It consists of three essays and is organized as follows.

In Chapter 2, we analyze software patch development (exploitation) in the context of software vulnerabilities which could be exploited by hackers. In particular, we analyze how the factors specific to vulnerabilities, patches, software, and software vendors affect the patch release behavior of software vendors. We develop a survival analysis model of the patch release behavior of software vendors based on the cost-based framework of software vendors. We test it using a data set compiled from the National Vulnerability Database (NVD), United States Computer Emergency Readiness Team (US-CERT), and vendor web sites.

In Chapter 3, we study exploitation and exploration in the content of OSS development. We introduce the use of organizational theory on exploration and exploitation together with social network analysis as a theoretical lens to study different types of sub-networks in OSS development. We empirically examine the differences

between exploitation (patch development) and exploration (feature request) networks of developers in OSS projects in terms of their social network structure. We identify a new category of developers (ambidextrous developers) in OSS projects who contribute to patch development as well as feature request activities. We use a data set collected from the SourceForge database. Our results indicate that a patch development network has greater internal cohesion and network centrality than a feature request network. In contrast, a feature request network has greater external connectivity than a patch development network.

In Chapter 4, we explore ambidexterity and ambidextrous developers in the context of OSS project performance. We introduce the use of organizational theory on ambidexterity together with social network analysis as a theoretical lens to study OSS project performance. Recent research on OSS development has studied the social network structure of software developers as determinant of project success. However, this stream of research has focused on the project level, and has not recognized the fact that software projects could consist of different types of activities, each of which could require different types of expertise and network structures. We develop a theoretical construct for ambidexterity based on the concept of ambidextrous developers. We empirically illustrate the effects of ambidexterity and network characteristics on OSS project performance. We also study the effects of social network properties of OSS developers on OSS project performance. We use a data set collected from the SourceForge database. Our results indicate that a moderate level of ambidexterity, external cohesion, and technological diversity are desirable for project success. Project success is also positively related to internal cohesion and network centrality. We illustrate the roles of ambidextrous

developers and their differences compared to other developers. In summary, this dissertation explores software development using an under-researched theoretical lens.

CHAPTER 2: PATCH RELEASE BEHAVIORS OF SOFTWARE VENDORS IN RESPONSE TO VULNERABILITIES: AN EMPIRICAL ANALYSIS

2.1. Introduction

Software vulnerabilities of information systems have become a significant concern for organizations, since these vulnerabilities result in security attacks such as virus, theft of information, and denial of service (DOS) leading to significant financial losses to users (Gordon et al. 2006). The existence of software vulnerabilities is an important reason for security attacks (Ransbotham and Mitra 2009, Arora et al. 2006b). The number of vulnerabilities reported by Computer Emergency Response Team / Coordination Center (CERT/CC) increased dramatically from 171 to 7326 between 1995 and 2007. The public disclosure of such software vulnerabilities increases the risks posed by security attacks (Arora et al. 2006b). Disclosed software vulnerabilities could expose the systems of unprotected users to security attacks and could be exploited by attackers to compromise organizational information systems through these unprotected systems. At the same time, disclosure of vulnerabilities is used by social organizations such as CERT/CC to hasten the release of software patches by vendors.

One of the important characteristics of a disclosed vulnerability is its severity and prior research has found that the severity of the vulnerability affects the patch release behavior of software vendors (Arora et al. 2010a, Kannan et al. 2007, Png et al. 2008). However, these studies treat the severity as an aggregated measure. We expect that vulnerabilities with same severity measures can, however, pose different risks for data assets of an organization (Houmb et al. 2008, Mell and Scarfone 2007), depending on the type of impact. For example, vulnerabilities that result in unauthorized information disclosure may need to be responded to differently compared to vulnerabilities that lead to unavailability of data. Hence, the patch release behavior of vendors is likely to be different according to the expected impact of different aspects of the severity of the vulnerability. We use the data on vulnerabilities published by United States Computer Emergency Readiness Team (US-CERT) and cross-reference the data with National Vulnerability Database (NVD) to capture detailed information about vulnerabilities and patches released by vendors. We then test a survival model of the vulnerability with this data, thereby extending prior research on patch release behavior and helping to understand vendor's patch release response in a more comprehensive way.

Our analysis shows that while confidentiality impact and integrity impact are the most important vulnerability characteristics that affect the patch release behavior of software vendors, these two impacts are highly correlated with each other. Therefore, vendors tend to show almost identical behavior for the same level of impact of confidentiality or integrity when releasing their patches. In general, higher confidentiality impact or integrity impact makes vendors release patches faster while higher availability impact results in slower release of patches. However, further analysis highlights some

interesting differences between open source software (OSS) and proprietary software vendors in terms of their response to availability impact of vulnerabilities. We also find that the patch release behavior of software vendors is different if the patch is an update rather than a new release. Introducing the type of patch in our paper is a response to a call in prior research (Arora et al. 2010a) for additional variables to describe patch release behavior and for using additional data sources. Prior research has recognized differences between OSS and proprietary vendors in debugging software (Raymod 1999). We find interesting differences between the behavior of open source software and proprietary software vendors in releasing patches for vulnerabilities, depending on the type of patch (upgrade vs. new release). Prior research also indicated that patch quality depends on patch development time (Arora et al. 2008) and cost of developing software (Arora et al. 2008, Slaughter et al. 1998). We find that lower quality patches are released faster because of the trade-off between the quality of a patch and its cost to the vendors.

2.2. Literature Review

While there are multiple streams of research that help understand vendors' patch release behavior, the common theoretical underpinning of most of these studies is that the release time and quality of security patches are largely determined by the economic behavior of vendors each of whom have a specific cost structure. Arora et al. (2008) introduce a model of vulnerability disclosure and software vendors' patch release behavior involving a social coordinator (e.g., CERT/CC) as an external entity who sets the disclosure time and a vendor who decides on the patch release time. The model focuses on a vendor's cost and social cost as the two types of costs. Social cost is total customer loss resulting from the exploitation of unpatched vulnerabilities by security

attackers. The vendor's cost consists of two terms: the cost of patch development and the portion of social cost internalized by the vendor. The cost of developing a patch is determined by a vendor based on patch development time and patch quality because accelerating patch development and increasing patch quality draw upon more resources (Arora et al. 2008). Hence, vendors are likely to incur higher patch development costs with accelerating patch development or increasing patch quality because of allocation of more resources (Arora et al. 2008, Slaughter et al. 1998) or quality improvement processes (Slaughter et al. 1998). In contrast, vendors internalize the portion of social cost in the form of either a loss in reputation, a loss in future sales, or as customer support costs (Arora et al. 2008). In this model, social coordinators set the disclosure policy and influence vendor's patch release behavior by determining optimal vulnerability disclosure time in order to minimize social cost. In response, a vendor decides on the patch release time in order to minimize its expected cost. Therefore, vendor's patch release decision depends on the trade-off between these costs. Arora et al. (2008) indicated that a vendor is more responsive if a greater portion of the customer loss is internalized by the vendor.

Arora et al. (2010a) also used the same model and indicated the positive impact of vulnerability disclosure on the patch release time, i.e., vulnerabilities that were publicly disclosed were patched faster. Cavusoglu et al. (2007) use an analytical model to study vulnerability disclosure mechanisms (i.e., instant disclosure, no disclosure, and optimal disclosure) on patch release decisions. They show that even though each disclosure mechanism ensures the release of patch by vendors, early disclosure does not always lead to faster patch release by vendors because vendors may incur higher development costs if they release patches faster. Hence, they may trade off development costs for higher

internalized customer loss. These studies provide analytical evidence that vulnerability disclosure affects the vendors' patch release behavior and patch release time. However, the importance of these studies is to develop the model explaining software vendors' patch release behavior in terms of their cost structure.

Prior research on corporate social responsibility identified government and interest groups, such as customers, as external entities who impose costs on a firm (Baron 2001). Market and legislation mechanisms lead firms to internalize the external effects of their decisions (Baron 2001). Therefore, economic activity of companies is affected by government and customers through two mechanisms. One is through market, in which customers influence economic activity of firms through either a loss in reputation, a loss in future sales, or as customer support costs. The other is through government legislations, in which government influences economic activity of firms through legislations and associated penalties. From corporate governance perspective, the primary objective of managers is to maximize shareholder value (Brigham and Ehrhardt 2008, Baron 2001). However, Wood (1991) argued that business and society are interwoven rather than being distinct entities and society has certain expectations for appropriate business behavior and outcomes. Therefore, the social responsibility of business encompasses the economic, legal, and ethical expectations (Carroll 1979). It is possible that vendor patch release behavior is affected by such legislative mechanisms, through their impacts on vendor's cost components, such as internalization of social cost.

The impact of the severity of vulnerability on the patch release behavior of software vendors has also been studied in prior research (Arora et al. 2010a). In Kannan et al. (2007), reactions of vendors differ for different types of security attacks because an

attack may target a specific vulnerability and with a different purpose. The impact of the severity of vulnerabilities on the number of attacks has been studied by Png et al. (2008), who conclude that software vendors tend to spend more effort developing patches for vulnerabilities having a higher severity impact than for vulnerabilities having a lower severity impact. The confidentiality-integrity-availability (CIA) framework assesses security attacks based on the risk posed by the attacks to confidentiality, integrity, and availability of data assets. For example, theft of information is categorized as a confidentiality attack since it poses the risk of unauthorized disclosure of information. Virus attacks are categorized as integrity attacks since they pose the risk of unauthorized modification of data assets. Denial of service (DOS) attack is categorized as an availability attack since it makes systems unavailable. The severity of vulnerability studied by Arora et al. (2010a) is an aggregate variable and does not capture the confidentiality, integrity, and availability dimensions of vulnerabilities on the patch release behavior of software vendors. It may be possible that components of vendor's cost, such as internalization of social costs, may be different for different dimensions of vulnerability impact.

Software complexity is a major factor influencing the software's maintenance efforts (Banker and Slaughter 2000, Banker et al. 1998, Kemerer 1995, Roberts et al. 2004) and maintenance costs (Banker and Slaughter 2000, Banker et al. 1998). Software maintenance is the modification of a software product after delivery in order to correct faults, to improve performance or other attributes, and to enhance the product by adapting it to a modified environment (IEEE 1983). Since releasing a patch for vulnerability is part of the software maintenance effort, software complexity is an important variable when

studying vendor patch release behavior. Software complexity generally refers to the characteristics of the data structures and procedures within software products that make it difficult to understand and change and software complexity has been strongly linked with software maintenance efforts (Banker et al. 1998). These results are supported by the study of Banker and Slaughter (2000) in which software development is regarded as software enhancement activity. Software complexity has also been studied as part of the complexity of information systems development projects (Xia and Lee 2005). In this study, the complexity has been analyzed based on the structural aspects of projects capturing the impact of variety and interdependency of project elements on complexity. As the number of project elements and their interdependencies increases, it becomes more difficult to control the project.

Prior research has shown the impact of patch quality on patch development time (Arora et al. 2008) and cost of developing software (Arora et al. 2008, Slaughter et al. 1998). This is based on the idea that software vendors choose the patch quality in order to minimize their expected costs, which consist of the patch development cost and the portion of total customer loss resulting from vulnerabilities exploited by attackers (Arora et al. 2008). Arora et al. (2006a) also showed that a software vendor has incentives to release a buggier product early and fix it later.

Raymond (1999) indicated that the nature of software debugging, an important task in patch development, is different for proprietary and OSS vendors. OSS development depends on contributions and collaboration of volunteer software developers (Liu and Iyer 2007, Feller and Fitzgerald 2002). Prior studies show that collaboration among product design teams is associated with a reduction in product

development cycle time (Espinosa et al. 2007, Banker et al. 2006) and a reduction in software development cost (Jaisingh et al. 2008). Sen (2007) concludes that OSS vendors benefit from the collective network of software developers in OSS development process resulting in quicker releases of software than proprietary vendors.

Software can be categorized as application software and system software based on what specific tasks the software is designed to accomplish (O'Brien and Marakas 2008) and the category can affect the patch development efforts. Meil and Scarfone (2007) indicated that the level of access to the operating system (i.e., root or user level of access) provides different level of control over the operating system (OS) and show that vulnerabilities at an operating system level are typically more severe than that on an application level. Arora et al. (2010b) examined empirically the impact of competition among multiple vendors on the patch release time. In particular, they looked at disclosure threat effect, which is the effect on patch release time of the possibility that another vendor releases a patch earlier and implicitly discloses the vulnerability. The disclosure threat significantly reduces the patch release time (Arora et al. 2010b), which is consistent with the results by Cavusoglu et al. (2007) who analyzed the multiple vendor case and concluded that the patch release of one vendor affects the patch release decisions of other vendors. Jaisingh et al. (2008) show that the software development process of one vendor is affected by competition from other software developers.

The literature reviewed above suggests that vulnerability characteristics such as confidentiality, integrity, and availability could impact vendors' patch release behavior. The patch release may also be affected by the software complexity, the software quality, the types of software development processes, and the differences between software

categories. Prior research also indicates that the presence of multiple vendors significantly affects the vendor's patch release decision. Although prior research suggests the impact of these factors on the vendor's patch release behavior, our study is the first empirical attempt to analyze the impact of the individual vulnerability characteristics (i.e., confidentiality, integrity and availability) as well as the impact of patch quality. The extant software vulnerability patch literature also does not consider the impact of software complexity and its interaction with other factors on vendor patch release behavior. On the premise that the time and cost to develop a patch are dependent on the software complexity, we collect data regarding patch types for vulnerabilities and examine how these patch types affect vendor patch release behavior. The following section discusses the vulnerability scoring system that measures the severity of individual vulnerability characteristics.

2.3. Common Vulnerability Scoring System (CVSS)

Information security is defined as the combination of the attributes of confidentiality, integrity, and availability (Avizienis et al. 2004). The Control Objectives for Information and Related Technology (COBIT) standards is a widely accepted set of guidance for IT governance and also identities confidentiality, integrity, and availability of data assets as important for IT governance. According to the Forum of Incident Response and Security Teams (FIRST), confidentiality refers to limiting information access and disclosure to only authorized users, as well as preventing access by or disclosure to unauthorized users (Mell et al. 2007). Integrity refers to the trustworthiness and guaranteed veracity of information while availability refers to the accessibility of data resources (Mell et al. 2007).

While confidentiality, integrity, and availability are referred to as the primary attributes of information security, secondary attributes such as accountability (Avizienis et al. 2004, Pfleeger and Pfleeger 2003, Biskup 2009), assurance (Avizienis et al. 2004) and authenticity (Avizienis et al. 2004) refine or specialize the primary attributes (Avizienis et al. 2004). For example, accountability is defined as availability and integrity of the identity of the person who performed an operation (Avizienis et al. 2004). The use of information should be transparent so that it is possible to determine whether a particular use is appropriate under a given set of rules and that the system enables individuals and institutions to be held accountable for misuse (Weitzner et al. 2008). Assurance is derived from integrity and is defined as the prevention of the unauthorized modification or deletion of information, while authenticity is defined as the integrity of a message content and origin (Avizienis et al. 2004). Although secondary attributes carry more specific information about the vulnerability than primary attributes, the impact of vulnerability is best represented by the impact scores of primary attributes. Moreover, the NVD database does not disclose any of the secondary attributes. Hence, the scope of this study is limited to primary attributes.

The Common Vulnerability Scoring System (CVSS)[1] provides standard measures for the impacts of vulnerability based on its severity. The characteristics of vulnerabilities are assumed constant over time and across user environments, and categorized into exploitability and impact subgroups (Mell et al. 2007). The CVSS gives an aggregated severity score (on a scale of 0 to 10) for each vulnerability. This aggregated severity score is calculated by combining exploitability and impact subscores. Although

[1] The Common Vulnerability Scoring System (CVSS) is a standard for scoring the impact of vulnerabilities. The CVSS was originally introduced by the National Infrastructure Advisory Council (NIAC) and is currently managed by the Forum of Incident Response and Security Teams (FIRST).

exploitability subscores capture how a vulnerability is accessed, the focus of this study is on impact subscores since the three impact metrics (i.e. confidentiality, integrity and availability) measure the direct impact of an exploited vulnerability on data assets (Chandramouli et al. 2006). Possible values for each impact subscore are "None, Partial and Complete"[2]. While "None" indicates a total absence of the impact on data assets, "Complete" means the maximum impact on data assets. "Partial" refers to a partial impact on data assets. A higher value for each impact metric indicates higher severity for that metric.

Although the definitions of the vulnerability characteristics do not overlap, they may be correlated across vulnerabilities. For example, if security attackers gain a right to modify data without authorization as a result of the exploitation of vulnerability, they may also acquire the content of data. Therefore, violations of integrity may also lead to the violation of confidentiality. The modification or destruction of data makes data unavailable or inaccessible and thus violations of integrity may also result in the violation of availability. We will analyze these possible correlations between any pairs of vulnerability characteristics in the Research Methodology section.

2.4. Theoretical Background and Hypotheses

The primary purpose of this study is to analyze how the factors specific to vulnerabilities, patches, software, and software vendors affect the patch release behavior of software vendors based on extending the cost-based framework of Arora et al. (2008). We develop a model of vendor patch release behavior and posit that software vendors' patch release behaviors are affected by three impact dimensions of vulnerabilities, the

[2] Corresponding coefficients of possible values for impact metrics are available at http://www.first.org/cvss/cvss-guide.pdf

types of patches, the types of software vendors, the software categories, and the patch quality. We also posit that software vendors' patch release behaviors are affected by the presence of multiple vendors whose products have the same vulnerability. We consider government's role in affecting components of vendor's cost, such as internalization of social costs, and propose that government actions cause software vendors to internalize a higher amount of social cost via government enforcements and legislative penalties on customers.

2.4.1. Vulnerability Characteristics: Confidentiality, Integrity, and Availability

Arora et al. (2010a) show that vendors release patches faster for more severe vulnerabilities. However, the vulnerability severity used in their study is an aggregated score computed from individual vulnerability characteristics (Mell et al. 2007) and the impacts of individual vulnerability characteristics on vendors' patch release behavior has not been captured. Theoretically, different combinations of vulnerability characteristics can produce the same aggregated severity score but pose different risks to users since the actual risk depends on the impact of each vulnerability characteristic on data assets (Houmb et al. 2008, Mell and Scarfone 2007). For example, a successfully exploited vulnerability can cause a complete loss of confidentiality, and a partial loss of integrity, but no loss of availability. In order to capture the impact of vulnerability characteristics in more depth, the CIA framework, which underlies the confidentiality, integrity, and availability impacts of vulnerability, is used in our study. The CIA framework is used in prior research to classify security attack types (Kannan et al. 2007). Different types of security attacks (e.g., virus attacks, theft of information, and DOS attacks) target different aspects of data assets (i.e., confidentiality, integrity, and availability). Security attacks

that result in unauthorized information disclosure, such as theft of credit card numbers or other customer information, are categorized as confidentiality attacks (e.g., theft of information). Security attacks that result in authorized modification or total destruction of data assets are categorized as integrity attacks (e.g., virus attacks). Security attacks that make data resources or systems inaccessible are classified as availability attacks (e.g., DOS attacks).

Confidentiality, integrity, and availability impacts are considered as important vulnerability characteristics by Sarbanes-Oxley (SOX) legislation and companies have to establish the internal control over these dimensions. This view is supported by the Federal Information Security Management Act (FISMA) that requires federal agencies to protect information and information systems from unauthorized access, disclosure, use, modification, or destruction of data to strengthen information security and provide integrity, confidentiality and availability. Software vendors incur a higher cost from customer loss resulting from vulnerabilities exploited by security attackers if the vulnerabilities are critical (Arora et al. 2008). This view is supported by Png et al. (2008) who report that software vendors expend greater effort to develop patches for vulnerabilities having a higher severity impact. Software vendors are expected to react faster in developing patches for vulnerabilities having a higher impact on confidentiality, integrity and availability of data assets because they internalize a greater amount of customer loss resulting from exploitation of vulnerabilities having a higher impact on confidentiality, integrity and availability. Hence, this leads us to the following hypotheses:

H1: Higher confidentiality impact of vulnerabilities is associated with faster patch release by vendors.

H2: Higher integrity impact of vulnerabilities is associated with faster patch release by vendors.

H3: Higher availability impact of vulnerabilities is associated with faster patch release by vendors.

2.4.2. Vulnerability Characteristics: Confidentiality vs. Integrity, and Availability

Confidentiality attacks such as theft of information result in unauthorized information disclosure. However, violations of confidentiality in the form of information disclosure cannot be recoverable, while violations of integrity or availability can be recovered by other means, such as having backup systems. Violations of confidentiality may also be harder to detect than exploitations of integrity or availability. The risk of unauthorized information disclosure for financial institutions has been demonstrated by Johnson (2008). This study also provides companies with strategies on how to control information disclosure. The confidentiality of information is associated with privacy and regulated by various forms of legislation (O'Brien and Marakas 2008). The Electronic Communication Privacy Act and the Computer Fraud and Abuse Act prohibit intercepting data communication and stealing data (O'Brien and Marakas 2008). The Health Insurance Portability and Accountability Act (HIPAA), which includes the privacy rules and the security rules, addresses issues related to individual health insurance. The HIPAA creates the safeguards against the unauthorized use, disclosure, and distribution of an individual's health care information held by health care service providers without the specific consent or authorization. The Gramm-Leach-Bliley Act

(GLBA) is another legislation that includes provisions to govern the disclosure, and protection of consumers' nonpublic personal information held by companies. According to the GLBA, companies are required to develop information security plans to protect consumers from unauthorized disclosure of their nonpublic personal information. Given the significant legislative pressure regarding privacy and confidentiality, we expect that confidentiality impact is considered as more critical to respond to than availability and integrity impact types. This is because software vendors are likely to internalize a greater amount of customer loss for the violation of confidentiality than for the violation of integrity or availability. This leads us to the following hypothesis:

H4: Confidentiality score has a greater positive impact on vendors' patch release time than integrity score or availability score.

2.4.3. Vulnerability Characteristics: Integrity vs. Availability

Integrity attacks such as virus attacks result in unauthorized modification or total destruction of data assets. Modified or destructed data can be backed up and restored to a clean state relatively easily when compared with violations of confidentiality. However, in case of random data modifications by security attackers it is difficult to notice the violation and to determine what has been changed. Violations of integrity may also result in the violation of availability since modified or destructed data is not available. Availability attacks such as DOS attacks make a system unavailable but it can be recovered relatively easily since violations of availability are easier to notice than those of integrity. For example, the HIPAA regulates that the access to and modification of health care information is limited to authorized persons. This view is supported by Sarbanes-Oxley whose objective is to improve accountability of information (Anand

2008). We conclude that integrity impacts of vulnerability are more severe than availability impacts because software vendors internalize a greater amount of customer loss for the violation of integrity than for the violation of availability. This leads us to the following hypothesis:

H5: Integrity score has a greater positive impact on vendors' patch release time than availability score.

2.4.4. Patch Types

Software complexity generally refers to the characteristics of the data structures and procedures within software products that make it difficult to understand and change the software (Banker et al. 1998). A significant portion of the software developer's time is required to understand the functionality of the software to be changed (Banker et al. 1998). Software complexity also affects the effectiveness of development teams (Roberts et al. 2004). The more complex the software product, the more effort required to mitigate the negative impact of software complexity on software development process. The structure of software development projects such as variety and interdependency of project elements also determine the software complexity (Xia and Lee 2005). As the number of project elements increases, software development becomes more difficult to control. Software complexity also determines the maintenance cost of software since high level of software complexity interferes with the process of comprehending the application and makes it difficult for developers to efficiently and correctly modify the application (Banker and Slaughter 2000). We measure software complexity in terms of patch types: an update or a new release. A new release patch is required to modify an entire product and the data structures and procedures are more complex for a new release. Typically a

new release is larger in size than an update patch. We assume that a new release type of software requires more development effort than an update type. Therefore, software vendors incur higher development costs for a new release type of patch than an update type of patch. This leads us to the following hypothesis:

H6: An update type of patch is released faster than a new release type of patch by vendors.

2.4.5. Software Vendor Types

The nature of the software debugging task for proprietary and OSS vendors leads to two fundamentally different software development styles: the cathedral model for proprietary vendors and the bazaar model for OSS vendors (Raymond 1999). Software development involves knowledge work and its most important resources are the specialized skills and expertise that a developer brings to the project development (Espinosa et al. 2007, Roberts et al. 2004, Faraj and Sproull 2000). Proprietary vendors use a more closed environment and the development process is characterized by a relatively strong control of design and implementation. In contrast, OSS vendors depend mainly on voluntary contributions of software developers and, hence, a patch for OSS product is developed in a collective manner beyond the boundaries of a single organization. The network of developers becomes more important for OSS projects and offers various benefits. First, collaboration among software developers can facilitate access to and sharing of resources, allowing developers to combine their knowledge, skills, and expertise. Second, new insights, ideas or ways to solve problems are conceived by any one and accessed by others. This leads to decrease in time required to find a solution for fixing vulnerabilities. Third, the volunteer group of software developers does

not depend on the schedule of a company to release a patch for vulnerabilities in the OSS. In contrast, the release calendar of a patch by a proprietary vendor may be subject to the vendor's marketing and strategic needs. Fourth, the cost of software development is mainly resulting from the investment on resources such as salaries paid to developers (Jaisingh et al. 2008). However, OSS vendors benefit from voluntary contributions of developers (Jaisingh et al. 2008). We expect that OSS vendors release patches faster than proprietary vendors because the voluntary contributions of software developers to OSS development reduces the development cost for OSS vendors. This view is consistent with the results of Banker et al. (2006) which show that collaboration among design teams in OSS projects is associated with a reduction in OSS development time. Espinosa et al. (2007) also concluded that collaboration leads to benefits such as shorter development time. This leads us to the following hypothesis:

H7: Open source vendors release patches faster than proprietary vendors.

2.4.6. Software Types

Software is primarily categorized as application software and system software based on what specific tasks the software is designed to accomplish (O'Brien and Marakas 2008). Application software, such as word processing, spreadsheets, graphics programs or electronic mail applications, are stand-alone function-specific programs that provide individual end users with common information processing tasks. In contrast, system software manages and provides a vital software interface among computer networks, hardware, and application software of end users. Operating systems, application servers or network management programs are examples of system software. Web-based software also provides a software interface over computer networks among

application software. For example, web browsers run other application software such as web-based applications (e.g., Google Document) or other programming codes (e.g., ActiveX controls). Therefore, web-based software such as web browsers is categorized as system software in our study. System software is more critical and important than application software, since it provides a software interface among computer networks, hardware, and application software of end users. Exploitation of vulnerabilities belonging to system software affects the entire system and gives security attackers an OS level of access and control to data resources. In contrast, exploitation of vulnerabilities belonging to application software affects an application itself and gives security attackers a limited level of access and control to data resources. We argue that vulnerabilities belonging to system software are more serious and patched faster than those belonging to application software because software vendors internalize more customer loss for the exploitation of vulnerabilities affecting system software than vulnerabilities affecting application software. This leads us to the following hypothesis:

H8: Vulnerabilities affecting system software are patched faster by vendors than those affecting application software.

2.4.7. Patch Quality

Software vendors choose a patch quality that minimizes the total of the patch development cost and the portion of total customer loss (Arora et al. 2008). Vendors incur higher patch development costs with the higher patch quality because of allocation of more resources (Arora et al. 2008, Slaughter et al. 1998) or quality improvement processes such as design reviews and code inspections (Slaughter et al. 1998). However, there are early mover advantages resulting in incentives to release products earlier despite

the product not being ready for release (Arora et al. 2010b). Hence, software vendors have incentives to release an incomplete patch with partial fix early and fix it later (Arora et al. 2006a) in order to offset the potential customer loss of not releasing any patch (Arora et al. 2008). Software vendors may sacrifice quality by eliminating quality improvement processes to achieve other objectives such as shorter development cycle time and reduced development cost (Slaughter et al. 1998). We argue that lower quality patches are released faster than higher quality patches because software vendors incur less development cost for low quality patches and further reduce the potential customer loss of not releasing any patch at least by releasing low quality patches early. This leads us to the following hypothesis:

H9: Lower quality patches are released faster than higher quality patches.

2.4.8. Presence of Multiple Vendors

The impact of the presence of multiple vendors for a given vulnerability on the patch release time has been analyzed from the competition perspective (Arora et al. 2010b, Cavusoglu et al. 2007). Arora et al. (2006a) study firms' incentives to release their software earlier to gain more market share and avoid the threat of competition in the market, since the software market offers significant early mover advantages. In the presence of multiple vendors, there is also a possibility that the release of a patch by another vendor discloses the vulnerability, putting pressure on other vendors who have not released a patch yet. We expect that the threat of disclosure and customers' penalty on late patching could decrease the patch release time by vendors when multiple vendors are affected by the same vulnerability. This leads us to the following hypothesis:

H10: Vulnerabilities affecting multiple vendors are patched faster than vulnerabilities affecting a single vendor.

2.5. Variable Definitions and Operationalization

2.5.1. Computation of Dependent Variable

Patch Release Time: Patch release time is the dependent variable in our model. The patch release time is measured as the number of days between the vendor notification date and the patch release date. In our model, we have set June 20, 2009 as a cutoff date for our study. If patch release date is earlier than our cutoff date, the patch release time is the number of days between the vendor notification date and the patch release date. However, some vendors had not released their patches by the cutoff date. These observations were removed from the data set because patch release time and other patch related information cannot be obtained. The vendor notification date is the date when a vendor first knows the existence of vulnerability in its product(s). Vendors usually are notified by US-CERT and for these vulnerabilities we have noted the vendor notification date from the US-CERT website. For some vulnerabilities, vendors have been notified by other security sources such as SecurityFocus, iDefense, and TippingPoint. US-CERT referred to these sources for the vendor notification date. For these vulnerabilities, we have noted the vendor notification date from the links provided by US-CERT. For some vulnerabilities, vendors provide a date when they first became aware of the existence of vulnerability in their products or when they were notified about the existence of vulnerability in their products. We select the earliest notification date from among those identified from US-CERT website, other security websites, and the software vendor's website. For some vulnerabilities, we use the vendor notification date as the date when

vulnerability information is first made public. Generally, the public date is the earliest of the date when the vulnerability information is first published, the date when an exploit was first discovered, or the date when the vendor first distributed a patch publicly. We have noted the vulnerability public date directly from the US-CERT website. The patch release date is the date when a vendor releases a patch for a vulnerability in its product. However, the same software vendor may release multiple patches for the same vulnerability. For these vulnerabilities, we selected the earliest date as the patch release date. US-CERT and NVD websites do not directly provide patch release dates. Instead they give the links for available solutions and refer us to patch release reports, security bulletins or security advisories published by software vendors. Patch release dates have been collected from these sources or from the digital signatures of patches.

2.5.2. Independent Variables

Vulnerability Characteristics (Confidentiality, Integrity, and Availability): Confidentiality, integrity, and availability impacts are independent variables in our model, and take a value of 0 for “None”, a value of 1 for “Partial”, and a value of 2 for “Complete”. Data regarding vulnerability characteristics are collected from the NVD website.

Patch Type (PType): Patch type data has been collected from the US-CERT website or software vendors’ website. We identified different types of patches such as configurations, scripts, updates, and new releases. We grouped configurations, scripts, and updates under the update type of patches. Patch type takes a value of 1 for new release patches and 0 for update type of patches.

Software Vendor Type (VType): Software vendor type has been identified for each vendor by referencing websites such as the Open Source Vulnerability Database (OSVDB). Software vendor type takes a value of 1 for open source vendors and 0 for proprietary vendors.

Software Type (SWType): Software type has been identified from the list of affected products provided by the US-CERT website and software vendors' website. Software type takes a value of 1 for system software and 0 for application software.

Patch Quality (MPatches): Beattie et al. (2002) measured the patch quality with the number of patch released for the same vulnerability. Khoshgoftaar et al. (2000) developed a software quality model and measured software quality with multiple releases of software based on the idea that software is improved from faults in the last release. Therefore, multiple patches imply lower quality in the first released patch. We measure patch quality with the number of patch released for the same vulnerability. We identified all patch releases along with patch release dates for the same vulnerability from patch release reports, security bulletins, security advisories published by software vendors, and the digital signatures of patches. It takes a value 1 if multiple patches are released for the same vulnerability and 0 otherwise.

Presence of Multiple Vendors (MVendor): The data on multiple vendors has been collected from the US-CERT and NVD websites. For each vulnerability, US-CERT and NVD lists all affected vendors. However, there are cases where US-CERT lists only one vendors but NVD lists multiple vendors, or vice versa. We treat both these cases as multiple vendor case. MVendor variable takes a value 1 if vulnerabilities affect multiple vendors and 0 otherwise.

2.5.3. Control Variable

Disclosure: Prior research has found that disclosure affects the patch release behavior of software vendors. Thus, disclosure is a control variable in our model and has been constructed as a time-dependent covariate, consistent with prior research (Arora et al. 2010a). If vulnerability information is disclosed before patch release date, disclosure takes value 1, which means vulnerability has been made public before the patch release, otherwise it takes a value of zero.

The description and operationalization of variables in our research are summarized in Table 1. The next section introduces the basis of our model and research methods that we adopted.

TABLE 1: Variables and Descriptions/Measures

Variables	Descriptions/Measures
Patch Release Time	Time taken in days by vendors to release a patch
Disclosure	Whether vulnerability information is disclosed before patch release date; 1 if vulnerability information is disclosed before patch release date, 0 otherwise
MPatches	Whether multiple patches are released for the same vulnerability; 1 if multiple patches are released for the same vulnerability, 0 otherwise
Confidentiality Impact	The impact of unauthorized information disclosure on data assets; 0 if none, 1 if partial, 2 if complete
Integrity Impact	The impact of unauthorized modification or total destruction of data on data assets; 0 if none, 1 if partial, 2 if complete
Availability Impact	The impact of unavailability of data resources on data assets; 0 if none, 1 if partial, 2 if complete
PType	The type of a patch released by a vendor; 0 if an update, 1 if a new release
VType	The type of a software vendor; 0 if a proprietary vendor, 1 if OSS vendor
SWType	The type of software affected by a vulnerability; 0 if application software, 1 if system software
MVendor	Whether there are multiple vendors affected by the same vulnerability; 1 if multiple vendors are affected by the same vulnerability, 0 otherwise

2.6. Research Methodology

The purpose of this study is to understand how factors specific to vulnerability characteristics (i.e., confidentiality, integrity and availability impacts), patches, software vendors and software affect the patch release behavior of software vendors, in particular the time to release a patch. This study also examines the impact of the presence of multiple vendors exposed to the same vulnerability on the patch release behavior of software vendors. In order to accomplish these research objectives and test the hypothesis developed in the previous sections, we use survival analysis based on the vulnerability life cycle model.

Arbaugh et al. (2000) developed a vulnerability life cycle model which captures all possible states that vulnerability can enter during its lifetime. The vulnerability life cycle model considers vulnerability as a birth and death process. In this study, the vendor notification event can be considered as the starting point of patch development process. However, a vendor has a choice whether to release a patch or not. If a vendor releases a patch for its product, the patch release date indicates the end of patch development process. The vulnerability survives as long as the vendor does not release a patch, and thus the time taken to release patch after the vendor knows about it is the survival time of vulnerability. In this study, we have considered the vulnerability life cycle as a survival model, in which the release of a patch by a software vendor is an event that ends the survival of a vulnerability.

The survival model is preferred to traditional multiple regression models in our study for several reasons. First, survival analysis does not impose any specific distributional assumptions. Second, survival analysis enables us to use time-dependent

covariates. In our model, vulnerability disclosure is a time dependent covariate, following Arora et al. (2010a). In contrast to survival analysis, multiple regression cannot handle time-dependent covariates.

We use Cox's proportional hazard model which is a semi-parametric survival model (Cox 1972). The patch release time is the dependent variable in our model. Peduzzi et al. (1995) suggested that at least 10 events are required for each independent variable in the model, and the regression coefficients become more biased with a decrease in the number of events for each variable. In this study, we have 10 variables including time-dependent covariates and the interactions terms. Therefore, the sample size of 722 is considered adequate when compared to the required sample size of 100. Cox's proportional hazard model is expressed as follows:

$$h(t, X(t)) = h_0(t) \exp[\sum_{i=1}^{k} \beta_i X_i + \sum_{j=1}^{n} \beta_j X_j(t)]$$

where $h_0\ (t)$ is the baseline hazard function at time t when all values of independent variables are equal to zero. X_i are independent variables for $i = 1...k$ and X_j are time-dependent covariates or the interaction terms for $j = 1...n$. β_i and β_j are model coefficients for $i = 1...k$ and $j = 1...n$ respectively.

2.7. Data

2.7.1. Data Sources and Collection

Vulnerability data analyzed in this study has been collected from two sources: US-CERT[3] and NVD[4], which are publicly available vulnerability databases. US-CERT

[3] The US-CERT is the operational part of the National Cyber Security Division (NCSD) of the U.S. Department of Homeland Security (DHS). The US-CERT provides the response and defense about vulnerabilities against cyber-attacks.

publishes vulnerability information in the form of "Vulnerability Notes". Vulnerability notes include vendor notification date, vulnerability public date, available solutions, the list of affected vendors, and the list of affected products for each vendor. US-CERT cross-references their vulnerability databases with NVD through the Common Vulnerabilities and Exposures (CVE)[5] identifiers assigned by NVD. In this study, we start with vulnerabilities that have been published by US-CERT, which are then cross-referenced with NVD vulnerability list in order to acquire vulnerability characteristics.

After learning about a vulnerability, US-CERT contacts the vendor(s) to confirm that their products are affected by the vulnerability. If a vendor acknowledges the vulnerability in its product(s), US-CERT lists the vendor's status as "vulnerable". If a vendor reports that its product(s) is not affected by the vulnerability, US-CERT lists the vendor's status as "not vulnerable". However, a vendor may choose not to respond to US-CERT. In this case, US-CERT lists the vendors' status as "unknown". It may be possible that a vendor was affected by the vulnerability even if US-CERT listed a vendor's status as "unknown". However, there is no practical way for us to verify whether a vendor was actually affected by the vulnerability or not. Therefore, we select only those vendors whose status is listed as "vulnerable" by US-CERT.

The unit of analysis is the vulnerability-vendor pair, since vulnerability can affect multiple vendors. We created a list of 792 unique vulnerabilities published by US-CERT from June 21, 2006 to June 20, 2009[6]. Because more information about the vulnerabilities

---

[4] The NVD is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP) standards. It is based on and synchronized with the CVE vulnerability naming standard.

[5] The CVE is a dictionary that provides common identifiers for publicly known information security vulnerabilities and exposures.

[6] Data, except multiple patch release data, have been collected during the period from August 2009 to November 2009. Multiple patch release data have been collected during the period of February 2011.

has to be collected from and cross-referenced with NVD, we removed 109 vulnerabilities from our list that have been published by US-CERT, but not by NVD. From the 256 vendors affected by these vulnerabilities, we created the initial data set of 1222 vulnerability-vendor pairs for 683 vulnerabilities and 256 vendors. From this list, we could not determine the patch release date for 251 observations involving 155 vulnerabilities and 142 vendors and, hence, they were removed from our data set. After calculating the patch release time for the remaining data, 59 observations involving 57 vulnerabilities and 33 vendors had a negative patch release time, which indicates that vendors released a patch before they were notified. These observations were removed from our data set. We dropped 183 observations, for 177 vulnerabilities and 30 vendors, for which the vendors discovered vulnerabilities in their products by themselves or with the help of third parties. Vendors disclosed these vulnerabilities to US-CERT along with the release of patches. In these cases, we cannot exactly determine when a vendor knew the existence of the vulnerability and we cannot determine the actual patch release time. Finally, we dropped 3 observations, for 3 vulnerabilities and 3 vendors, for which vendors had released their patches after the cutoff date. In these cases, patch related information, such as patch release time and patch type, cannot be obtained by the cutoff date. During the data analysis, we identified 4 observations as outliers, and hence they were removed from our data set. The final data set includes 722 observations involving 388 vulnerabilities and 156 vendors. Table 2 provides the descriptive statistics for the vulnerabilities in our data set for different years. In Figure 2, we create a histogram to show the distribution of patch release time for the vulnerability-vendor pairs in our data set. The figure shows that about 50% of vulnerabilities in our data set have been patched

within 30 days after vendors have been notified. The percentage of vulnerabilities patched gradually decreases over time.

TABLE 2: Descriptive Statistics for Vulnerabilities across Years

Year	Number of Observations	Number of Vulnerabilities	Number of Vendors
2006	280	134	28
2007	287	175	52
2008	132	65	59
2009	23	14	17
Total	722	388	156

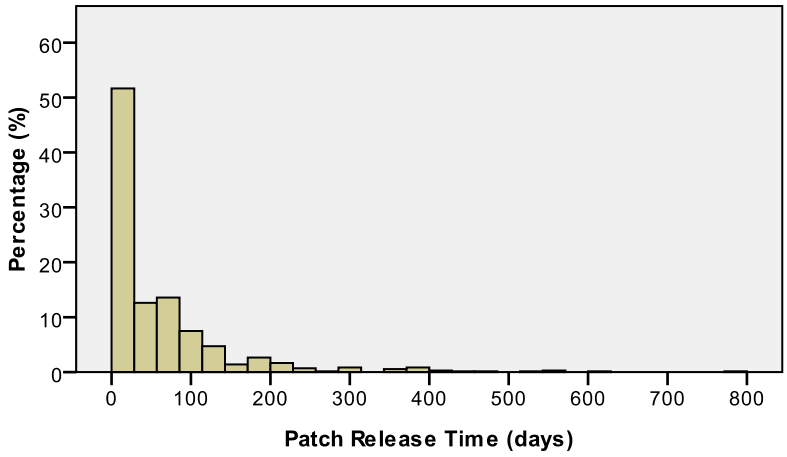

FIGURE 2: Distribution of Patch Release Time

2.7.2. Sample Data

We present some examples of vulnerability information in Table 3. Vulnerability VU#554257 was published by US-CERT and the same vulnerability was published by NVD under the identifier CVE-2007-2798. This vulnerability allows a remote, authenticated user to be able to execute arbitrary code on an affected system or cause the affected program to crash, resulting in a denial of service according to US-CERT. This vulnerability was discovered by an anonymous discoverer working with iDefense. US-CERT lists 45 vendors whose products may be affected by this vulnerability and

contacted all the vendors about the possible existence of this vulnerability in their products. Five vendors acknowledged the vulnerability in their products and US-CERT lists these vendors' status as "vulnerable". Four vendors reported that their products are not affected by this vulnerability and US-CERT lists their status as "not vulnerable". The remaining 36 vendors chose not to respond to US-CERT and their status is listed as "unknown". In this study, we focus on the vendors whose status is listed as "vulnerable" by US-CERT. Sun Microsystems, and Debian GNU are illustrative examples for this vulnerability VU#554257. US-CERT notified both vendors on 6/18/2007 about the existence of this vulnerability in their products. US-CERT made this vulnerability public on 6/26/2007, 8 days after notifying vendors. Debian GNU released its new release type of patch for this vulnerability on 6/28/2007, 10 days after its notification. On the other hand, Sun Microsystems released its update type of patch for this vulnerability on 8/15/2007, 58 days after its notification.

VU#993544 vulnerability was published by the US-CERT, and published by NVD under the name of CVE-2007-3382. This vulnerability can increase the possibility of a session hijacking success according to US-CERT. This vulnerability was reported to US-CERT by a third party. US-CERT notified Apache Tomcat about the possible existence of this vulnerability on 7/2/2007, and made public this vulnerability on 8/13/2007, 42 days after notifying vendors. This vulnerability affected Apache Tomcat version 4, 5, and 6. Apache Tomcat released a new version of Apache Tomcat 6 for this vulnerability 43 days after its notification, i.e., on 8/14/2007. Apache Tomcat released a new version of Apache Tomcat 5 for this vulnerability on 9/8/2007 and a new version of Apache Tomcat 4 for this vulnerability on 2/9/2008. Although US-CERT reported only

Apache Tomcat for this vulnerability, NVD listed multiple vendors such as Apple, HP, SUSE Linux, RedHat, and Apache Tomcat. Therefore, we record that multiple vendors were affected by this vulnerability.

VU#132419 vulnerability was published by the US-CERT, and published by NVD under the name of CVE-2008-1585. This vulnerability allows an attacker to execute arbitrary code and was reported to US-CERT by a member of GNUCITIZEN, a white hat community working with TippingPoint's Zero Day Initiative. Apple Computer was notified by TippingPoint on 5/8/2008 and, hence, the vendor notification date has been recorded for our study from TippingPoint website. TippingPoint and Apple Computer coordinated the patch release and public disclosure of the vulnerability on the same date, i.e., 6/9/2008. Apple Computer released a new version of Apple QuickTime for this vulnerability on 6/9/2008, 32 days after its notification. It also released a new version of Apple TV on 7/10/2008 that fixes this vulnerability.

TABLE 3: Sample Observations of Vulnerability-vendor Pairs

Vendor	Sun Microsystems	Debian GNU	Apache Tomcat	Apple Computer
CERT Name	VU#554257	VU#554257	VU#993544	VU#132419
NVD Name	CVE-2007-2798	CVE-2007-2798	CVE-2007-3382	CVE-2008-1585
Notification Date	6/18/2007	6/18/2007	7/2/2007	5/8/2008
Public Date	6/26/2007	6/26/2007	8/13/2007	6/9/2008
Patch Date	8/15/2007	6/28/2007	8/14/2007	6/9/2008
Patch Release Time	58 days	10 days	43 days	32 days
Disclosure Time	8 days	8 days	42 days	32 days
Confidentiality	Complete	Complete	Partial	Partial
Integrity	Complete	Complete	None	Partial
Availability	Complete	Complete	None	Partial
Patch Type	Update	New Release	New Release	New Release
Vendor Type	Proprietary vendor	OSS vendor	OSS vendor	Proprietary vendor
Software Type	System software	System software	System Software	Application Software
Multiple Vendor	Yes	Yes	Yes	No
Multiple Patches	No	No	Yes	Yes
Affected Product	Solaris	Debian GNU Linux	Apache Tomcat	Apple QuickTime

2.7.3. Data Analysis

One of the assumptions of Cox's proportional hazard model is the absence of outliers and the residual statistics such as deviance residuals that can be used to detect outliers (Allison 1995). Deviance residuals that exceed ±3 indicate possible outliers (Allison 1995). Our analysis showed that the deviance residuals of four observations exceeded -3.0 and these observations were removed from the data set.

Cox's proportional hazard model assumes that the hazards for each independent variable should be proportional over time and the hazard ratio should be constant (Allison 1995). We tested this proportionality assumption with Shoenfeld residuals and our results show that the software vendor type violates the proportionality assumption. According to Allison (1995), one of the remedies for this violation is to treat the violating covariate as a time-dependent variable. Hence, the software vendor type is treated as a time-dependent covariate, allowing the hazard ratio for the vendor type to change over time.

The goodness of fit of the model is tested with the chi-square value of the likelihood-ratio test which refers to the difference between the likelihood measures (-2LL) for the null model and the proposed model (Allison 1995). The Chi-square statistic for our model is 576.41 with 10 degrees of freedom, resulting in a p-value < 0.0001. We reject the null hypothesis that all effects of the independent variables are zero, and, hence, our base model is statistically significant. The results of the model are shown in Table 4.

TABLE 4: Model Results (Dependent Variable: Patch Release Time, N=722)

Variables	Hazard Ratio	Coefficients
Disclosure	2.954 ***	1.083
Multiple Patches	1.359 ***	0.307
Confidentiality	1.226	0.204
Integrity	1.064	0.061
Availability	0.904	-0.101
Patch Type	0.662 ***	-0.412
Vendor Type	10.429 ***	2.345
Software Type	1.451 ***	0.373
Multiple Vendor	1.681 ***	0.519
Vendor Type x Time	0.693 ***	-0.366
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level		

In our base model, confidentiality impact, integrity impact, and availability impact are not statistically significant. The results are contrary to our expectations and do not support our hypotheses regarding vulnerability characteristics. These unexpected results lead us to analyze the correlation between confidentiality, integrity, and availability. Although the definitions given for vulnerability characteristics by FIRST do not overlap, in practice there could be correlations between any pair of vulnerability characteristics. For example, if security attackers gain a right to modify data without authorization as a result of the exploitation of vulnerability, they may also acquire the content of data. Therefore, violations of integrity may also lead to the violation of confidentiality. The modification or destruction of data makes data unavailable or inaccessible. Therefore, violations of integrity may also result in violations of availability.

In addition, an assumption of Cox's proportional hazard model is the absence of multicolinearity (Allison 1995, Hosmer and Lemeshow 1999, Lin and Wei 1989). We examined the correlations between confidentiality, integrity, and availability using the Variance Inflation Factor (VIF) and Pearson Correlation analysis (Fox 1991).

Pearson Correlation analysis indicates statistically significant correlation among confidentiality, integrity, and availability, as shown in Table 5. The highest correlation was found between confidentiality and integrity, which may cause high standard errors for model coefficients.

TABLE 5: Pearson Correlation Analysis (N=722)

Variables	Confidentiality	Integrity	Availability
Confidentiality	1.000	0.928 ***	0.694 ***
Integrity	0.928 ***	1.000	0.631 ***
Availability	0.694 ***	0.631 ***	1.000
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level			

In their study on Cox's proportional hazard model, Poel and Lariviere (2004) indicated that lower correlations result in more stable parameter estimates. Their results show that the correlation coefficient cut-off can be 0.80 for Cox's proportional hazard model (Poel and Lariviere 2004). Given the definitions for vulnerability characteristics by FIRST, each dimension of vulnerability is expected to be different in its impact on information systems security. However, correlation analysis of our data set indicates that confidentiality impact and integrity impact could not be separated, in part due to the limitation of the vulnerability scoring system (CVSS) (Mell and Scarfone 2007). We also tested the seriousness of high correlation between confidentiality and integrity with variance inflation factor (VIF) (Fox 1991). The VIF is found to be 8.40 for confidentiality, 7.23 for integrity, and 1.93 for availability. The VIF values greater than 4.0 for confidentiality and integrity indicate high multicollinearity (Fox 1991). However, after dropping integrity impact in the model, the VIF has become 1.93 for both confidentiality impact and availability impact. Likewise, after dropping confidentiality impact from the data set, the VIF was reduced to 1.66 for both integrity impact and

availability impact. Therefore, we present two alternative models in our paper, with one retaining confidentiality impact and dropping integrity impact and other retaining integrity impact and dropping confidentiality impact.

The Chi-square statistic is 576.20 for the confidentiality model and 574.20 for the integrity model with 9 degrees of freedom, resulting in a p-value < 0.0001. Hence, both of our models are found to be statistically significant. We also evaluate the differences between the confidentiality model and the integrity model using the Likelihood Ratio Test (LRT) (Hosmer and Lemeshow 1999) by creating nested models with the base model. In other words, we tested whether the two models show statistically different model fits. The chi-square value of the LRT test is found as 0.211[7] for the confidentiality model and 2.211[8] for the integrity model with 1 degree of freedom, resulting in a p-value > 0.10. The result of LRT tests indicates that there is no statistical difference in model fits between the two models, so we decided to retain both the confidentiality and integrity models.

Even after resolving the multicollinearity problem with the confidentiality model and the integrity model, availability impact is still not found to be statistically significant although all other independent variables are. This led us to analyze the interaction terms that are missing in the models, since availability impact may not be observable by itself due to the interaction effect with other variables. We use backward stepwise regression starting with all two-way interactions in order to reach a final model, in which the interaction of availability impact with software vendor type is found significant. We

[7] The chi-square value of the LRT test for the confidentiality model, which is the difference between Chi-square statistics of two models, is calculated as follows: (0.211 = 576.41 – 576.20)

[8] The chi-square value of the LRT test for the integrity model, which is the difference between Chi-square statistics of two models, is calculated as follows: (2.211 = 576.41 – 574.20)

analyzed the vulnerability data set after including the interaction between software vendor type and availability impact. The Chi-square statistics is 591.34[9] for the confidentiality model and 589.39[10] for the integrity model with 10 degrees of freedom, resulting in a p-value < 0.0001. Hence, we conclude that our revised models (with the interaction between software vendor type and availability impact) are statistically significant. The results are shown in Table 6.

TABLE 6: Model Results (Dependent Variable: Patch Release Time, N=722)

	Confidentiality Model		Integrity Model	
Variables	Hazard Ratio	Coefficients	Hazard Ratio	Coefficients
Disclosure	2.935 ***	1.077	3.028 ***	1.108
Multiple Patches	1.371 ***	0.316	1.373 ***	0.317
Confidentiality	1.301 ***	0.263	-	-
Integrity	-	-	1.278 ***	0.245
Availability	0.642 ***	-0.443	0.685 ***	-0.379
Patch Type	0.672 ***	-0.397	0.677 ***	-0.390
Vendor Type	5.064 ***	1.622	5.350 ***	1.677
Software Type	1.485 ***	0.396	1.468 ***	0.384
Multiple Vendor	1.643 ***	0.497	1.577 ***	0.455
Vendor Type x Time	0.679 ***	-0.387	0.670 ***	-0.400
Vendor Type x Availability	1.713 ***	0.538	1.712 ***	0.538
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level				

We tested the robustness of our model by applying it on different data sets different time periods. First, we changed the cutoff date from June 20, 2009 to June 20, 2008 and created one data set with 622 observations covering the period from June 21, 2006 to June 20, 2008. Then, we changed the beginning date from June 21, 2006 to December 21, 2006, and created a second data set with 448 observations covering the period from December 21, 2006 to June 20, 2009. Lastly, we created a third data set

[9] The chi-square value of the LRT test for the confidentiality model is found as (-15.14 = 576.20 – 591.34) with 1 degree of freedom, resulting in a p-value < 0.005. The result of LRT test indicates that this model has the better fit.

[10] The chi-square value of the LRT test is found as (-15.19 = 574.20 – 589.39) with 1 degree of freedom, resulting in a p-value < 0.005. The result of LRT test indicates that this model has the better fit.

between December 21, 2006 and June 20, 2008 with 353 observations. The results from the three data sets were not statistically different. In order for data sets to accurately represent all years studied based on the number of vulnerabilities published in each year, we used a stratified random sampling method, and created data sets with 216, 361, and 505 observations. We also found that the results from these three data sets were not statistically different.

2.7.4. Results

The significance of the coefficient of each predictor is used to assess the support for the relevant hypothesis. The hazard ratio (HR) for each predictor is computed using the coefficients of the predictor and any other interactive terms involving the predictor. A HR value greater than 1 for a predictor implies that an increase in the value of the predictor will be associated with a quicker release of a patch by the vendor (positive impact). A HR value less than 1 implies that an increase in the value of the predictor will be associated with a slower release of the path (negative impact) and a HR value of 1 implies no effect of the predictor on the patch release time. The calculations of hazard ratios for our model variables are presented in Appendix A. Although we created two models (the confidentiality model and the integrity model), we use the confidentiality model to explain our main results except when we explain the results regarding integrity impact. The results show that vendor's patch release behavior is affected by the confidentiality impact score, integrity impact score, availability impact score, patch type, software vendor type, software type, patch quality, and the presence of multiple vendors. The result for disclosure (control variable in our model) is consistent with the results of previous studies (Arora et al. 2010a) and show that vulnerabilities that are made public

are patched 2.93 times faster than those that are not made public (see Table A1 in Appendix A).

Vendors release patches 1.30 times faster if the vulnerability's confidentiality impact score is 1 (partial) compared to vulnerabilities with no confidentiality impact. For vulnerabilities with a confidentiality impact score of 2 (complete), the patches are released 1.69 times faster compared to vulnerabilities with no confidentiality impact (see Table A3 in Appendix A). Therefore, our hypothesis *H1* is supported.

The integrity model shows that vendors release patches 1.28 times faster if the vulnerability's integrity impact score is 1 (partial) compared to vulnerabilities with no integrity impact. For vulnerabilities with a integrity impact score of 2 (complete), the patches are released 1.63 times faster compared to vulnerabilities with no integrity impact (see Table A4 in Appendix A). Therefore, our hypothesis *H2* is supported.

Confidentiality and integrity have statistically similar impacts on the patch release time because the VIFs and Pearson Correlation analyses show high correlation between confidentiality and integrity. Thus, they are the most serious vulnerability characteristics that significantly affect the patch release behavior of vendors by decreasing the patch release time. Hence, *H5* is supported, and *H4* is partially supported.

The model indicates that availability impact has an interaction with software vendor type. When we consider software vendor type, the impact of availability score on the patch release behavior is different for proprietary vendors and open source software (OSS) vendors. Vulnerabilities are patched 1.10 times faster by OSS vendors if availability impact is partial compared to no availability impact and 1.21 times faster if availability impact is complete (see Table A5 in Appendix A). Higher availability impact,

therefore, decreases the patch release time of OSS vendors. In contrast, proprietary vendors release patches slower (0.64 times slower if availability impact is partial compared to no availability impact and 0.41 times slower if availability impact is complete) (see Table A5 in Appendix A). Higher availability impact, therefore, increases the patch release time of proprietary vendors. The patch release behavior of proprietary vendors seems contrary to our expectations. One possible explanation is that proprietary vendors prioritize the patching needs for availability impacts differently compared to OSS vendors. Proprietary vendors may expect that vulnerabilities with availability impacts could be countered sooner by customer actions such as backup, redundant sites and other means without having to wait for proprietary vendor's patch release cycle. It could also be that the internalization of social cost in the absence of significant legislation affecting availability (in contrast to confidentiality) is different for PS vendors compared to OSS vendors. In other words PS vendors could view the components of social cost (e.g., loss of reputation, loss of future business, and customer support costs) differently compared to OSS vendors. This is an interesting issue that merits further research.

Vulnerabilities are patched 0.67 times slower if patch type is new release (see Table A6 in Appendix A). In other words, vulnerabilities are patched 1.49 times (i.e., 1/0.67 times) faster if patch type is update. An update type of patch, therefore, is patched faster than a new release type of patches. Thus, *H6* is supported.

Vulnerabilities pertaining to system software are patched 1.49 times faster than vulnerabilities belonging to application software (see Table A7 in Appendix A) and, hence, *H8* is supported. The results show that software vendors' response to

vulnerabilities belonging to different types of software is affected by the level of access to and control of data resources, and a software interface provided by software.

Software vendors release lower quality patches 1.37 times faster than higher quality patches (see Table A8 in Appendix A). Thus, H9 is supported. The results show that software vendors may choose to release lower patch quality to incur less development cost and also to reduce the potential customer loss by the early release of patches, even though of low quality.

Vulnerabilities affecting multiple vendors are patched 1.64 times faster than vulnerabilities affecting single vendor (see Table A2 in Appendix A). Thus, *H10* is supported. Vendor's patch release decision is affected by the presence of other vendors' products with the same vulnerability, and the possibility that other vendors release a patch earlier. The underlying incentive of software vendors to release a patch earlier could avoid the threat of disclosure and customers' penalty on late patching.

The model indicates that software vendor type has interaction with availability impact. It has also an interaction with time. When availability impact is none, we found that OSS vendors release patches 3.44 times faster than proprietary vendors (see Table A9 in Appendix A). When availability impact is partial, we found that OSS vendors release patches 5.89 times faster than proprietary vendors (see Table A9 in Appendix A). When availability impact is complete, we found that OSS vendors release patches 10.10 times faster than proprietary vendors (see Table A9 in Appendix A). Although the patch release time of software vendors is changed based on availability impact, the results show that OSS vendors always release patches faster than proprietary vendors, thus *H7* is supported.

2.8. Discussion and Conclusion

Studying the patch release behaviors of software vendors is an emerging research area with important policy implications. However, research in this area is just beginning to emerge, with one recent pioneering empirical study (Arora et al. 2010a). This paper both reinforces and adds to prior research (Arora et al. 2010a) by studying a proportional hazard model of patch release behavior. A major contribution of this paper is the fact that it highlights differential effects of confidentiality and availability impacts, and integrity and availability impacts on vendor patch release behavior. It also points to the possible importance of legislation as a means of influencing vendor patch release behavior.

The model presented in the previous sections addresses a call in prior research (Arora et al. 2010a) for models with more comprehensive sets of variables to explain software vendor patch release behavior. We have developed a model using cost-based theory that includes development cost as well as internalization of social cost in the presence of governmental action. Such a theory allows us to study different types of vulnerabilities and their impact on vendor patch release behavior. The results presented in the previous section, reinforce prior results and, in addition, provide new results with important implications for policy and future research. We find that the impacts of vulnerability severity, vendor type (open source or proprietary vendor), software type (system or application software), and patch quality on patch release behavior are consistent with prior research (Arora et al. 2010a). Our results also illustrate that vulnerabilities that impact multiple vendors are patched faster than vulnerabilities that impact single vendors, thus reinforcing the value of competition (Arora et al. 2010b, Cavusoglu et al. 2007). Our results point to differential impacts of different types of

vulnerabilities on patch release behavior. We find that vulnerabilities that have high confidentiality impact or high integrity impact are patched fastest. Given that confidentiality impact and integrity impact are governed by legislation, our results have important policy implications and illustrate that legislation could influence vendor behavior in a socially optimal manner.

Though our results are interesting and represent a significant addition to vulnerability disclosure research, additional research opportunities exist to use other methodologies such as survey research. It is important to recognize the limitations of CVSS. The CVSS mainly consists of three metric groups: 1) the base metrics which describe the vulnerability characteristics that are constant over time and across user environments, 2) the temporal metrics which describe vulnerability attributes that change over time but are the same across user environments, and 3) the environmental metrics which describe vulnerability attributes that are user environment specific. The base metrics is mandatory and the CVSS score is calculated based on the values of base metrics. However, the temporal and environmental metrics are optional and are not reported by the FIRST. The CVSS score can be expanded by the combination with the other two optional metrics. It is possible that differential impact of different types of vulnerability could be accentuated due to the optional metrics.

Although the temporal and environmental metrics are not in the scope of this study, the impact of environmental metrics on the results of this study should be considered. Mell and Scarfone (2007) argued that the proper implementation of the CVSS score is environment dependent. This view is consistent with the idea put forth by Frühwirth and Männistö (2009). They argued that the actual impact of software

vulnerabilities may vary across different types of user environments. In particular, this limitation of the CVSS may have an impact on the actual interpretation of availability impact since losing the availability of information systems is often caused by DOS attacks. The availability may be more important to some organizations (e.g., amazon.com) for business continuity, but less important to other organizations or individual users. On the other hand, this limitation is not easily addressed due to the difficulty of collecting environment-specific information. In our study, we assume that vendors do not target the particular type of user groups with the release of patches for their products.

CHAPTER 3: STRUCTURAL DIFFERENCES BETWEEN DIFFERENT TYPES OF OPEN SOURCE SOFTWARE NETWORKS

3.1. Introduction

Traditionally, software has been developed by organizations that do not make the source code of software publicly available. In the traditional software development, software developers have worked in local clusters of collaboration that were generally isolated within firms (Fleming and Marx 2006). More recently, open source software (OSS) development has become the alternative way of developing software. OSS development has brought together software developers spanning firm boundaries (Raymond 1999). OSS development mainly depends on voluntary contributions of software developers and OSS products are developed in a collective manner beyond the boundaries of a single organization (Raymond 1999). Thus, formerly isolated software developers have become large connected networks in OSS development. The network of software developers becomes more important for OSS projects and offers various benefits. First, collaboration among software developers can facilitate access to and sharing of resources, allowing developers to combine their knowledge, skills, and expertise (Raymond 1999).

Second, new insights, ideas or ways to solve problems are conceived by any one and accessed by others (Raymond 1999). Thus, OSS development has changed the conception of how software can be developed. However, not all software projects are completed successfully (Li et al. 2010). Understanding the factors that lead to successful OSS projects is an interesting area of current research. OSS development offers new research opportunities to better understand the network structure of OSS developers.

Software product after delivery is improved by correcting faults or enhanced by adding new features based on user requirements (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). The total cost of software maintenance is estimated to comprise at least 50% of total software life cycle costs (Van Vliet 2000, Kemerer and Slaughter 1999, Kemerer 1995). Thus, the modification of software after delivery is one of the major phases of software development. In software maintenance, we identified two important types of OSS project activities: patch development and feature request. Patch development activities are used to correct faults in software while feature request activities are used to enhance software by adding new features. Recent research on OSS development focused on the analysis of OSS requirements and used feature request activities in their analysis (Vlas and Robinson 2012). Software is defined as a knowledge product (Slaughter et al. 2006) and critical inputs to software development are skills and experience of developers (Li et al. 2010). Therefore, each activity requires different structure of collaboration and knowledge sharing among the developers since each activity has different objectives.

In an organizational context, exploitation and exploration have been identified as two types of activities for the development and use of knowledge in organizations (March

1991). Prior research indicates that different types of tasks require different communication patterns and different amount of communication based on characteristics and nature of a task (Katz and Tushman 1979). A task can differ along several dimensions including time span, specific vs. general problem orientation, and the generation of new knowledge vs. using existing knowledge (Katz and Tushman 1979). March (1991) has suggested that exploitation and exploration represent fundamentally incompatible and inconsistent activities. For example, exploitation represents activities that improve existing organizational competencies and build on the existing technological trajectory. Therefore, exploitation broadens existing knowledge and skills, improves established designs, and expands existing products and services. In contrast, exploration represents activities that changes the organizational competencies and build on a different technological trajectory. Therefore, exploration requires new knowledge, offers new designs, and creates new products and services. In addition, exploitation is related to efficiency, centralization, and tight cultures while exploration is associated with flexibility, decentralization, and loose cultures (Benner and Tushman 2003). Therefore, exploitation and exploration require different organizational structures (Benner and Tushman 2003, Levinthal and March 1993). Different organizational structures for exploitation and exploration enable exploitative teams to develop the best viable solutions, and enable exploratory teams to explore new ideas (Fang et al. 2010).

Recent research on OSS development has focused on the project level (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006). However, this stream of research has not recognized the fact that projects could consist of different types of activities, each of which could require different types of expertise and network structures.

Developers who engage in project activities that are exploitation-oriented may be networked differently compared to those who are engaged in exploration-oriented project activities. Therefore, recent research on OSS development has not differentiated between different types of OSS activities nor between different types of OSS networks. We propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration) based on organizational theory (March 1991). In the context of OSS development, developing a patch would be an example of an exploitation activity. Requesting a new software feature would be an example of an exploration activity. To the best of our knowledge, this is the first research to study OSS development at the activity level.

In this dissertation, we introduce the use of organizational theory on exploration and exploitation together with social network analysis as a theoretical lens to study different types of sub-networks in OSS development. Thus, we studied exploitation and exploration in the content of OSS development. We empirically examined the differences between exploitation (patch development) and exploration (feature request) networks of developers in OSS projects in terms of their social network structure. We used a data set collected from the SourceForge database. Our results indicate that a patch development network has greater internal cohesion and network centrality than a feature request network. In contrast, a feature request network has greater external connectivity than a patch development network.

3.2. Literature Review

There are multiple streams of research that help us to understand the structural differences of OSS networks. A software product after delivery is improved by correcting

faults or enhanced by adding new features based on user requirements (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). Thus, in software maintenance, we identified two important types of OSS project activities: patch development and feature request. Software is a knowledge product (Slaughter et al. 2006) and critical inputs to software development are skills and experience of developers (Li et al. 2010). Therefore, each activity requires different structure of collaboration and knowledge sharing among the developers since each activity has different objectives. Recent studies on social network literature indicated that network structures determine the structure of collaboration and knowledge sharing among actors.

In an organizational context, exploitation and exploration have been identified as two types of activities for the development and use of knowledge in organizations (March 1991). Prior research indicates that different types of tasks require different communication patterns and different amount of communication based on characteristics and nature of a task (Katz and Tushman 1979). March (1991) has suggested that exploitation and exploration represent fundamentally incompatible and inconsistent activities. Exploitation creates a narrow range of deeper solutions and more distinctive competences since exploitation results in the convergence of ideas (March 1991). In contrast, exploration creates a wide range of undeveloped new ideas and limited distinctive competence (March 1991). Therefore, exploitation and exploration require different organizational structures.

3.2.1. Open Source Software Development

Software maintenance is defined as the modification of a software product after delivery to correct faults, to improve performance or other attributes, and to enhance the

product by adapting it to a modified environment (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). Thus, a software product is improved by correcting faults or enhanced by adding new features based on user requirements.

Raymond (1999) indicated that the different nature of software development process for proprietary and OSS vendors leads to two fundamentally different software development styles: the cathedral model for proprietary vendors and the bazaar model for OSS vendors (Raymond 1999). Software development involves knowledge work and its most important resource is the specialized skills and expertise that a developer brings to the project development (Espinosa et al. 2007, Roberts et al. 2004, Faraj and Sproull 2000). Proprietary software is developed in a more closed environment and, hence, proprietary software development is characterized by a relatively strong control of design and implementation (Raymond 1999). In contrast, OSS vendors mainly depend on voluntary contributions of software developers and, hence, OSS products are developed in the collective manner beyond the boundaries of a single organization (Raymond 1999). Therefore, OSS development depends on contributions and collaboration of volunteer software developers (Liu and Iyer 2007, Feller and Fitzgerald 2002). The network of developers becomes more important for OSS projects and offers various benefits. First, collaboration among software developers can facilitate access to and sharing of resources, allowing developers to combine their knowledge, skills, and expertise. Second, new insights, ideas or ways to solve problems are conceived by any one and accessed by others. This leads to increase the performance of developer teams to find a solution for developing patches or to add new features.

Given the benefits of voluntary contributions of software developers for OSS development, the impact of network structure of OSS developer network (Singh et al. 2011, Singh 2010, Grewal et al. 2006) and the formation of OSS developer teams (Hahn et al. 2008) have been intensively studied. Recent studies showed that the network structure of OSS developers significantly affects OSS project success (Singh et al. 2011, Singh 2010, Grewal et al. 2006).

3.2.2. Open Source Software Collaboration Network

In social network literature, an affiliation network is a special kind of network which depends on the affiliation between two groups (Wasserman and Faust 1994). Therefore, an affiliation network has two-modes. The first mode is a set of actors such as developers. The second mode is a set of events such as OSS activities to which the actors belong. The term affiliation refers to membership or participation to events. Therefore, actors are related to each other through their joint affiliation with or their co-membership to events. Events are also related to each other through common actor(s).

OSS software development is a community-based model which involves collaboration among software developers. OSS developers may work on multiple activities concurrently. An activity starts when a developer open new activity under a project. Other developers may join and start participating to an activity. An activity is performed by developers who joined to that activity. Thus, OSS developers belong to multiple activities. A co-membership relationship exists between two developers if they work together on the same activity. Similarly, a relationship between two activities also exists if they share some developer(s). This kind of relationships between developers and activities can be represented by an affiliation network. In OSS network, actors are

developers, and events are activities such as patch development and feature request activities.

3.2.3. Exploitation and Exploration Networks

March (1991) modeled two general situations involving the development and use of knowledge in organizations: the exploitation of old certainties and the exploration of new possibilities. The first is the case of mutual learning between members of an organization. The second is the case of learning and competitive advantage in competition for primacy. Exploitation includes things such as refinement, choice, production, efficiency, selection, implementation, and execution (March 1991). In contrary, exploration includes things captured by terms such as search, variation, risk taking, experimentation, play, flexibility, discovery, and innovation (March 1991). According to Benner and Tushman (2003), exploitation represents activities that involve improvements in existing components and build on the existing technological trajectory. Exploitation is incremental innovations and designed to meet the needs of existing customers or markets (Benner and Tushman 2003). It broadens existing knowledge and skills, improves established designs, and expands existing products and services. Hence, exploitation builds on existing knowledge and reinforces existing skills, processes, and structures (Benner and Tushman 2002, Levinthal and March 1993, Lewin et al. 1999). In contrast, exploration represents activities that involve a shift to a different technological trajectory and changes the organizational competencies. Exploration is radical innovations and designed to meet the needs of emerging customers or markets (Benner and Tushman 2003). It offers new designs, creates new markets. Thus, exploration

requires new knowledge or departures from existing knowledge (Benner and Tushman 2002, Levinthal and March 1993).

For March (1991), exploitation and exploration represent the fundamentally incompatible and inconsistent activities. Exploitation creates a narrow range of deeper solutions and more distinctive competences in the short-run, which comes at the cost of long-term performance since exploitation results in the convergence of ideas by eliminating the differences (March 1991). In contrary, exploration creates a wide range of undeveloped new ideas and too little distinctive competence in the long-term, which comes at the cost of short-term performance (March 1991). Moreover, exploitation is related to efficiency, centralization, and tight cultures while exploration is associated with flexibility, decentralization, and loose cultures (Benner and Tushman 2003). Therefore, exploitation and exploration require different organizational structures (Benner and Tushman 2003, Levinthal and March 1993). Different organizational structures for exploitation and exploration enable exploitative teams to develop the best viable solutions, and enable exploratory teams to explore new ideas (Fang et al. 2010). Recent studies found that organizational units pursuing exploration are smaller, more decentralized, and more flexible than those responsible for exploitation (Benner and Tushman 2003, Christensen 1998, Tushman and O'Reilly 1996).

3.2.4. Social Network and Team Structure

Software development is a highly interdependent task and requires team members to interact with each other intensively to produce a successful system (He et al. 2007). Therefore, interactions among team members are necessary activities to transform team members' knowledge to team knowledge that increase the project success (He et al.

2007). However, the nature of OSS development characterized by volunteer contribution of software developers poses challenges in coordination among developers (Espinosa et al. 2007, Roberts et al. 2004, Banker et al. 2006). Coordination is the process of managing dependencies among activities (Malone and Crowston 1994). When the activities of multiple individuals need to interrelate, the interdependencies among activities should be well managed (Espinosa et al. 2007). Espinosa et al. (2007) indicated that when software is produced from multiple locations, it becomes more difficult to manage dependencies among activities and to coordinate developers, which increases the development time. Therefore, the coordination among developers becomes important for project success in software development.

He et al. (2007) created a model of the formation and evolution of team cognition and analyzed the impacts of preexistent and ongoing collaboration ties on the formation of team cognition in software project teams. Team cognition refers to the mental models collectively held by a group of individuals that enable them to accomplish tasks by acting as a coordinated unit (He et al. 2007). Team cognition helps software project teams effectively manage their members' knowledge, expertise, and skills as integrated assets (He et al. 2007, Espinosa et al. 2007). Team cognition is created by both preexisting conditions and ongoing team interactions. Preexisting conditions reflect both the prior knowledge of team members and any previous shared experiences that team members have. Team interactions refer to the interactive activities that members perform to carry out project tasks and facilitate team performance. He et al. (2007) showed that the positive relationship between team performance and team cognition. Similarly, Hahn et al. (2008) studied the impact of prior collaboration ties on OSS collaboration team

formation mechanisms and on OSS project success. They indicated that team cohesion is related to preference for repeat collaborations and results from prior relationships between developers to benefit from prior relationships. Team members also tend to interact more frequently with other members with whom they share some type of proximity or similarity (Rosenkopf and Almeida 2003, Rosenkopf and Nerkar 2001).

In social network literature, social capital is defined as resources embedded in social networks, and resources that can be accessed or mobilized through social ties in the networks (Coleman 1988, Lin 2005). Through social ties, an actor may capture other actors' resources. These social resources can generate a return for the actor. In addition, because of the facilitative role of network structure, relationships among actors in a network are described as network resources (Gulati 1999). Recent studies also indicated that the position of a team in a network affects team outcomes (Singh et al. 2011, Singh 2010, Zaheer and Bell 2005, Reagans and Zuckerman 2001, Jansen et al. 2006, Schilling and Phelps 2007, Rosenkopf and Almeida 2003, Rosenkopf and Nerkar 2001).

In social network literature, there are two contradictory perspectives about the form of network structures: the internal focus or social closure perspective (Coleman 1988) and the external focus or structural holes perspective (Burt 1992). From Coleman (1988)'s social closure perspective, the optimal social structure is one generated by building dense, interconnected networks. Social closure inside a group indicates the presence of relationships or the absence of structural holes within a group, and is thought to foster identification with the group (Reagans and Zuckerman 2001) and a level of mutual trust, which facilitates exchange and collective action (Coleman 1988). Social closure enables the convergence of individual interests to pursuit common initiatives and

to facilitate mutual coordination (Reagans and Zuckerman 2001). From Burt (1992)'s structural holes perspective, constructing networks consisting of disconnected alters is the optimal strategy. Structural holes perspective focuses value derived from bridging gaps (i.e., structural holes) between nodes in a social network (Burt 1992). This boundary spanning structure generates information benefits since information tends to be relatively redundant within a given group (Burt 1992). As a result, actors who develop ties with disconnected groups gain access to a broader range of ideas and opportunities than those who have restricted access to single group (Granovetter 1973). Although prior research on social network analysis indicated the trade-off between two contradictory perspectives, these two perspectives do not conflict with one another (Reagans and Zuckerman 2001). While the social closure perspective highlights the importance of the presence of relationships in local interactions (i.e., internal cohesion), the external focus perspective highlights information benefits created by structural holes that divide a social network globally (i.e., external cohesion).

Ahuja (2000) studied the impact of social network structures on innovation in terms of direct ties, indirect ties, and structural holes. The debate on structural holes suggests that an accurate understanding of the role of structural holes in the collaboration network must account for both Coleman's and Burt's variants of the argument (Ahuja 2000). Similarly, direct and indirect ties may vary in their content, which highlights the importance of decomposing the firm's ego network into distinct and separate elements and identifying the contents transmitted through each type of tie (Ahuja 2000). According to Ahuja (2000), network ties are associated with two distinct kinds of network benefits. First, they can provide the benefit of resource sharing, allowing firms to combine

knowledge, and skills. Second, collaborative linkages can provide access to knowledge spillovers, serving as information conduits through which news of technical breakthroughs, new insights to problems, or failed approaches travels from one firm to another. In distinguishing between the resource-sharing and knowledge-spillover benefits of collaboration, it is important to distinguish between know-how and information (Kogut and Zander 1992). Know-how entails accumulated skills and expertise in some activity. Information refers primarily to facts that can be transmitted through communication (Kogut and Zander 1992, Szulanski 1996). The resource-sharing benefits of collaboration relate primarily to the transfer and sharing of know-how while the knowledge-spillover benefits are likely to involve predominantly information. Ahuja (2000) found that direct and indirect ties both have a positive impact on innovation but that the impact of indirect ties is moderated by the number of a firm's direct ties. Direct ties potentially provide both resource sharing and knowledge spillover benefits. However, indirect ties do not entail formal resource sharing benefits but can provide access to knowledge spillovers. Structural holes influence both resource sharing and access to novel information (Ahuja 2000). Structural holes have both positive and negative influences on innovation. Specifically, increasing structural holes has a negative effect on innovation, so the optimal structure of networks depends on the objectives of the network members.

Zaheer and Bell (2005) examined the impact of the network structure on the performance and innovativeness of companies by focusing on the external connectivity constructed as structural holes. They highlight the importance of connections to external sources for innovativeness. Zaheer and Bell (2005) found that firms bridging structural holes are more innovative and perform better than other firms. They also indicated that

the internal connectiveness enables firms to further exploit the ideas obtained from external resources.

Jansen et al. (2006) focused on the differences of exploration and exploitation, and examined the impact of internal cohesion and centralization on exploitation and exploration. They found that internal connectedness within teams positively affects the performance of exploitation and exploration teams while centralization negatively affects exploration teams. However, Balkundi and Harrison (2006) indicated that teams that are central in their inter-group network tend to perform better.

Schilling and Phelps (2007) examined the impact of clustering on the innovative output of firms that are members of the network. Innovation is characterized as a process in which solutions are discovered via search process that leads to the creation of new knowledge or the novel recombination of known elements of knowledge, problems, or solutions (Fleming 2001). Schilling and Phelps (2007) indicated the positive association between clustering and innovation output.

Rosenkopf and Nerkar (2001) studied the impact of organization and technology domain on subsequent technological development. They stressed the importance of knowledge internally acquired from the similar technology domains on exploitation, and the importance of knowledge externally acquired from the distinct technology domains on exploration. In other words, organizations can develop more distinctive competence and becomes more expert in their current domain if they focus on their current organizational domain and the similar technological areas. Distinctive competences can improve the performance of developer teams on exploitation (March 1991, Rosenkopf and Nerkar 2001). In contrast, organizations can develop more diverse and less

distinctive competence if they focus on their external organizational domain and the distinct technological areas. More diverse and less distinctive competence can improve the performance of developer teams on exploration (March 1991, Rosenkopf and Nerkar 2001). Lazer and Friedman (2007) on their agent-based simulation model of information sharing found that a network that maintains diversity is better for exploration than other networks, supporting a more thorough search for solutions in the long run.

Social network analysis (Wasserman and Faust 1994) has been used in a variety of contexts to study the relationship between social entities. Based on the findings of social network research (Gnyawali and Madhavan 2001, Ahuja 2000, Uzzi 1999, Uzzi 1997, Uzzi 1996, Watts and Strogatz 1998, Krackhardt 1998, Wasserman and Faust 1994, Burt 1992, Coleman 1988, Freeman, 1979, Granovetter 1973), organizational research (Schilling and Phelps 2007, Hansen 2002, Hansen 1999, Reagans and Zuckerman 2001), and OSS development research (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006), structural properties of the networks are used to analyze the network. Many structural properties of these networks could have multiple social network measures. For example, there are different types of internal cohesion measures (clustering coefficient, repeat ties, third party ties, and structural equivalence), external connectivity measures (external cohesion, direct ties, indirect ties, and technological diversity), and network location measures (degree centrality, betweenness centrality, and closeness centrality).

3.3. Theoretical Background and Hypotheses

In software development literature, software product after delivery is improved by correcting faults or enhanced by adding new features based on user requirements (Banker

and Slaughter 2000, Banker et al. 1998, IEEE 1983). Therefore, we identified two types of OSS project activities: patch development and feature request. Patch development activities are used to correct faults in software while feature request activities are used to enhance software by adding new features. In organizational literature, exploitation and exploration have been identified as two types of activities for the development and use of knowledge in organizations (March 1991). Combining findings of organizational literature and software development literature, we propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration). In the context of OSS development, developing a patch would be an example of an exploitation activity. Requesting a new software feature would be an example of an exploration activity.

Prior research indicates that different types of tasks require different communication patterns and different amount of communication based on characteristics and nature of a task (Katz and Tushman 1979). A task can differ along several dimensions including time span, specific vs. general problem orientation, and the generation of new knowledge vs. using existing knowledge (Katz and Tushman 1979). This is consistent with the view of March (1991) who has suggested that exploitation and exploration represent fundamentally incompatible and inconsistent activities. For example, exploitation represents activities that improve existing organizational competencies and build on the existing technological trajectory. Therefore, exploitation broadens existing knowledge and skills, improves established designs, and expands existing products and services. In contrast, exploration represents activities that changes the organizational competencies and build on a different technological trajectory.

Therefore, exploration requires new knowledge, offers new designs, and creates new products and services. In addition, exploitation is related to efficiency, centralization, and tight cultures while exploration is associated with flexibility, decentralization, and loose cultures (Benner and Tushman 2003). Therefore, exploitation and exploration require different organizational structures (Benner and Tushman 2003, Levinthal and March 1993). Different organizational structures for exploitation and exploration enable exploitative teams to develop the best viable solutions, and enable exploratory teams to explore new ideas (Fang et al. 2010). In addition, software is a knowledge product (Slaughter et al. 2006) and critical inputs to the software development are skills and experience of developers (Li et al. 2010). Therefore, each project activity could require different types of expertise and network structures. In this dissertation, we empirically examined the differences between exploitation (patch development) and exploration (feature request) networks of developers in OSS projects in terms of their social network structure.

3.3.1. Internal Cohesion

OSS development mainly depends on voluntary contributions of software developers and OSS products are developed in the collective manner (Raymond 1999). OSS development process is characterized by the lack of a relatively strong control of design and implementation (Raymond 1999) and the lack of face-to-face communication (Singh et al. 2011). Therefore, OSS teams require constructive environment to foster trust, reciprocity norms and shared identity, and to improve collaboration and cooperation among developers (Singh et al. 2011).

Internal cohesion increases the information transmission capacity of a team (Schilling and Phelps 2007). First, internal cohesion improves access to information since the same information is available via multiple paths (Schilling and Phelps 2007). Information introduced into a team will quickly reach other team members through multiple paths. Multiple paths also enhance the fidelity of information received. Developers can compare information received from multiple partners, helping them to identify whether it is distorted or incomplete (Schilling and Phelps 2007). Second, internal cohesion makes information exchange meaningful and useful (Schilling and Phelps 2007). It can increase the dissemination of alternative interpretations of problems and their potential solutions, deepening the shared understanding and stimulating collective problem solving. Shared knowledge develops over time from prior familiarity with the product being developed and team members (Espinosa et al. 2007, He et al. 2007). Shared knowledge improves coordination among team members because it enables team members to develop more accurate explanations and expectations about tasks and other team members (Espinosa et al. 2007) because prior interactions enable developers to acquire information about skills and capabilities of other developers (Granovetter 1985) and who knows what (Faraj and Sproull 2000). In addition, shared knowledge of problems and solutions enhances further learning (Schilling and Phelps 2007). Third, internal cohesion can make developers more willing and able to improve information exchange and cooperation among team members by fostering trust, reciprocity norms, and shared identity (Coleman 1988, Uzzi and Spiro 2005, Adler and Kwon 2002, Levin and Cross 2004, Hansen 1999, Ahuja 2000). Enhanced trust, reciprocity norms, and shared identity results in a high level of cooperation and

collaboration by providing self-enforcing informal governance mechanisms (Schilling and Phelps 2007). Fourth, internal cohesion fosters group identification which enables the convergence of individual interests to pursuit common initiatives and to facilitate mutual coordination (Reagans and Zuckerman 2001). Fifth, internal cohesion also helps developers to develop team cognition which promote team coordination (Espinosa et al. 2007, He et al. 2007). Team cognition refers to the mental models collectively held by a group of individuals that enable them to accomplish tasks by acting as a coordinated unit (He et al. 2007). Thus, team cognition helps developer teams effectively manage team members' knowledge, expertise, and skills as integrated assets (He et al. 2007, Espinosa et al. 2007).

Internal cohesion results in a high level of cooperation and collaboration among team members. By improving the information transmission capacity of a team, it also enables to exchange and integrate greater amounts of information and knowledge more rapidly. Internal cohesion allows individuals to develop a deep understanding to further refine and improve existing products, and processes (Rowley et al. 2000). However, internal cohesion diffuses strong norms and establishes shared expectations (Uzzi 1997, Rowley et al. 2000). Therefore, it reduces deviant behavior, limits search scope, and increases selective perception of alternatives. Internal cohesion may results in the homogenization of information within a team (Burt 1992, Granovetter 1973) and the convergence of knowledge and ideas (Levinthal and March 1993). Therefore, internal cohesion may limit access to alternative ways of thinking and novel information (Nahapiet and Ghoshal 1998).

Patch development teams (exploitation teams) should be built on existing knowledge and reinforces existing skills, processes, and structures (Benner and Tushman 2003, Levinthal and March 1993, Lewin et al. 1999). Patch development teams broadens existing knowledge and skills, improves established designs, and expands existing products and services (Benner and Tushman 2003). Rowley et al. (2000) indicated that internal cohesion enables team members to develop a deep understanding to further refine and improve existing products and processes. In contrast, feature request teams (exploration teams) should be built upon diverse knowledge that resides outside of the team (Rosenkopf and Nerkar 2001) and require new knowledge (Benner and Tushman 2002, Levinthal and March 1993). Therefore, feature request teams are required to acquire more novel information from external resources than patch development teams. Hence, internal cohesion is more likely to enhance patch development activities when compared to feature request activities. We argue that the internal cohesion of patch development teams is greater than the internal cohesion of feature request teams. This leads us to the following hypothesis:

H1: The internal cohesion of patch development teams will be greater than the internal cohesion of feature request teams.

3.3.2 External Connectivity

Although the internal cohesion of a project team provides various benefits in terms of trust and information transmission capacity, project developers have access to external resources from their relationships to other developers outside of a project team. The structure and type of external relationships affect the ability of project developers to acquire various types of information (Singh et al. 2011). By following prior research, we

focus on the external network structure (the cohesion of external connections), types of external connections (direct ties and indirect ties) and technological characteristics of external connections that affect the diversity of external knowledge available to a focal project.

External connections are associated with two distinct kinds of information benefits (Ahuja 2000). First, they can provide the benefit of resource sharing which allows teams to combine knowledge, and skills acquired from outside teams. Second, they can provide access to knowledge spillovers which serves as information conduits through which news of technical breakthroughs, new insights to problems, or failed approaches acquired from outside project teams. Although direct ties potentially provide both resource sharing and knowledge spillover benefits (Ahuja 2000), they more likely provide redundant information (Hansen 1999). However, indirect ties do not provide resource sharing benefits but can provide access to knowledge spillovers. Therefore, information provided by indirect ties is novel information (Hansen 1999). On the other hand, external cohesion provides both resource sharing and knowledge spillovers benefits (Ahuja 2000). Although how external contacts are connected with each other affects types of information, the characteristics of the external contacts may also affect the diversity of knowledge. External contacts with different technological expertise are more likely to provide novel information and knowledge.

3.3.2.1 External Cohesion

External cohesion is the cohesion among the external contacts of a project (Singh et al. 2011). External cohesion is based on the idea of a structural hole which means the absence of a connection between two developers who are connected to the common third

parties. Therefore, structural holes are defined as gaps in information flows between actors connected to the same actor but not directly connected to each other (Burt 2000). A structural hole separates developers on either side of the hole and creates the brokerage opportunities for those developers to obtain information from disconnected developers (Burt 1992). Therefore, structural holes provide both resource sharing and knowledge spillovers benefits (Granovetter 1973).

External cohesion basically measures the extent to which external contacts of a project are connected to each other. If external contacts of a project are highly connected with each other (high external cohesion or low structural holes), a project is highly constrained to have access to novel information since too much cohesion results in homogenization of information and external contacts of a project may have relatively redundant information (Burt 2004, Burt 1992, Granovetter 1973). However, high external cohesion also enhances trust, reciprocity norms, and shared identity (Coleman 1988, Uzzi and Spiro 2005, Adler and Kwon 2002, Levin and Cross 2004). High external cohesion also improves access to external resources by enhancing information transmission capacity of the network since the same information is available via multiple paths (Schilling and Phelps 2007). Multiple paths also enhance the fidelity of the information received (Schilling and Phelps 2007). In contrast, if external contacts of a project are not connected with each other (low external cohesion or high structural holes), a project have access to novel information from remote parts of the network such as other disconnected project groups (Burt 1992). Therefore, the level of cohesion among the external contacts of a project determines the diversity of knowledge acquired from external contacts.

OSS network is made up of distinct developer teams in which developers are highly connected with each other within each project team, but weakly connected to other developers across other project teams (Singh 2010). Project teams tend to be heterogeneous across a network in terms of the knowledge they possess and produce because each team started with the different initial conditions (Fang et al. 2010). Therefore, external resources provide new knowledge, ideas, and insights (Rosenkopf and Almeida 2003).

Knowledge is developed through combinations of existing and new knowledge (Kogut and Zander 1992). The process of sharing ideas with other projects that have novel information is to generate new knowledge, rather than merely exchanging existing information (Nahapiet and Ghoshal 1998). This idea is consistent with the idea put forth by March (1994) that projects connected to other projects that have novel information may replicate innovative ideas and generate more new ideas which can be used to introduce new and innovative products. A project whose external contacts are not highly connected has access to new knowledge, ideas, and insights from disconnected external projects (Burt 2004, Burt 1992) and they are able to develop new knowledge through knowledge recombination (Rosenkopf and Almeida 2003). Therefore, a project whose external contacts are not highly connected is able to develop new understandings not possible to those whose external contacts are highly connected (Zaheer and Bell 2005). Combining diverse knowledge from other projects (different technology areas) also enhances the capacity for creative learning (Fleming 2001, Kogut and Zander 1992, Reagans and Zuckerman 2001). Feature request teams (exploration teams) should be built upon diverse knowledge that resides outside of the team (Rosenkopf and Nerkar 2001)

and require new knowledge (Benner and Tushman 2002, Levinthal and March 1993). Therefore, feature request teams are required to acquire more novel information from external resources than patch development teams. In order to acquire more novel information from external resources, external contacts of a project should be diversified in terms of knowledge they hold, thereby they should not be highly connected (low external cohesion). We argue that external contacts of patch development teams are more connected with each other than external contacts of feature request teams. This leads us to the following hypothesis:

H2: The external cohesion of patch development teams will be greater than the external cohesion of feature request teams.

3.3.2.2. Direct Ties

Direct ties in a social network potentially provide both resource sharing and knowledge spillover benefits (Ahuja 2000). First, direct ties enable knowledge sharing. When developers collaborate to develop a technology, the resultant knowledge is available to all developers. Thus, each developer can potentially receive a greater amount of knowledge from a collaborative activity than it would obtain from a comparable research investment made independently (Ahuja 2000). Second, collaboration facilitates bringing together complementary skills from different developers. In addition, direct ties among two developers imply opportunities for repeat interactions (Singh et al. 2011). Repeat interactions allow for resource pooling and joint problem solving (Kogut and Zander 1992). However, over time, repeated interactions using the same direct ties are more likely provide redundant information to a focal team (Hansen 1999). Hence, the knowledge spillover benefit which is important for feature request activities could

decrease over time. The resource sharing benefit which is important for patch development activities is more likely greater than knowledge spillover benefit. Direct ties allow developers to combine knowledge and skills using repeating interactions (Kogut and Zander 1992). Repeated interactions through direct ties allow for resource pooling and joint problem solving (Kogut and Zander 1992) which do not decrease due to repeated interactions. Hence, repeated interactions through direct ties are more likely to enhance patch development activities when compared to feature request activities. Since direct ties are also expensive to maintain (Hansen 1999, Hansen 2002, Shane and Cable 2002), we argue that they are more likely to be maintained for repeated use. Therefore, we argue that patch development teams have a large number of direct ties than feature request teams. This leads us to the following hypothesis:

H3: The number of direct ties of patch development teams will be greater than the number of direct ties of feature request teams.

3.3.2.3. Indirect Ties

External connection can be a channel of communication between developers through indirect contacts (Ahuja 2000). An indirect tie between two developers exists when two developers do not work together but can be reached through mutual partners. Therefore, indirect ties provide developers with access not just to knowledge held by their immediate partners but also to knowledge held by their partner's partners (Gulati and Garguilo 1999). However, indirect ties are distant and infrequent relationships (Granovetter 1973). Therefore, they are less likely to provide opportunities for repeat interactions and they are not as conducive to resource pooling as direct ties (Singh et al. 2011). They provide access to novel information by bridging otherwise disconnected

developers (Granovetter 1973). Indirect ties can provide access to knowledge spillovers (Ahuja 2000), serving as information conduits through which news of technical breakthroughs, new insights to problems, or failed approaches travels from one developer to another (Ahuja 2000). Information provided by indirect ties is more likely novel information (Hansen 1999). Innovation is characterized as a process in which solutions are discovered via the creation of new knowledge or the novel recombination of known elements of knowledge, problems or solutions (Fleming 2001). Therefore, the knowledge spillover benefit provided by indirect ties is more important for feature request activities. Distant and infrequent interactions through indirect ties are more likely to enhance feature request activities when compared to patch development activities. Therefore, we argue that feature request teams have a large number of indirect ties than patch development teams. This leads us to the following hypothesis:

H4: The number of indirect ties of feature request teams will be greater than the number of indirect ties of patch development teams.

3.3.2.4. Technological Diversity

Although how external contacts are connected with each other affects types of information, the characteristics of external contacts may also affect the diversity of knowledge since they may vary in terms of technological areas (Rosenkopf and Nerkar 2001). External contacts in different technological areas are more likely to provide novel information and knowledge (Fleming 2001, Kogut and Zander 1992).

Patch development and feature request teams enjoy an enhanced capacity for creative learning since diverse ideas provide alternative ways of thinking, more options for creating new combinations which enhance both problem solving (patch development

teams) and innovation (feature request teams) (Reagans and Zuckerman 2001). However, patch development teams can be built upon similar technology to create distinctive competence (March 1991, Rosenkopf and Nerkar 2001, Henderson and Cockburn 1994). Patch development teams become more expert in their technology area (March 1991, Rosenkopf and Nerkar 2001). Therefore, patch development teams can draw most of their members from similar technology areas to create more distinctive competence. In contrast, feature request teams can develop more diverse and less distinctive competence if they focus on different technological areas. More diverse and less distinctive competence enhances exploration (March 1991, Rosenkopf and Nerkar 2001). Therefore, feature request teams can draw most of their members from different technology areas to create diverse and less distinctive competence.

Rosenkopf and Nerkar (2001) stressed the importance of knowledge acquired from similar technology areas for exploitation, and the importance of knowledge acquired from distinct technology areas for exploration. In other words, patch development teams can develop more distinctive competence and becomes more expert if they focus on their technological areas or similar technological areas. In contrast, feature request teams can develop more diverse and less distinctive competence if they focus on different technological areas. Therefore, we argue that the technological diversity of feature request teams is greater than the technological diversity of patch development teams. This leads us to the following hypothesis:

H5: The technological diversity of feature request teams will be greater than the technological diversity of patch development teams.

3.3.3. Network Location

Centrality is defined as the extent to which an actor occupies a central position in the network (Wasserman and Faust 1994). Developers who are more active in the network act as a central actor in the network and are viewed as major channels of information in the network (Singh et al. 2011, Singh et al. 2007). High centrality enables greater amounts of information and knowledge to be exchanged and integrated more rapidly. First, high centrality allows developers to have a broad range of knowledge, including an understanding where such knowledge is located and how to obtain it (Hansen 2002), which is unavailable to peripheral developers (Lin et al. 2007). Central developers occupy a structurally advantageous position to see a more complete picture of all the alternatives available in the network than the peripheral developers, so they have a broad range of opportunities unavailable to those in the periphery (Lin et al. 2007). A central developer has access to unique knowledge, including an understanding where such knowledge is located and how to obtain it (Hansen 2002). With such information, centrality enables a developer to make better decisions (Balkundi and Harrison 2006). Second, high centrality also allows developers to have quick access to knowledge in the network (Uzzi 1997, Powell and Smith-Doerr 1994). High centrality also allows developers to rapidly disseminate knowledge in the network (Powell and Smith-Doerr 1994). Third, high centrality allows developer to control (Wasserman and Faust 1994, Pfeffer and Salancik 1978), and regulate information flow among other developers (Wasserman and Faust 1994, Krackhardt 1996), dispensing what is needed to other team members (Balkundi and Harrison 2006). Thus, high centrality enhances a developer's ability to be central to the flow of information and resources in the network.

High centrality may allow a developer to have access to greater amounts of relatively redundant knowledge from their immediate contacts (Hansen 2002). Once developers accumulate too much relatively redundant knowledge, they may tent to be blinded to alternative opportunities over time, which leads to learning myopia (Levinthal and March 1993). Central developers may have a tendency to have access to relatively redundant information, which results in the convergence of knowledge and ideas, and may incur the risks of learning myopia (Levinthal and March 1993). Therefore, centrality may be associated with the acquisition of relatively redundant knowledge and experience, which hinders the exploration of new ideas (Lin et al. 2007). Centrality also decreases the likelihood that team members seek innovative and new solutions (Jansen et al. 2006, Atuahene-Gima 2003). Therefore, we argue that centrality is more likely lower for feature request teams. In contrast, Jansen et al. (2006) indicated that centralized authority is beneficial to speeding up exploitation. Exploitation mainly depends on the existing competence and processes, so it is limited in scope and newness (Jansen et al. 2006). Therefore, we argue that the centrality of patch development teams is more likely greater than the centrality of feature request teams. This leads us to the following hypothesis:

H6: The centrality of patch development teams will be greater than the centrality of feature request teams.

3.4. Data

3.4.1. Data Sources and Collection

OSS network data required for this study has been collected from the SourceForge database (SourceForge.net). The SourceForge database is the primary repository for OSS projects and accounts for about 90% of all open source projects (Singh et al. 2011).

Although all OSS projects are not hosted at the SourgeForge database and there are other OSS hosting websites such as BerliOS Developer and GNU Savannah, the SourgeForge database is the largest OSS development and collaboration website (Xu et al 2005). It can be considered as the most representative of the OSS community because the large number of projects and developers registered the SourgeForge database (Singh et al. 2011, Grewal et al. 2006, Xu et al 2005). Researchers analyzing issues related to OSS development phenomenon have predominantly used SourceForge data (Singh et al. 2011, Singh 2010, Singh 2007, Grewal et al 2006). The SourceForge database provides storage space and services to OSS projects in order to organize and coordinate software development activities by providing project web servers, trackers, mailing lists, discussion boards, and software releases (Xu et al 2005). This database contains software for download as well as statistics related to OSS projects. Researchers can create database programs to download statistics that are of interest.

Our research objective is to empirically illustrate the differences between exploration (feature request) and exploitation (patch development) networks of developers in OSS projects in terms of their social network structure. Therefore, we need to collect affiliation network data in order to construct these networks. Given a set of activities (patch development and feature request) and developers, there are two methods to collect affiliation network data: Snowball method and Whole network method (Hanneman and Riddle 2005). The whole network method yields maximum information, but it can also be difficult to execute while the snowball method yields considerably less information about network structure, but it is often less difficult to implement (Hanneman and Riddle 2005).

The snowball method begins with a focal actor or set of actors. Then, all the actors connected to a focal actor or set of actors are tracked down. The snowball process continues until no new actors are identified, or a large enough number of observations is collected for analysis. However, there are major potential limitations of the snowball method (Hanneman and Riddle 2005). First, actors who are not connected (i.e. actors in different components) are not reached through this method. The snowball method may tend to overstate the connectedness and solidarity of populations of actors based on the starting actors and their connectivity to other actors. Therefore, there is no guaranteed way of finding all of the connected individuals in the population.

The whole network method requires that we collect information about each developer's ties with all other developers. Because we collect information about ties between all developer-activity pairs, full network data gives the complete picture of relations in the population (Hanneman and Riddle 2005). Whole network data is necessary to properly define and measure many of the structural concepts of network analysis (Hanneman and Riddle 2005). Whole network data also allows for very powerful descriptions and analyses of social structures (Hanneman and Riddle 2005). However, whole network data may also be very difficult to collect. The data collection task is made more manageable by determining an appropriate boundary around the network since the whole network method examines actors that are regarded as bounded social collectives (Marsden 2005, Singh et al. 2011). This is the predominant method used in situation where an appropriate network boundary is established. Prior studies on OSS development used software development platforms called project foundries as a network boundary. Project foundries are mainly built on programming languages, thereby project foundry

and programming language are similar concepts. For example, Singh et al. (2011) used participation in Python foundry (uses Python programming language) and Grewal et al. (2006) used participation in Perl foundry (uses Perl programming language) as a network boundary. However, foundry data associated with OSS projects was not available at the SouceForce database after 2005. Therefore, there is no way for us to associate projects with foundries.

We used the whole network method to collect affiliation network data and selected the C programming language as a network boundary. The selection of the C programming language as a network boundary is acceptable for several reasons. First, it is the system implementation language for the UNIX operating system and UNIX/Linux operating system is dominant in OSS community (Subramanian et al. 2009). Second, it is one of the preferred languages of OSS developers for codes that require portability, need faster processing, have real-time requirements, or are tightly coupled to the UNIX/Linux kernel (Subramanian et al. 2009). Third, developers who are familiar with the programming language are able to understand the source code easily (Subramanian et al. 2009), thereby more efficient knowledge sharing may be possible within a project or across projects written in the same programming language. Fourth, we analyzed the number of projects and associated developers across programming languages and found that the C language is in the top three languages used by the large number of software developers at SourgeForge.

Data collection started by identifying developer-activity pairs since OSS developers may work on multiple projects simultaneously if they are members of different artifact teams (either different patch development or feature request activities).

A relationship exists between any two developers if they are members of the same artifact team and consequently work together on the same activity. These kinds of relationships between developers and activities can be represented by an affiliation network (Wasserman and Faust 1994). Affiliation data for activities and developers (associated with projects) has been collected from the SourceForge database for projects registered from January 1999 to December 2008 at the SourceForge website. We have set December 2008 as a cutoff date for our study for several reasons. First, constructing feature request and patch networks (developer affiliation networks for different feature request and patches) and calculating a variety of social network measures are extremely computation intensive especially for larger networks. We used social network software (UCINET) (Borgatti et al. 2002) to perform calculations and wrote our own code when required to construct networks as well as to perform some calculations. We analyzed the number of developers for projects written in the C language for each year from 2003 to 2011. We found that networks (especially project developers' network used in Chapter 4) have large number of developers (≥15,000) after December 2008 as shown in Table 7. This results in extremely large networks that are challenging to process with UCINET. Second, the first data snapshot of the SourceForge database is available for January 2003. The difference between our cutoff date and the first data snapshot date of the SourceForge data is 5 years which provides sufficient variation in network characteristics. Third, we had a concern for data availability of our dependent variables (the number of versions) in Chapter 4 because the SourceForge database provides data for our dependent variables until December 2008.

TABLE 7: Project Statistics across Years

Years	Number of Projects	Number of Developers
Jan 31, 2003	741	4,371
Dec 31, 2004	1,271	6,999
Dec 31, 2005	1,532	8,400
Dec 31, 2006	1,830	9,935
Dec 31, 2007	2,117	11,330
Dec 31, 2008	2,374	12,665
Dec 31, 2009	2,515	14,933
Dec 31, 2010	2,608	15,564
Dec 31, 2011	2,665	15,950

In order to identify developer-activity pairs, we identified all projects that match following criteria. First, we included the projects which are written in the C language (our network boundary). Second, we excluded projects which have neither patch nor feature request activities in order to ensure the creation of developer-activity pairs. If a project has neither patch nor feature request activities, that project does not yield a developer-activity pair in our networks. This also ensures the calculation of project ambidexterity as described in Chapter 4. Prior research also indicated that a large proportion of projects hosted at the SourceForge database show no activity (Singh et al. 2011, Singh 2010, Chengalur-Smith and Sidorova 2003). These projects would be dead nodes in the network and the relationships involving them would not facilitate any knowledge transfers or spillovers (Singh et al. 2011). Therefore, including such projects in the network may lead to misleading results. Following prior research (Singh et al. 2011, Singh 2010), we excluded those projects. If a project has neither patch nor feature request activities, we considered those projects as inactive because we assume that they showed no sign of activity since their inception until December 2008. For the projects that match our criteria, we identified all patch development and feature request activities that have been successfully closed by using their “Activity ID”, “Activity Descriptions” and

"Status". Patch development activities are defined as activities to correct faults in software (SourceForge.net). Feature request activities are defines as software enhancement activities to add new features based on new user requirement (SourceForge.net). Then, we identified the developers who joined to either patch development or feature request activities. This allows us to collect separate affiliation network data (developer-activity pairs) and construct separate affiliation networks for projects for patch and feature request activities.

Based on the finding of organizational literature (Jansen et al. 2009, Gupta and Govindarajan 2000), some developers are expected to be members of teams involved in exploitative activities (patch development) and members of teams involved in exploratory activities (feature request). Consistent with the finding of organizational literature, we identified a new category of developers (ambidextrous developers) in OSS projects who contribute to multiple types of OSS activities. Therefore, we identified three types of developers in the OSS community: patch developers, feature request developers, and ambidextrous developers. Patch developers are developers who work on patch development activities while feature request developers are developers who work on feature request activities. Ambidextrous developers are developers who are members of patch development and feature request teams and consequently work on both patch development and feature request activities simultaneously. Therefore, there is an overlapping between patch developers and feature request developers and patch development and feature request networks. Although the focus of this chapter is to empirically illustrate the differences between exploration (feature request) and

exploitation (patch development) networks, we develop a theoretical construct for ambidexterity based on the concept of ambidextrous developers in Chapter 4.

Two separate affiliation networks were constructed based on the type of activities: a patch network, and a feature request network. A patch network includes developers involved in patch development activities (patch developers and ambidextrous developers). A feature request network includes developers involved in feature request activities (feature request developers and ambidextrous developers).

3.4.2. Network Construction

OSS network data analyzed in this study is the affiliation data between developers and activities. Social network of the OSS community is represented by an affiliation network such as a two-mode network based on a developer-activity pair. However, in order to analyze the structure of OSS networks, we need a one-mode network at the developer level. Therefore, we construct a patch network of developer and a feature request network of developer in two steps.

We construct separate affiliation networks for patch development activities and feature request activities based on developer-activity pairs. In these affiliation networks, the actors are unique developers, and the events are either patch development or feature request activities. A relationship exists between two developers if they work together on the same activity. Figure 3 illustrates the process of developer affiliation network construction. In Figure 3a, each activity has its own set of developers. A square node represents a unique activity and a circular node represents a unique developer. A link between any two developers exists if they work on the same activity. Figure 3b shows the developer network for individual activities. However, some developers (D5 and D10)

work on more than one activity simultaneously. Thus, they belong to more than one activity team and they are used to connect the individual teams in the network as shown in Figure 3c (which shows the affiliation network of developers across activities). In Figure 3c, a node represents a unique developer. We construct two separate affiliation networks of developers for patch development and feature request activities.

A binary adjacency matrix (the matrix A) of affiliation networks represents the relationships between activities and developers in the network in Figure 4a. The adjacency matrix of affiliation networks lists unique developers across multiple activities for a patch network and a feature request network. A row represents developers, and a column represents activities. When a developer belongs to an activity, the corresponding matrix element gets a value of one, and zero otherwise. The transpose (the matrix A^T) of an adjacency matrix of affiliation networks represents the relationships between activities and developers in the network in Figure 4b. A row represents activities, and a column represents developers. We converted two-mode network data to one-mode network data by multiplying an adjacency matrix (the matrix A) of affiliation networks with the dot product of the transpose (the matrix A^T) of an adjacency matrix of affiliation networks expressed as follows:

$$X^A = AA^T$$

Adjacency matrixes (the matrix X^A) of a patch network and a feature request network represent the relationships between any two developers in Figure 4c. The row and the column represent unique developers. A value of one or more corresponding to the pair of two developers in the network indicates a presence of a relationship between them, and a value of zero indicates the absence of relationship. The adjacency matrix is

undirected because relationship among two developers is mutual. We converted all values greater than one to one which simply indicates a presence of a relationship between two developers. These final adjacency matrices are our final patch and feature request networks which are used in our analysis. The final patch network includes 23,603 artifacts and 4,727 unique developers under 1,173 projects. The final feature request network includes 31,504 artifacts and 6,656 unique developers under 1,892 projects.

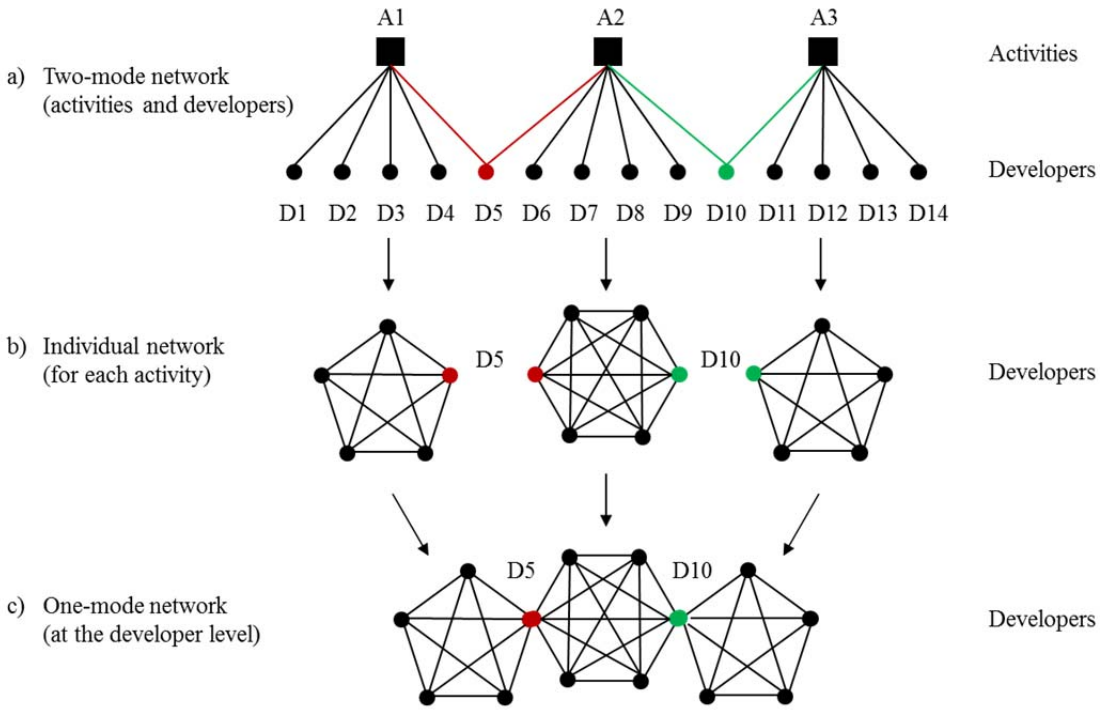

FIGURE 3: OSS Network Construction at the Activity Level

$$A = \begin{array}{c} \\ D1 \\ D2 \\ D3 \\ D4 \\ D5 \\ D6 \\ D7 \\ D8 \\ D9 \\ D10 \\ D11 \\ D12 \\ D13 \\ D14 \end{array} \begin{array}{c} \begin{array}{ccc} A1 & A2 & A3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

a) Two-Mode Adjacency Matrix of Activities and Developers

$$\boldsymbol{A^T} = \begin{array}{c} \\ A1 \\ A2 \\ A3 \end{array} \begin{array}{c} \begin{array}{ccccccccccccc} D1 & D2 & D3 & D4 & D5 & D6 & D7 & D8 & D9 & D10 & D11 & D12 & D13 & D14 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

b) Transpose of Two-Mode Adjacency Matrix of Activities and Developers

FIGURE 4: Matrix Representations of OSS Networks at the Activity Level

$$X^{A} = \begin{array}{c} \\ D1 \\ D2 \\ D3 \\ D4 \\ D5 \\ D6 \\ D7 \\ D8 \\ D9 \\ D10 \\ D11 \\ D12 \\ D13 \\ D14 \end{array} \begin{array}{c} \begin{array}{cccccccccccccc} D1 & D2 & D3 & D4 & D5 & D6 & D7 & D8 & D9 & D10 & D11 & D12 & D13 & D14 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

c) One-Mode Adjacency Matrix of OSS Activities

FIGURE 4: Cont'd

3.5. Variable Definitions and Operationalization

OSS network data analyzed in this study is the affiliation network data of developers at the artifact level. However, we aggregated data to the project level in order to test our hypotheses because of following concerns. First, some projects have relatively more artifacts while some projects have relatively few artifacts. In Table 8, the maximum number of artifacts for projects is 2791 in patch development network and 862 in feature request network while the minimum number of artifacts for projects is 1 in both patch development network and feature request network. Second, artifacts under the same project have almost the same set of developers. Therefore, most of the observations may be the same at the artifact level. Therefore, OSS network data analyzed in this study is an aggregated data at the project level to eliminate repeated observations.

TABLE 8: Descriptive Statistics for Patch and Feature Request Networks

	Patch Network	FR Network
Number of Project	1,173	1,892
Artifact Level Statistics		
Total Number of Artifact	23,603	31,504
Average Number Artifact per Project	20.12	16.65
StdDev of Number Artifact per Project	123.95	52.70
Min Number Artifact per Project	1	1
Max Number Artifact per Project	2791	862
Developer Level Statistics		
Total Number of Developers	4,727	6,656
Number of Ambidextrous Developers	3,140	3,140
Number of Artifact Developers	1,587	3,516
Average Number of Developers per Project	4.37	3.73
StdDev of Number of Developers per Project	5.78	5.26
Min Number of Developers per Project	1	1
Max Number of Developers per Project	73	75

We aggregated the affiliation network data of developers to the project level in four steps. First, we calculated social network measures for individual developers (i.e. clustering coefficient, the number of direct ties) and developer pairs (i.e. the number of repeat ties, the number of third party ties) in patch development and feature request networks. These network measures are used to calculate variables. Second, we associated developers to activities by using “Developer ID” and “Activity ID” from their membership to activities. Third, we associated activities to projects by using “Activity ID” and “Project ID”. This allowed us to associate developers to projects by creating relationship between “Developer ID” and “Project ID”. Thus, we identified the set of unique developers for each project. However, some developers work on multiple activities under the same project. We ensured that those developers are represented only one time under each project since we identified unique developers by removing their multiple occurrences. Fourth, we calculated variables for each project from network measures of project developers. The final data set for feature request activities includes

1,892 projects and 6,656 unique developers. The final data set for patch development activities includes 1,173 projects and 4,727 unique developers.

Social network analysis (Wasserman and Faust 1994) has been used in a variety of contexts to study the relationship between social entities. Structural properties of the networks are used to analyze the network. Many structural properties of these networks could have multiple social network measures. For example, there are different types of internal cohesion measures (clustering coefficient, repeat ties, third party ties, and structural equivalence), external connectivity measures (external cohesion, direct ties, indirect ties, and technological diversity), and network location measures (degree centrality, betweenness centrality, and closeness centrality). Consistent with previous studies on social network research (Gnyawali and Madhavan 2001, Ahuja 2000, Uzzi 1999, Uzzi 1997, Uzzi 1996, Watts and Strogatz 1998, Krackhardt 1998, Wasserman and Faust 1994, Burt 1992, Coleman 1988, Freeman, 1979, Granovetter 1973), organizational research (Schilling and Phelps 2007, Hansen 2002, Hansen 1999, Reagans and Zuckerman 2001), and OSS development research (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006), we categorized our social network variables into three categories: internal cohesion, external connectivity, and network location. In the following section, we describe our variables used in this study along with the construction of their measures.

3.5.1. Internal Cohesion

We measured internal cohesion for a project with clustering coefficient, repeat ties, third party ties, and structural equivalence (Jaccard similarity and correlation similarity).

Clustering Coefficient: The clustering coefficient captures the degree to which the overall network contains localized pockets of dense connectivity (Watts and Strogatz 1998, Watts 1999). The clustering coefficient mainly measures the extent to which two related developers share a relationship with a common third.

We measured the clustering coefficient for a project by following Watts and Strogatz (1998). For each project developer, we calculated the clustering coefficient (see Appendix B for the calculation of clustering coefficient). We took an average of each project developer's clustering coefficient over all the project developers to calculate a measure of the clustering coefficient for a project.

The clustering coefficient lies strictly in the range from 0 to 1. The value of 1 indicates that all developers in the network share a direct relationship with each other. That means each developer is directly connected to all other developers in the network, which results in extreme clustering. In contrast, the value of 0 indicates that any two connected developers do not share a relationship with a common third. A high score of the clustering coefficient indicates greater clustering.

Repeat Ties: Repeated collaboration among project members captures the strength of interpersonal connections among team members (Uzzi 1996, Uzzi 1999, Singh et al. 2011). Strong interpersonal connections indicate the presence of repeat collaborations among project members (Uzzi 1997). As developers interact more frequently, the strength of the collaborative tie increases, and they develop more closer and cohesive relationships (Granovetter 1973, Hansen 1999). Team members rely on repeated ties developed through joint participation in past teams because they are motivated to continue to work with those with whom they have collaborated in the past (Hahn et al.

2008). Repeated ties from past interactions may result in greater trust and knowledge for developers (Uzzi and Spiro 2005).

We measured the number of repeated ties for a project by following Singh et al. (2011). We counted the total number of projects on which each pair of project developers have worked together. We divided this number by the total number of pairs that exist in a project to calculate a measure of repeat ties for a project. A high score of repeat ties indicates that project developers have worked together on several projects.

Third Party Ties: Third party ties support direct relationships and imply that a project team is composed of developers who work with many of the same collaborators (Szulanski 1996, Coleman 1988, Singh et al. 2011). Third part ties are important for the existence of effective norms and the trustworthiness in social structures (Coleman 1988). Similarly, the concept of simmelian ties are the same with third party ties (Krackhardt 1998). Two people are simmelian tied to one another if they are reciprocally and strongly tied to each other and to another one in common (Krackhardt 1998). Simmelian ties enhance the conflict resolution and group norms (Krackhardt 1998).

We measured the number of third party ties for a project by following Singh et al. (2011). We counted the total number of third party ties of all pairs of project developers around the members of a project team (besides the focal team members). We divided this number by the total number of pairs that exist in a project to calculate a measure of third party ties for a project. A high score of third party ties indicates that project developers have worked together with other developers on several projects.

Structural Equivalence: The structural equivalence measures to the extent to which two actors have identical relationships to all other actors, i.e. they jointly occupy

the structurally equivalent position in the network (Wasserman and Faust 1994). Thus, the structural equivalence is a pair-level measure of how similar the actors' network patterns are. Structurally equivalent actors have a similar pattern of relationships to other actors in the network (Wasserman and Faust 1994, Gnyawali and Madhavan 2001). Structurally equivalent actors tend to have similar profiles and behaviors (Gnyawali and Madhavan 2001). Structurally equivalent actors tend to interact with similar others in similar ways, which results in similar attitudes, resources, and behaviors (Gnyawali and Madhavan 2001). Therefore, structurally equivalent actors may have similar asset, information, and resources (Gnyawali and Madhavan 2001).

We measured the structural equivalence for a project with two measures: Jaccard similarity, and Correlation similarity (Wasserman and Faust 1994). Jaccard similarity measures the similarity of the relationships of two developers by comparing the size of the overlap against the size of the relationships of two developers (Wasserman and Faust (1994). Correlation similarity measures the similarity of the relationships of two developers by calculating Pearson's correlation of the relationships of two developers (Wasserman and Faust (1994). Correlation similarity measures the strength of the relationship between two developers and it is based on the similarity in pattern of ties whereas Jaccard similarity account for the identity of ties between two developers.

We calculated Jaccard similarity and correlation similarity as follows. We calculated the total of Jaccard similarity and correlation similarity of all pairs of project developers. We divided these numbers by the total number of pairs that exist in a project to calculate measures of Jaccard similarity and correlation similarity for a project. Jaccard similarity and correlation similarity lie strictly in the range from 0 to 1. A value of one

represents perfect structural equivalence whereas a value of zero represents no structural equivalence. A high score of structural equivalence indicates that project developers worked with many of the same developers.

3.5.2. External Connectivity

We measured external connectivity for a project with external cohesion, direct ties, indirect ties, and technological diversity.

External Cohesion: We measured the external cohesion with Burt's (1992) network constraint. Network constraint measures the extent to which a project member's external contacts share relationships with each other.

We calculated the external cohesion for a project as follows. For each project developer, we calculated the network constraint (see Appendix B for the calculation of external cohesion). We took an average of each project developer's network constraint over all the project developers to calculate a measure of the network constraint for a project. Higher values of external cohesion indicate that external contacts of a project are more directly connected with each other, which indicates greater external cohesion. In contrast, lower values of external cohesion indicate that external contacts of a project are less directly connected with each other, which indicates smaller external cohesion.

Direct Ties: We measured direct ties by following Ahuja (2000). Direct ties measure the extent to which project members are directly connected to external contacts. Direct ties are also associated with the capacity of a project to acquire tacit knowledge from outside (Singh et al. 2011).

We calculated direct ties for a project as follows. For each project developer, we counted the number of developers who a project developer has ties with other than the

other team members of the project. We took an average of this number over all the project developers to calculate a measure of direct ties for a project. Higher values of direct ties indicate that a project is more directly connected to external contacts.

Indirect Ties: Indirect ties are ties that provide access to external developers at path distances of two or greater (local project developers' partner's partners), which excluded direct ties. Indirect ties measure the extent to which project members are indirectly connected to external partner's partners. Indirect ties are also associated with the capacity of a project to acquire explicit knowledge from outside (Singh et al. 2011).

We used two measures for indirect ties. The first measure is the number of indirect ties. For each project developer, we counted the number of developers with whom a project developer does not have a direct tie but can reach through others (at path distances of two or greater, which excluded direct ties). We took an average of this number over all the project developers to calculate a measure of indirect ties for a project.

This measure does not account for the weakening or decay of tie strength as distance between two developer's increases (Ahuja 2000). Burt (1992) provided a frequency decay measure for indirect ties that accounts for this decline in tie strength across distant ties (see Appendix B for the calculation of indirect ties with frequency decay function). Thus, our second measure for indirect ties is a frequency decay measure proposed by Burt (1992). The argument for the frequency decay function is that the rate at which the strength of a relation decreases with the increasing length of its corresponding path distance should vary with the social structure in which it occurs (Burt 1992). The larger the number of developers to which the focal project developer must devote their time and energy, the weaker the relationship that the focal project developer

can sustain with any individual developer. Thus, decay in the strength of a relationship is related to the number of other developers reached at each path distance.

For each project developer, we calculated a frequency decay function for indirect ties. We took an average of this number over all the project developers to calculate a measure of indirect ties with a frequency decay function for a project. Higher values of indirect ties indicate that a project is more indirectly connected to external partner's partners at path distances of two or greater.

Technological Diversity: Technological diversity measures the extent to which two projects are different in terms of the angular distance of their technological positions. In order to calculate the technological diversity for a project, we defined the technological position of a project. The technological position of a project can be defined in terms of different dimensions such as the type of the project, programming language, user interface, and operating system (Singh et al. 2011). Each of these dimensions represents different type of technical expertise. Project type represents the application domain knowledge whereas the other three dimensions represent the tools knowledge and expertise that comprise the knowledge of process, data and functional architecture (Kim and Stohr 1998, Singh et al. 2011). The similarity of domain and tools affect the amount of knowledge that can be reused from one project to another (Singh et al. 2011).

Following Jaffe (1986), we characterized a project's technological position by a vector $F_p = (F_1 \ldots F_k)$, where k is the total number of categories under the four dimensions, and F_k is an indicator variable that equals to 1 if the project p falls under the category k. A project can fall under several categories within a single dimension. Technological diversity between the two projects p and q is then calculated by the angular separation or

uncentered correlation of their vectors (see Appendix B for the calculation of technological diversity).

We calculate the technological diversities of all pairs of a focal project with all of the projects with which it shares a developer. We summed these measures and divided it by the number of projects (the total number of project pairs) to calculate a measure of technological diversity for a project. Technological diversity lies in the range from 0 to 1. A value of one represents the greatest technological diversity between two projects.

3.5.3. Network Location

We measured network location for a project with network centralities: degree centrality, closeness centrality, and betweenness centrality.

Degree Centrality: We measured the degree centrality with Freeman's (1979) degree centrality. Degree centrality is the measure of how many an actor is connected to other actors in the network through direct connections (Freeman 1979, Wasserman and Frost 1994). Degree centrality of a developer reflects the activeness of a developer in the network. Developers who are more active in the network act as a central actor in the network and are viewed as major channels of information in the network (Singh et al. 2011, Singh et al. 2007).

We calculated the degree centrality for a project as follows. For each project developer, we calculated the degree centrality (see Appendix B for the calculation of degree centrality). We took an average of each project developer's degree centrality over all the project developers to calculate a measure of the degree centrality for a project.

The degree centrality is normalized by dividing by the maximum possible degree in the network which is that one actor is connected to all other actors in the network. This

calculation results in that the degree centrality lies in the range from 0 to 1. However, UCINET reports the normalized degree centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of degree centrality for a project ranges from 0 to 100. A high score of the degree centrality indicates a project is comprised of developers who are connected to many developers in the network.

Betweenness Centrality: We measured the betweenness centrality with Freeman's (1979) betweenness centrality. Betweenness centrality is the measure of how often a developer falls on the shortest path between pairs of other developers (Freeman 1979, Wasserman and Faust 1994). Developers with a high betweenness centrality lie in the shortest path of information flow between other developers. These developers can exert control over information flow among other developers, and potentially may have some control over the interactions between other developers (Wasserman and Faust 1994). Thus, betweenness centrality signifies a developer's ability to be central to the flow of information and resources in the network. These developers can be important to the network-wide information diffusion process by occupying a central position on the shortest path between other developers in a network.

We calculated the betweenness centrality for a project as follows. For each project developer, we calculated the betweenness centrality (see Appendix B for the calculation of betweenness centrality). We took an average of each project developer's betweenness centrality over all the project developers to calculate a measure of the betweenness centrality for a project.

The betweenness centrality is normalized by dividing by the maximum possible betweenness in the network which is the number of pairs of actors not including a focal actor (the maximum possible paths passing through a focal actor). This calculation results in that the betweenness centrality lies in the range from 0 to 1. However, UCINET reports the normalized betweenness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of betweenness centrality for a project ranges from 0 to 100. A high score of the betweenness centrality indicates a project is comprised of developers who fall on many shortest paths between other developers.

Closeness Centrality: We measured the closeness centrality with Freeman's (1979) closeness centrality. Closeness centrality is the measure of how close an actor is to all other actors in the network through direct and indirect connections (Freeman 1979, Wasserman and Frost 1994). It basically measures the inverse of the sum of geodesic distances between actors in the network, thereby an actor with high closeness centrality has minimum geodesic distances to other actors. Closeness centrality signifies a developer's ability to reach resources in the network (Gulati and Gargiulo1999). Information would have to travel over shorter distances to reach a developer who is more central in the network (Wasserman and Faust 1994). A developer who is close to many developers can quickly interact and communicate with them without passing through many intermediaries (Wasserman and Faust 1994).

We calculated the closeness centrality for a project as follows. For each project developer, we calculated the closeness centrality (see Appendix B for the calculation of closeness centrality). We took an average of each project developer's closeness centrality

over all the project developers to calculate a measure of the closeness centrality for a project.

The closeness centrality is normalized by multiplying by the maximum possible path distance in the network which is that one actor is connected to another one actor passing through all other actors in the network. This calculation results in that the closeness centrality lies in the range from 0 to 1. However, UCINET reports the normalized closeness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of closeness centrality for a project ranges from 0 to 100. A high score of the closeness centrality indicates a project is comprised of developers who are very close to all other developers in the network via shortest paths.

3.6. Research Methodology

Our research objective is to empirically examine the differences between exploitation (patch development) and exploration (feature request) networks of developers in OSS projects in terms of their social network structure. In order to accomplish these research objectives and test the hypothesis developed in the previous sections, we employed two statistical methods. First, we used the paired T-test (Cohen 1988, Cohen 1977) to determine whether there is a statistically significant difference between the patch development and feature request networks in terms of their social network structure. With the paired T-test, we tested the difference between the patch development and feature request networks at the project level by using social network variables (internal cohesion, external connectivity, and network location). Second, we also used the Quadratic Assignment Procedure (QAP) (Hubert and Schultz 1976, Hubert

1987) in order to examine the degree of dissimilarity between patch development and feature request networks. The QAP test preserves the integrity of the network structures. With the QAP test, we tested the difference between the patch development and feature request networks at the network level. The QAP test also provides greater reliability to the findings the paired T-test and improves its robustness.

3.6.1. The Paired T-test

The paired t-test compares the means of two related groups to detect whether there are any statistically significant differences between their means (Cohen 1988, Cohen 1977). The paired t-test is the within-groups design in which subjects in each group are matched into pairs and the same subjects contribute to independent variables in each group (Ha and Ha 2012). The major advantage of the within-groups design is to minimize the amount of error variance associated with individual differences that occur between subjects and this increases the power of the test (Ha and Ha 2012, Cohen 1988, Myers and Well 1991).

The paired t-test requires that subjects in two groups should be paired. The mean difference score of two groups is a measure of independent variables that will be compared to the mean difference score of the null hypothesis. The mean difference score of the null hypothesis is assumed to be zero. If there is no difference between two groups in terms of independent variables, the mean difference score of paired groups will be zero or very close to the mean difference score of the null hypothesis. However, if there is difference between two groups in terms of independent variables, the mean difference score of two groups will be greater or less than zero. Therefore, the assumptions of the paired t-test are centered on the difference scores of two groups.

OSS network data analyzed in this study is the affiliation network data of developers at the artifact level. We created a list of 1,173 projects for patch development activities and a list of 1,892 projects for feature request activities from the SourceForge database for projects registered from January 1999 to December 2008. As described in the variable definition part, we aggregated data to the project level in order to test our hypotheses. The paired t-test assumes that the observations in the two groups should be related (Cohen 1988, Myers and Well 1991). Therefore, we have matched projects from patch development and feature request networks into pairs and each project contributes to independent variables of both patch development and feature request networks. Therefore, the unit of analysis is the pair of projects belonging to both patch development and feature request networks.

One of important issues for the paired t-test is the absence of outliers. An outlier is an observation with an extreme value and univariate statistics such as a standard score can be used detect outliers (Tabachnick and Fidell 2007). Standard scores that exceed ±3 indicate possible univariate outliers (Tabachnick and Fidell 2007). We detected outliers for each individual variable and then removed all observations that have at least one outlier for at least one variable. The final data set includes 690 observations (projects) belonging to both patch development and feature request networks. We tested our hypotheses by the final data set including 690 observations (projects). We report the descriptive statistics of our variables in Table 9.

The paired t-test also assumes that the difference scores of paired groups should follow the normal distribution (Cohen 1988, Myers and Well 1991). We tested this

normality assumption with the Kolmogorov-Smirnov and Shapiro-Wilk tests of normality. We found that the difference scores of paired groups are normally distributed.

TABLE 9: Descriptive Statistics of Paired Variables (N=690)

Variable Type	Variable Name	Mean	Std. Dev.	Std. Error Mean
Internal Cohesion	Clustering Coefficient (Patch)	0.574	0.462	0.018
	Clustering Coefficient (FR)	0.541	0.474	0.018
	Repeat Ties (Patch)	0.771	0.473	0.018
	Repeat Ties (FR)	0.752	0.474	0.018
	Third Party Ties (Patch)	0.153	0.780	0.030
	Third Party Ties (FR)	0.133	0.763	0.029
	Jaccard Similarity (Patch)	0.495	0.458	0.017
	Jaccard Similarity (FR)	0.473	0.464	0.018
	Correlation Similarity (Patch)	0.659	0.437	0.017
	Correlation Similarity (FR)	0.660	0.444	0.017
External Connectivity	External Cohesion (Patch)	0.749	0.312	0.012
	External Cohesion (FR)	0.767	0.304	0.012
	Direct Ties (Patch)	5.382	6.615	0.252
	Direct Ties (FR)	4.938	6.244	0.238
	Indirect Ties (Patch)	2.546	4.648	0.177
	Indirect Ties (FR)	6.721	18.529	0.705
	Indirect Ties FD (Patch)	0.065	0.119	0.005
	Indirect Ties FD (FR)	1.655	5.439	0.207
	Technological Diversity (Patch)	0.195	0.276	0.011
	Technological Diversity (FR)	0.184	0.273	0.010
Network Location	Degree Centrality (Patch)	0.110893	0.132161	0.005031
	Degree Centrality (FR)	0.072070	0.087780	0.003342
	Betweenness Centrality (Patch)	0.000439	0.001819	0.000069
	Betweenness Centrality (FR)	0.000092	0.000357	0.000014
	Closeness Centrality (Patch)	0.021226	0.000136	0.000005
	Closeness Centrality (FR)	0.011553	0.006378	0.000243

Pearson correlation analysis indicates statistically significant correlations between paired variables (see Table C1 in Appendix C). This is the within-groups design in which we used the same projects belonging to both patch development and feature request activities, thereby project characteristics are the same for patch development and feature request activities. The significant correlations between paired variables indicate that the

differences between patch development and feature request activities are associated with network structures of developer teams, not other random effects.

3.6.2. Results of the Paired T-test

The significance of the t statistic of each paired variable is used to assess the support for the relevant hypothesis. The null hypothesis assumes that the mean difference of paired variable will be zero. The mean difference displays the average difference between patch development and feature request teams for each variable. The mean difference greater than zero implies that the mean of the variable of patch development teams is greater than the mean of the variable of feature request teams. The mean difference smaller than zero implies that the mean of the variable of patch development teams is smaller than the mean of the variable of feature request teams. We summarize the results of our hypotheses in Table 10. We report the results of the paired T-test in Table 11.

TABLE 10: Summary of Hypotheses

Variable Type	Hypotheses	Tested with Variable	Results	Comments
Internal Cohesion	Hypothesis 1	Clustering Coefficient	Supported	
	Hypothesis 1	Repeat Ties	Supported	
	Hypothesis 1	Third Party Ties	Not Supported	Not significant
	Hypothesis 1	Jaccard Similarity	Supported	
	Hypothesis 1	Correlation Similarity	Not Supported	Not significant
External Connectivity	Hypothesis 2	External Cohesion	Not Supported	Opposite of hypothesis
	Hypothesis 3	Direct Ties	Supported	
	Hypothesis 4	Indirect Ties	Supported	
	Hypothesis 4	Indirect Ties FD	Supported	
	Hypothesis 5	Technological Diversity	Not Supported	Not significant
Network Location	Hypothesis 6	Degree Centrality	Supported	
	Hypothesis 6	Betweenness Centrality	Supported	
	Hypothesis 6	Closeness Centrality	Supported	

We measured internal cohesion for a project with clustering coefficient, repeat ties, third party ties, Jaccard similarity, and correlation similarity. In the hypothesis *H1*, we expect that *the internal cohesion of patch development teams will be greater than the internal cohesion of feature request teams*. We found support for our first hypothesis for clustering coefficient, repeated ties, and Jaccard similarity. The mean of the clustering coefficient of patch development teams is 0.033 points greater than the mean of the clustering coefficient of feature request teams, and this difference is significant (2.974, $p < 0.01$). The internal cohesion of patch development teams is greater than the internal cohesion of feature request teams in terms of clustering coefficient. The mean of the repeat ties of patch development teams is 0.019 points greater than the mean of the repeat ties of feature request teams, and this difference is significant (1.813, $p < 0.1$). The internal cohesion of patch development teams is greater than the internal cohesion of feature request teams in terms of repeat ties. The mean of Jaccard similarity of patch development teams is 0.021 points greater than the mean of Jaccard similarity of feature request teams, and this difference is significant (2.130, $p < 0.05$).

TABLE 11: The Paired T-test Results (N=690)

Variable Type	Paired Variable Names	Mean Difference	Std. Dev.	Std. Error Mean	t	Sig. (2-tailed)
Internal Connectivity	Clustering Coefficient (Patch) Clustering Coefficient (FR)	0.033	0.293	0.011	2.974	0.003 ***
	Repeat Ties (Patch) Repeat Ties (FR)	0.019	0.270	0.010	1.813	0.070 *
	Third Party Ties (Patch) Third Party Ties (FR)	0.020	0.391	0.015	1.348	0.178
	Jaccard Similarity (Patch) Jaccard Similarity (FR)	0.021	0.264	0.010	2.130	0.033 **
	Correlation Similarity (Patch) Correlation Similarity (FR)	-0.001	0.267	0.010	-0.054	0.957
External Connectivity	External Cohesion (Patch) External Cohesion (FR)	-0.017	0.176	0.007	-2.572	0.010 **
	Direct Ties (Patch) Direct Ties (FR)	0.444	3.556	0.135	3.283	0.001 ***
	Indirect Ties (Patch) Indirect Ties (FR)	-4.174	16.414	0.625	-6.680	0.000 ***
	Indirect Ties FD (Patch) Indirect Ties FD (FR)	-1.590	5.392	0.205	-7.744	0.000 ***
	Technological Diversity (Patch) Technological Diversity (FR)	0.011	0.234	0.009	1.252	0.211
Network Location	Degree Centrality (Patch) Degree Centrality (FR)	0.039	0.076	0.003	13.400	0.000 ***
	Betweenness Centrality (Patch) Betweenness Centrality (FR)	0.000	0.002	0.000	5.559	0.000 ***
	Closeness Centrality (Patch) Closeness Centrality (FR)	0.010	0.006	0.000	40.028	0.000 ***
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level						

The internal cohesion of patch development teams is greater than the internal cohesion of feature request teams in terms of Jaccard similarity. Therefore, our hypothesis *H1* is supported by results of clustering coefficient, repeated ties, and Jaccard similarity. However, we did not find support for our first hypothesis for third party ties and correlation similarity since the difference between patch development and feature request teams are not significant at the 0.10 alpha level. The internal cohesion of patch development teams is the same as the internal cohesion of feature request teams in terms of third party ties and correlation similarity. Therefore, our hypothesis *H1* is partially supported. The results of repeat ties and third party ties merit further discussion. Repeat ties and third party ties are based on social interactions among developers. One possible explanation for the insignificance of third party ties is that there may be a few social interactions for the pairs of developers with common third parties, and these interactions may not be have enough strength to support third party ties. In addition, third party ties measure the number of relationship of a pair of developers to common third parties outside the focal project team. Therefore, third party ties do not measure strong relationships between two developers, but measure the relative relationship of already connected two developers to the common third. Thus, they may represent relatively loose connections. The common third developer is an outside developer of a focal team, and thereby, that developer may not directly foster trust, reciprocity norms and shared identity within a focal team which facilitate collaboration and cooperation among focal project team members. In contrast, repeat ties capture the strength and deepness of the relationship between two developers. The strength and deepness of relationship indicates two developers interact more frequently, and they develop more closer and cohesive

relationships. Repeat ties from past interactions also result in greater trust within a focal team. The results of Jaccard similarity and correlation similarity also merit further discussion. Jaccard similarity and correlation similarity are measures for the structural equivalence of developers. Correlation similarity measures the strength of the relationship between two developers in terms of the similarity in a connectivity pattern of ties between two developers. However, Jaccard similarity accounts for the identity of ties between two developers, i.e. who is connected to who. Although Jaccard similarity considers the identity of ties, correlation similarity does not consider the identity of ties. Connecting to the same developers is more important than connecting the same number of developers in terms of internal cohesion. If developers are connected to the same developers, they may develop more closer and cohesive relationships which results in greater trust within a focal team.

We measured external connectivity for a project with external cohesion, direct ties, indirect ties, and technological diversity. However, we developed different hypotheses for each external connectivity measures. In the hypothesis *H2*, we expect that *the external cohesion of patch development teams will be greater than the external cohesion of feature request teams*. The mean of the external cohesion of patch development teams is 0.017 points smaller than the mean of the external cohesion of feature request teams, and this difference is significant (-2.572, $p < 0.01$). Therefore, the external cohesion of patch development teams is smaller than the external cohesion of feature request teams. Although the network structures of patch development and feature request teams are different in terms of external cohesion, the result is contrary to our expectations and does not support our hypothesis H2. External cohesion measures the

extent to which external contacts of a project are connected to each other. Low external cohesion allows a focal project which is connected to disconnected projects to acquire more novel information from those disconnected projects. We selected the C programming language as a network boundary. Within our network boundary, all projects use the C programming language. Therefore, all projects in our data set are technologically similar in terms of programming language. This is consistent with the results of our fifth hypothesis. As explained later, we found that the technological diversity of patch development teams is the same as the technological diversity of feature request teams. The selection of the C programming language eliminates other projects using different programming languages. Projects using different programming languages may develop distinct knowledge from other projects using the same programming language. They may be technologically diverse and provide access to novel information. In addition, they may not be highly connected to each other. This means lower external cohesion for a local project. Feature request teams may be more connected to projects using different programming languages than patch development teams. However, the selection of the C programming language may remove external connections to other projects using different programming languages. This may result in high external cohesion for feature request teams since most external connections of feature request teams may have been removed. The inclusion of multiple programming languages may produce results which will be consistent with our hypothesis regarding external cohesion.

In the hypothesis *H3*, we expect that *the number of direct ties of patch development teams will be greater than the number of direct ties of feature request teams*. The mean of direct ties of patch development teams is 0.444 points greater than the mean

of direct ties of feature request teams, and this difference is significant (3.283, $p < 0.01$). The number of direct ties of patch development teams is greater than the number of direct ties of feature request teams. Therefore, our hypothesis *H3* is supported.

In the hypothesis *H4*, we expect that *the number of indirect ties of feature request teams will be greater than the number of indirect ties of patch development teams*. We measured indirect ties for a project with the number of indirect ties, and the number of indirect ties calculated with frequency decay function. We found support for our fourth hypothesis for both measures. The mean of indirect ties of patch development teams is 4.174 points smaller than the mean of indirect ties of feature request teams, and this difference is significant (-6.680, $p < 0.01$). In addition, the mean of frequency decayed indirect ties of patch development teams is 1.590 points smaller than the mean of frequency decayed indirect ties of feature request teams, and this difference is significant (-7.744, $p < 0.01$). The number of indirect ties of patch development teams is smaller than the number of indirect ties of feature request teams. Therefore, our hypothesis *H4* is supported.

In the hypothesis *H5*, we expect that *the technological diversity of feature request teams will be greater than the technological diversity of patch development teams*. However, we did not find support for our fifth hypothesis since the difference between patch development and feature request teams are not significant at the 0.10 alpha level. The technological diversity of patch development teams is the same as the technological diversity of feature request teams. Therefore, our hypothesis *H5* is not supported. We selected the C programming language as a network boundary. Within our network boundary, all projects use the C programming language. Therefore, all projects in our

data set are technologically similar in terms of programming language. The selection of the C programming language eliminates other projects using different programming languages. Projects using different programming languages may develop distinct knowledge from other projects using the same programming language. They may be technologically diverse. The inclusion of multiple programming languages may produce results which will be consistent with our hypothesis regarding technological diversity.

We measure network location for a project with network centralities: degree centrality, betweenness centrality, and closeness centrality. In the hypothesis *H6*, we expect that *the centrality of patch development teams will be greater than the centrality of feature request teams*. We found support for our sixth hypothesis for the degree centrality, betweenness centrality, and closeness centrality. The mean of the degree centrality of patch development teams is 0.038 points greater than the mean of the degree centrality of feature request teams, and this difference is significant (13.400, $p < 0.01$). The centrality of patch development teams is greater than the centrality of feature request teams in terms of degree centrality. The mean of the betweenness centrality of patch development teams is 0.00034 points greater than the mean of the betweenness centrality of feature request teams, and this difference is significant (5.559, $p < 0.01$). The centrality of patch development teams is greater than the centrality of feature request teams in terms of betweenness centrality. The mean of the closeness centrality of patch development teams is 0.021 points greater than the mean of the closeness centrality of feature request teams, and this difference is significant (40.028, $p < 0.01$). The centrality of patch development teams is greater than the centrality of feature request teams in terms of

closeness centrality. Therefore, our hypothesis *H6* is supported by results of the degree centrality, betweenness centrality, and closeness centrality.

3.6.3. Power Analysis for the Paired T-test

The power of a statistical test is the probability that the test will reject the null hypothesis when the null hypothesis is false, i.e. the probability of not committing a Type II error (Cohen 1988, Greene 2003). The statistical power is calculated as (1 – ß) where the ß (beta) is the Type II error (the probability of failing to reject the null hypothesis when it is false).

When the alpha (α) is set at 0.05, Cohen (1988) assumes that the risk of failure to find the beta (β) may be about four times less serious than the risk of finding what does not exist (α). The test with the power greater than 0.80 is considered statistically powerful at the 0.05 alpha level (Cohen 1988, Mazen et al. 1985). Given the number of observations (N=690 projects) and the significance alpha level (α=0.05), we calculated the power (1– β) of our T-tests by following Cohen (1988). We report the results of the power test in Table 12.

We found that the power of all variables except unsupported variables (third party ties, correlation similarity, and technological diversity) is greater than the cut-off point of 0.80. The high statistical power indicates that the T-test more likely detects the true effect of the phenomenon and rejects the null hypothesis. The high statistical power also indicates that the sample size for those variables are more than enough. However, the power of unsupported variables (third party ties, correlation similarity, and technological diversity) is lower than the cut-off point of 0.80. This indicates that the results of the T-test for unsupported variables are not powerful. We may need more observations for

those variables since greater sample size reduces the standard error and increases the statistical power (Cohen 1988, Mazen et al. 1985).

TABLE 12: The Statistical Power of the Paired T-tests (Alpha = 0.05)

Variable Type	Paired Variable Names	Power
Internal Connectivity	Clustering Coefficient (Patch) Clustering Coefficient (FR)	0.997
	Repeat Ties (Patch) Repeat Ties (FR)	0.888
	Third Party Ties (Patch) Third Party Ties (FR)	0.757
	Jaccard Similarity (Patch) Jaccard Similarity (FR)	0.960
	Correlation Similarity (Patch) Correlation Similarity (FR)	0.051
External Connectivity	External Cohesion (Patch) External Cohesion (FR)	0.994
	Direct Ties (Patch) Direct Ties (FR)	>0.999
	Indirect Ties (Patch) Indirect Ties (FR)	>0.999
	Indirect Ties FD (Patch) Indirect Ties FD (FR)	>0.999
	Technological Diversity (Patch) Technological Diversity (FR)	0.312
Network Location	Degree Centrality (Patch) Degree Centrality (FR)	>0.999
	Betweenness Centrality (Patch) Betweenness Centrality (FR)	>0.999
	Closeness Centrality (Patch) Closeness Centrality (FR)	>0.999

3.6.3. Quadratic Assignment Procedure (QAP)

The QAP is a social analysis method to compare two networks (Baker and Hubert 1981) and examine the degree of dissimilarity between them (Hubert and Schultz 1976, Hubert 1987). The QAP is used to test the null hypothesis that two social network are uncorrelated or dissimilar (Hubert and Schultz 1976, Hubert 1987). The QAP is a nonparametric permutation-based test that preserves the integrity of the network structures. The QAP can determine the distribution of all possible correlations given the

structures of two matrices by generating all correlations that result from permuting the rows and columns of one matrix to those of second matrix.

The QAP has several advantages. First, it does not impose any specific distributional assumptions since it is a permutation-based nonparametric test (Baker and Hubert 1981). Second, it takes advantage of the dyadic information represented in each matrix by preserving the integrity of the network structures (Baker and Hubert 1981). Third, it can be used for non-independent relationships (Baker and Hubert 1981). Fourth, it is immune to the highly complex autocorrelation structure of network data (Krackhardt 1987, Krackhardt 1988). Fifth, the QAP is relatively unbiased (Krackhardt 1987).

The QAP requires social networks which should be represented in the form of square matrixes with equal size. However, the patch development network includes 4,727 unique developers whereas the feature request network includes 6,656 unique developers. Therefore, we created separate sample networks for patch development and feature request activities by extracting developers along with their network connections with each other from patch development network and feature request networks.

We have identified three types of developers in the OSS community: patch developers, feature request developers, and ambidextrous developers. The patch network consists of patch and ambidextrous developers whereas the feature request network consists of feature request and ambidextrous developers. In order to accurately represent ambidextrous and non-ambidextrous developers in each network, we used a stratified random sampling method. Each sample network consists of 1,000 developers stratified based on the ratios of ambidextrous and non-ambidextrous developers in each network. We created 25 stratified sample networks for patch development activities, and 25

stratified sample networks for feature request activities. We created 25 sample networks pairs by pairing patch development sample networks and feature request sample networks.

We used the QAP test as implemented in UCINET 6 (Borgatti et al. 2002). The QAP test reports five (similarity) measures (Jaccard, correlation, simple matching, Goodman-Kruskal Gamma, and Hamming distance). Although the result of the QAP test is the same across five measures, we report the result of the QAP test with Jaccard similarity and correlation similarity between sample networks of patch development and feature request activities since these two measures are our variables as shown in Table 10.

We report the results of the QAP test in Table 13. We compared 25 stratified sample network pairs from patch development and feature request networks. In Table 13, each row is a comparison of two stratified sample networks from patch development and feature request networks. We found that sample patch development and sample feature request networks are not similar with each other. Although the correlation and Jaccard similarities for the network pairs 5 and 17 are significant at 0.1 alpha level, the correlation and Jaccard similarities are as low as 0.003 which indicates that sample patch development and sample feature request networks for the network pairs 5 and 17 have different network structures. For other pairs, we found that sample patch development and sample feature request networks are not similar with each other since the correlation and Jaccard similarities are very low (e.g., 0.002) and they are not significant. We accept the null hypothesis that two network are uncorrelated. The results of the QAP test are

consistent with the results of the paired T-test. Therefore, we found that patch development and feature request networks have different network structures.

TABLE 13: Comparison of Stratified Sample Networks of Developers from Patch Development and Feature Request Networks (Network Size=1000)

Sample Patch Network	Sample FR Network	Correlation Similarity	Sig.	Jaccard Similarity	Sig.
1	1	0.000	0.616	0.001	0.616
2	2	0.000	0.621	0.001	0.648
3	3	0.000	0.464	0.001	0.464
4	4	0.001	0.348	0.001	0.348
5	5	0.003	0.084 *	0.002	0.084 *
6	6	0.001	0.308	0.001	0.308
7	7	0.001	0.440	0.001	0.857
8	8	0.001	0.353	0.000	0.888
9	9	0.001	0.474	0.001	0.844
10	10	0.001	0.420	0.000	0.858
11	11	0.002	0.121	0.002	0.121
12	12	0.000	0.545	0.001	0.545
13	13	0.000	0.676	0.001	0.606
14	14	0.001	0.475	0.001	0.834
15	15	0.000	0.564	0.001	0.564
16	16	0.002	0.110	0.002	0.110
17	17	0.002	0.092 *	0.002	0.092 *
18	18	0.001	0.420	0.000	0.858
19	19	0.000	0.498	0.001	0.498
20	20	0.002	0.176	0.000	1.000
21	21	0.002	0.159	0.002	0.159
22	22	0.001	0.321	0.001	0.321
23	23	0.001	0.394	0.001	0.394
24	24	0.002	0.131	0.002	0.131
25	25	0.002	0.126	0.000	1.000
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level					

3.7. Discussions and Contributions

We empirically examined the differences between exploitation (patch development) and exploration (feature request) networks of developers in OSS projects in terms of their social network structure. In order to accomplish these research objectives and test our hypothesis, we employed two statistical methods. First, we used the paired T-

test to determine whether there is a statistically significant difference between the patch development and feature request networks in terms of their social network structure. With the paired T-test, we tested the difference between the patch development and feature request networks at the project level by using social network variables for internal cohesion, external connectivity, and network location. Our results for the paired T-test show that patch development and feature request networks have different network structures. Our results indicate that a patch development network has greater internal cohesion and network centrality than a feature request network. In contrast, a feature request network has greater external connectivity than a patch development network. We tested the statistical power of the results of the paired T-test. The power analysis indicates that the T-test more likely detects the true effect of the phenomenon. The high statistical power also indicates that the sample size is more than enough. Second, we also used the QAP test in order to examine the degree of dissimilarity between patch development and feature request networks at the network level. The results of the QAP test are consistent with the results of the paired T-test. The QAP test provides greater reliability to the results of the T-test.

We found that the internal cohesion of patch development teams is greater than the internal cohesion of feature request teams. As measured by clustering coefficient, repeated ties, and Jaccard similarity, our findings indicate that different measures of internal cohesion are consistent and patch development teams have greater internal cohesion than feature request teams. Our results indicate that developers in patch development teams have greater trust with each other due to high internal cohesion, which improves collaboration and cooperation, and facilitates information exchange in

patch development teams. However, the result of internal cohesion measured by third party ties is not significant. This could be because, although repeat ties and third party ties are based on social interactions among developers, repeat interactions between two developers are much stronger than third party interactions with common third parties. Thus, repeat interactions result in greater trust within a focal team. In addition, the result of internal cohesion measured by correlation similarity is not significant. Jaccard similarity and correlation similarity measures the structural equivalence of developers. However, Jaccard similarity considers the identity of ties whereas correlation similarity does not consider the identity of ties. The results could indicate that connecting to the same developers is more important than connecting the same number of developers since developers may develop more closer and cohesive relationships which results in greater trust within a focal team.

We found that the external cohesion of patch development teams is smaller than the external cohesion of feature request teams. Our results also indicate that the technological diversity of patch development teams is the same as the technological diversity of feature request teams. These results are contrary to our expectations possibly because of the choice of one programming language as a network boundary. Within our network boundary, all projects are technologically similar in terms of programming language. The selection of one programming language eliminates other projects using different programming languages. Projects using different programming languages may develop distinct knowledge from other projects using the same programming language. They may be technologically diverse and provide access to novel information. In addition, they may not be highly connected to each other. This means lower external

cohesion for a local project. Feature request teams may be more connected to projects using different programming languages than patch development teams. However, the choice of one programming language may remove external connections to other projects using different programming languages. This may result in high external cohesion for feature request teams since most external connections of feature request teams may have been removed. The inclusion of multiple programming languages may produce results which will be consistent with our hypotheses regarding external cohesion and technological diversity.

We found that the number of direct ties of patch development teams is greater than the number of direct ties of feature request teams. Our results indicate that direct ties facilitate resource pooling by enabling patch development teams to combine more (relatively redundant) knowledge with repeating interactions than feature request teams. We found that the number of indirect ties of patch development teams is smaller than the number of indirect ties of feature request teams. Our results indicate that indirect ties enable feature request teams to access more novel information through knowledge spillovers than patch development teams.

We found that the centrality of patch development teams is greater than the centrality of feature request teams. As measured by degree centrality, betweenness centrality, and closeness centrality, our findings indicate that different measures of centrality are consistent and patch development teams have greater centrality than feature request teams. Our results indicate high centrality enables patch development teams to exchange and integrate greater amounts of information more rapidly. It also enables patch development teams to control and regulate information flow among developers.

By providing a more nuanced understanding of different types of sub-networks in OSS development, this dissertation makes several important theoretical and practical contributions.

From a theoretical perspective, we introduce the use of organizational theory on exploration and exploitation together with social network analysis as a theoretical lens to study different types of sub-networks in OSS development. Recent research on OSS development has studied the social network structure of software developers as determinant of project success (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006). However, this stream of research has focused on the project level, and has not recognized the fact that projects could consist of different types of activities, each of which could require different types of expertise and network structures. We propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration) based on organizational theory (March 1991). In the context of OSS development, developing a patch would be an example of an exploitation activity. Requesting a new software feature would be an example of an exploration activity. To the best of our knowledge, this is the first research to study OSS development at the activity level. Our data selection and analysis method are different from prior research and novel.

This dissertation develops the theory for and then empirically tests the differences between exploration and exploitation networks in OSS development in terms of their social network structure. Our empirical results illustrate that these two types of networks are significantly different in terms of their social network structure.

We identify a new category of developers (ambidextrous developers) in OSS projects who contribute to exploitative activities (patch development) as well as exploratory activities (feature request). A new theoretical construct for project ambidexterity has been developed based on the concept of ambidextrous developers. A nuanced understanding of different types of activities and the concept of ambidextrous developers open opportunities for future research as discussed in the next chapter.

This dissertation also makes several important contributions to practice. We show that exploitation and exploration activities in OSS development require specific network structures based on characteristics and nature of each activity. Therefore, we provide OSS project leaders with a way to optimize their exploitative and exploratory teams based on requirements of each activity. OSS project leaders can allow some developers to specialize in each activity. They can allow some developers to work on both activities in order to enhance the ability of those developers (ambidextrous developers) to integrate exploitative and exploratory teams. In a result, they can possibly better manage OSS projects. This discussion is evolved further in the next chapter.

3.9. Limitations and Future Research

We examine the differences between exploitation (patch development) and exploration (feature request) networks of developers in OSS projects in terms of their social network structure. We assume that network structure affects knowledge transfer. However, we did not observe knowledge transfer directly but rather infer it from the relationship between network structure and project performance. Knowledge may flow through other mechanisms. For example, a developer may acquire knowledge from unconnected activities by using their software or by analyzing their software's source

code. In this dissertation, we did not consider other mechanisms for knowledge flow. We did not analyze characteristics of individual team members such as their experiences and motivations which may also influence the extent to which knowledge is transferred or absorbed (Cohen and Levinthal 1990). These aspects of relationships can be analyzed in order to understand network structures in detail. These limitations have been recognized in prior research on OSS social networks (Singh et al. 2011, Sing 2010).

We selected one programming language as a network boundary. Therefore, our data is restricted to projects using the same programming language. Future research can collect data for multiple programming languages.

We did not analyze the performance of exploitative and exploratory teams at the activity level. Future research can analyze the performance of exploitative and exploratory teams at the activity level.

CHAPTER 4: TEAM PERFORMANCE IN OPEN SOURCE SOFTWARE NETWORKS: THE EFFECT OF AMBIDEXTERITY ON THE PROJECT PERFORMANCE

4.1. Introduction

Traditionally, software has been developed by organizations that do not make the source code of software publicly available. In the traditional software development, software developers have worked in local clusters of collaboration that were generally isolated within firms (Fleming and Marx 2006). More recently, open source software (OSS) development has become the alternative way of developing software. OSS development has brought together software developers spanning firm boundaries (Raymond 1999). OSS development mainly depends on voluntary contributions of software developers and OSS products are developed in a collective manner beyond the boundaries of a single organization (Raymond 1999). Thus, formerly isolated software developers have become large connected networks in OSS development. The network of software developers becomes more important for OSS projects and offers various benefits. First, collaboration among software developers can facilitate access to and sharing of resources, allowing developers to combine their knowledge, skills, and expertise (Raymond 1999). Second, new insights, ideas or ways to solve problems are conceived by any one and accessed by others (Raymond 1999).

Thus, OSS development has changed the conception of how software can be developed. However, not all software projects are completed successfully (Li et al. 2010). Understanding the factors that lead to successful OSS projects is an interesting area of current research. OSS development offers new research opportunities to better understand the network structure of OSS developers.

Software product after delivery is improved by correcting faults or enhanced by adding new features based on user requirements (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). The total cost of software maintenance is estimated to comprise at least 50% of total software life cycle costs (Van Vliet 2000, Kemerer and Slaughter 1999, Kemerer 1995). Thus, the modification of software after delivery is one of the major phases of software development. In software maintenance, we identified two important types of OSS project activities: patch development and feature request. Patch development activities are used to correct faults in software while feature request activities are used to enhance software by adding new features. Recent research on OSS development focused on the analysis of OSS requirements and used feature request activities in their analysis (Vlas and Robinson 2012). Software is defined as a knowledge product (Slaughter et al. 2006) and critical inputs to software development are skills and experience of developers (Li et al. 2010). Therefore, each activity requires different structure of collaboration and knowledge sharing among the developers since each activity has different objectives.

In an organizational context, exploitation and exploration have been identified as two types of activities for the development and use of knowledge in organizations (March 1991). Prior research indicates that different types of tasks require different

communication patterns and different amount of communication based on characteristics and nature of a task (Katz and Tushman 1979). A task can differ along several dimensions including time span, specific vs. general problem orientation, and the generation of new knowledge vs. using existing knowledge (Katz and Tushman 1979). March (1991) has suggested that exploitation and exploration represent fundamentally incompatible and inconsistent activities. For example, exploitation represents activities that improve existing organizational competencies and build on the existing technological trajectory. Therefore, exploitation broadens existing knowledge and skills, improves established designs, and expands existing products and services. In contrast, exploration represents activities that changes the organizational competencies and build on a different technological trajectory. Therefore, exploration requires new knowledge, offers new designs, and creates new products and services. In addition, exploitation is related to efficiency, centralization, and tight cultures while exploration is associated with flexibility, decentralization, and loose cultures (Benner and Tushman 2003). Therefore, exploitation and exploration require different organizational structures (Benner and Tushman 2003, Levinthal and March 1993). Different organizational structures for exploitation and exploration enable exploitative teams to develop the best viable solutions, and enable exploratory teams to explore new ideas (Fang et al. 2010).

Recent research on OSS development has studied the social network structure of software developers as determinant of project success (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006). However, this stream of research has focused on the project level, and has not recognized the fact that projects could consist of different types of activities, each of which could require different types of expertise and network

structures. We propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration) based on organizational theory (March 1991). In the context of OSS development, developing a patch would be an example of an exploitation activity. Requesting a new software feature would be an example of an exploration activity.

While exploitation and exploration represent fundamentally incompatible and inconsistent activities (March 1991), recent research on organizational literature has stressed the importance of a balance between exploitation and exploration for organizational survival (Benner and Tushman 2003, Tushman and O'Reilly 1996). Structural differentiation is a proposed mechanism for organizations to build an ambidextrous organization (Benner and Tushman 2003, Tushman and O'Reilly 1996). Structural differentiation refers to the subdivision of organizational tasks into distinct organizational units that develop appropriate contexts for exploitation and exploration activities. Recent studies found that ambidextrous organizations perform better (Fang et al. 2010, Jansen et al. 2009, Jansen et al. 2006, Lin at al. 2007). However, the coordination and integration of exploitative and exploratory activities is a necessary step in achieving ambidexterity (Jansen et al. 2009, Gilbert 2006, Smith and Tushman 2005, Tushman and O'Reilly 1996). We identified a new category of developers (ambidextrous developers) in OSS projects who contribute to exploitative activities (patch development) and exploratory activities (feature request). We propose that ambidextrous developers are an integration mechanism between patch development and feature request activities. We develop a theoretical construct for project ambidexterity based on the concept of ambidextrous developers. We construct ambidexterity as a measure of the ability of OSS

projects to pursue both exploitative and exploratory activities concurrently. To the best of our knowledge, this is the first research to study ambidexterity and ambidextrous developers in OSS development.

In this dissertation, we introduce the use of organizational theory on ambidexterity together with social network analysis as a theoretical lens to study OSS project performance. We studied the effects of ambidexterity and coordination mechanisms (ambidextrous developers) on OSS project performance. We also studied the effects of social network properties of OSS developers on OSS project performance. We used a data set collected from the SourceForge database. We empirically illustrate the significance of ambidexterity and network characteristics on OSS project performance. We illustrate that a moderate level of ambidexterity, external cohesion, and technological diversity are desirable for project success.

4.2. Literature Review

There are multiple streams of research that help us to understand the structural differences of OSS networks. A software product after delivery is improved by correcting faults or enhanced by adding new features based on user requirements (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). Thus, in software maintenance, we identified two important types of OSS project activities: patch development and feature request. Software is a knowledge product (Slaughter et al. 2006) and critical inputs to software development are skills and experience of developers (Li et al. 2010). Therefore, each activity requires different structure of collaboration and knowledge sharing among the developers since each activity has different objectives. Recent studies on social network literature indicated that network structures determine the structure of

collaboration and knowledge sharing among actors. Recent research on OSS development has focused on project success as the function of the social network structure of software developers (Singh et al. 2011, Singh 2010). They founded that OSS network structure affects project success.

In an organizational context, exploitation and exploration have been identified as two types of activities for the development and use of knowledge in organizations (March 1991). Prior research indicates that different types of tasks require different communication patterns and different amount of communication based on characteristics and nature of a task (Katz and Tushman 1979). March (1991) has suggested that exploitation and exploration represent fundamentally incompatible and inconsistent activities. Exploitation creates a narrow range of deeper solutions and more distinctive competences since exploitation results in the convergence of ideas (March 1991). In contrast, exploration creates a wide range of undeveloped new ideas and limited distinctive competence (March 1991). Therefore, exploitation and exploration require different organizational structures.

Recent research on organizational literature has argued that organizations need to become ambidextrous, and perform explorative and exploratory activities simultaneously in different organizational units (e.g., Benner and Tushman 2003, Tushman and O'Reilly 1996). Units that engage in exploitation build on existing knowledge and extend existing products and services. Units that engage in exploration pursue new knowledge and develop new products and services. Therefore, structural differentiation is a proposed mechanism for organizations to build an ambidextrous organization (Benner and Tushman 2003, Tushman and O'Reilly 1996). Structural differentiation refers to the

subdivision of organizational tasks into distinct organizational units that develop appropriate contexts for explorative and exploratory activities. However, the coordination and integration of explorative and exploratory activities is a necessary step in achieving ambidexterity (Jansen et al. 2009, Gilbert 2006, Smith and Tushman 2005, Tushman and O'Reilly 1996). Cross-functional interfaces have been proposed as an integration mechanism to enable knowledge exchange between exploitative and exploratory units (Jansen et al. 2009, Gupta and Govindarajan 2000).

4.2.1. Open Source Software Development

Software maintenance is defined as the modification of a software product after delivery to correct faults, to improve performance or other attributes, and to enhance the product by adapting it to a modified environment (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). Thus, a software product is improved by correcting faults or enhanced by adding new features based on user requirements.

Raymond (1999) indicated that the different nature of software development process for proprietary and OSS vendors leads to two fundamentally different software development styles: the cathedral model for proprietary vendors and the bazaar model for OSS vendors (Raymond 1999). Software development involves knowledge work and its most important resource is the specialized skills and expertise that a developer brings to the project development (Espinosa et al. 2007, Roberts et al. 2004, Faraj and Sproull 2000). Proprietary software is developed in a more closed environment and, hence, proprietary software development is characterized by a relatively strong control of design and implementation (Raymond 1999). In contrast, OSS vendors mainly depend on voluntary contributions of software developers and, hence, OSS products are developed

in the collective manner beyond the boundaries of a single organization (Raymond 1999). Therefore, OSS development depends on contributions and collaboration of volunteer software developers (Liu and Iyer 2007, Feller and Fitzgerald 2002). The network of developers becomes more important for OSS projects and offers various benefits. First, collaboration among software developers can facilitate access to and sharing of resources, allowing developers to combine their knowledge, skills, and expertise. Second, new insights, ideas or ways to solve problems are conceived by any one and accessed by others. This leads to increase the performance of developer teams to find a solution for developing patches or to add new features.

Given the benefits of voluntary contributions of software developers for OSS development, the impact of network structure of OSS developer network (Singh et al. 2011, Singh 2010, Grewal et al. 2006) and the formation of OSS developer teams (Hahn et al. 2008) have been intensively studied. Recent studies showed that the network structure of OSS developers significantly affects OSS project success (Singh et al. 2011, Singh 2010, Grewal et al. 2006).

4.2.2. Open Source Software Collaboration Network

In social network literature, an affiliation network is a special kind of network which depends on the affiliation between two groups (Wasserman and Faust 1994). Therefore, an affiliation network has two-modes. The first mode is a set of actors such as developers. The second mode is a set of events such as OSS projects to which the actors belong. The term affiliation refers to membership or participation to events. Therefore, actors are related to each other through their joint affiliation with or their co-membership to events. Events are also related to each other through common actor(s).

OSS software development is a community-based model which involves collaboration among software developers. OSS developers may work on multiple projects concurrently. Thus, OSS developers belong to multiple projects. A co-membership relationship exists between two developers if they work together on the same projects. Similarly, a relationship between two projects also exists if they share some developer(s). This kind of relationships between developers and projects can be represented by an affiliation network. In OSS network, actors are developers, and events are projects.

4.2.3. Ambidextrous Organization through Exploitation and Exploration Networks

March (1991) modeled two general situations involving the development and use of knowledge in organizations: the exploitation of old certainties and the exploration of new possibilities. The first is the case of mutual learning between members of an organization. The second is the case of learning and competitive advantage in competition for primacy. Exploitation includes things such as refinement, choice, production, efficiency, selection, implementation, and execution (March 1991). In contrary, exploration includes things captured by terms such as search, variation, risk taking, experimentation, play, flexibility, discovery, and innovation (March 1991). According to Benner and Tushman (2003), exploitation represents activities that involve improvements in existing components and build on the existing technological trajectory. Exploitation is incremental innovations and designed to meet the needs of existing customers or markets (Benner and Tushman 2003). It broadens existing knowledge and skills, improves established designs, and expands existing products and services. Hence, exploitation builds on existing knowledge and reinforces existing skills, processes, and structures (Benner and Tushman 2002, Levinthal and March 1993, Lewin et al. 1999). In

contrast, exploration represents activities that involve a shift to a different technological trajectory and changes the organizational competencies. Exploration is radical innovations and designed to meet the needs of emerging customers or markets (Benner and Tushman 2003). It offers new designs, creates new markets. Thus, exploration requires new knowledge or departures from existing knowledge (Benner and Tushman 2002, Levinthal and March 1993).

For March (1991), exploitation and exploration represent the fundamentally incompatible and inconsistent activities. Exploitation creates a narrow range of deeper solutions and more distinctive competences in the short-run, which comes at the cost of long-term performance since exploitation results in the convergence of ideas by eliminating the differences (March 1991). In contrary, exploration creates a wide range of undeveloped new ideas and too little distinctive competence in the long-term, which comes at the cost of short-term performance (March 1991). Moreover, exploitation is related to efficiency, centralization, and tight cultures while exploration is associated with flexibility, decentralization, and loose cultures, (Benner and Tushman 2003).

While exploration and exploitation represent two fundamentally different approaches to organizational learning, recent studies on organizational literature have stressed the importance of a balance between exploitation and exploration for organizational survival (Benner and Tushman 2003, Siggelkow and Levinthal 2003, O'Reilly and Tushman 2004, Tushman and O'Reilly 1996, Levinthal and March 1993). This view is supported by research on absorptive capacity which argues that although internal knowledge processing and external knowledge acquisition are both necessary, excessive dominance by one or the other will be dysfunctional (Cohen and Levinthal

1990). Consistent with March (1991)'s model of organizational learning, Narayanan et al. (2009) examined the impacts of task specialization and task variety on the performance of software maintenance teams. Task specialization is defined as the cumulative experience at a specific task while task variety is defined as exposure to experience that is dispersed across distinct tasks (Narayanan et al. 2009). They indicate that task specialization and task variety affect the performance of software maintenance teams through different mechanisms. However, task specialization and task variety jointly drive the performance of software maintenance teams, thereby achieving a proper balance between task specialization and task variety leads to the highest performance (Narayanan et al. 2009).

The balanced view of exploitation and exploration is embedded in the concept of ambidextrous organizations. Ambidextrous organizations are composed of structurally differentiated exploitative and exploratory units (Benner and Tushman 2003, Siggelkow and Levinthal 2003, O'Reilly and Tushman 2004, Tushman and O'Reilly 1996, Levinthal and March 1993). Structural differentiation is a proposed mechanism for organizations to build an ambidextrous organization (Benner and Tushman 2003, Tushman and O'Reilly 1996). Structural differentiation refers to "the state of segmentation of the organizational system into subsystems, each of which tends to develop particular attributes in relation to the requirements posed by its relevant external environment" (Lawrence and Lorsch 1967). In other words, structural differentiation refers to the subdivision of organizational tasks into distinct organizational units that develop appropriate contexts for exploitation and exploration activities. Structural differentiation establishes differences across organizational units in terms of mindsets, time orientations, functions, and

product/market domains (Lawrence and Lorsch 1967). Structural differentiation can help ambidextrous organizations to maintain multiple competencies that address paradoxical demands (Gilbert 2005). Structural differentiation protects ongoing operations in exploitative units from interfering with emerging competences being developed in exploratory units (Jansen et al. 2009). Therefore, exploitation and exploration activities can be achieved without corrupting the internal structures and processes within each unit's area of operation. Distinct organizational units can develop the best viable solutions (i.e., exploitation), while still ensuring to explore new ideas (i.e., exploration) (Fang et al. 2010). In this approach, organizational units pursuing exploration are smaller, more decentralized, and more flexible than those responsible for exploitation (Benner and Tushman 2003, Christensen 1998, Tushman and O'Reilly 1996). Therefore, structural differentiation helps ambidextrous organizations to be capable of simultaneously exploiting existing competencies and exploring new opportunities (Raisch et al. 2009).

Jansen et al. (2009) recognize organizational ambidexterity as a dynamic capability that refers to the routines and processes by which ambidextrous organizations mobilize, coordinate, and integrate contradictory efforts, and allocate, reallocate, combine, and recombine resources and assets across differentiated exploitative and exploratory units. Although the structural differentiation of exploitative and exploratory activities is important to achieve organizational ambidexterity, ambidextrous organizations also need to facilitate collective action (O'Reilly and Tushman 2004, Jansen et al. 2009). This view is supported by O'Reilly and Tushman (2007) who argue that the crucial task is not the simple organizational structural design in which exploitative and exploratory units are separated, but the processes through which these

units are integrated in a value enhancing way. The structural differentiation of exploitation and exploration activities may lead to distinct organizational capabilities and competences (March 1991, Gilbert 2006). However, distinct capabilities and competences developed within each unit must be effectively allocated, mobilized, and integrated to generate new combinations (Sirmon et al. 2007). Therefore, the coordination and integration of exploitative and exploratory units is a necessary step in achieving ambidexterity (Gilbert 2006, Tushman and O'Reilly 1996, Jansen et al. 2009). Recent research on organizational literature recognizes different types of integration mechanisms such as cross-functional interfaces (Jansen et al. 2009, Jansen et al. 2006, Lawrence and Lorsch 1967). Cross-functional interfaces are the cross-functional team of common organizational members from both exploitative and exploratory units (Jansen et al. 2009).

In the similar concept, Koza and Lewin (1998) extended March's (1991) concepts into the strategic alliance literature to explore the balance between exploitative and explorative alliances. For example, exploitative alliances are built on a firm's aim to leverage existing capabilities and competencies (Rothaermel and Deeds 2004). However, exploratory alliances are built on a firm's desire to discover new opportunities, build new competencies, and adapt to environmental changes (Koza and Lewin 1998). The concept of ambidexterity in alliance formation has been conceptualized in several ways. Lavie and Rosenkopf (2006) identified three dimensions of ambidexterity: function-based, structure-based, and attribute based dimensions. Structure based dimension based on the network structure. For example, Lin et al. (2007) measured ambidexterity based on the network structure of alliances.

4.2.4. Social Network and Team Structure

Software development is a highly interdependent task and requires team members to interact with each other intensively to produce a successful system (He et al. 2007). Therefore, interactions among team members are necessary activities to transform team members' knowledge to team knowledge that increase the project success (He et al. 2007). However, the nature of OSS development characterized by volunteer contribution of software developers poses challenges in coordination among developers (Espinosa et al. 2007, Roberts et al. 2004, Banker et al. 2006). Coordination is the process of managing dependencies among activities (Malone and Crowston 1994). When the activities of multiple individuals need to interrelate, the interdependencies among activities should be well managed (Espinosa et al. 2007). Espinosa et al. (2007) indicated that when software is produced from multiple locations, it becomes more difficult to manage dependencies among activities and to coordinate developers, which increases the development time. Therefore, the coordination among developers becomes important for project success in software development.

He et al. (2007) created a model of the formation and evolution of team cognition and analyzed the impacts of preexistent and ongoing collaboration ties on the formation of team cognition in software project teams. Team cognition refers to the mental models collectively held by a group of individuals that enable them to accomplish tasks by acting as a coordinated unit (He et al. 2007). Team cognition helps software project teams effectively manage their members' knowledge, expertise, and skills as integrated assets (He et al. 2007, Espinosa et al. 2007). Team cognition is created by both preexisting conditions and ongoing team interactions. Preexisting conditions reflect both the prior

knowledge of team members and any previous shared experiences that team members have. Team interactions refer to the interactive activities that members perform to carry out project tasks and facilitate team performance. He et al. (2007) showed that the positive relationship between team performance and team cognition. Similarly, Hahn et al. (2008) studied the impact of prior collaboration ties on OSS collaboration team formation mechanisms and on OSS project success. They indicated that team cohesion is related to preference for repeat collaborations and results from prior relationships between developers to benefit from prior relationships. Team members also tend to interact more frequently with other members with whom they share some type of proximity or similarity (Rosenkopf and Almeida 2003, Rosenkopf and Nerkar 2001).

In social network literature, social capital is defined as resources embedded in social networks, and resources that can be accessed or mobilized through social ties in the networks (Coleman 1988, Lin 2005). Through social ties, an actor may capture other actors' resources. These social resources can generate a return for the actor. In addition, because of the facilitative role of network structure, relationships among actors in a network are described as network resources (Gulati 1999). Recent studies also indicated that the position of a team in a network affects team outcomes (Singh et al. 2011, Singh 2010, Zaheer and Bell 2005, Reagans and Zuckerman 2001, Jansen et al. 2006, Schilling and Phelps 2007, Rosenkopf and Almeida 2003, Rosenkopf and Nerkar 2001).

In social network literature, there are two contradictory perspectives about the form of network structures: the internal focus or social closure perspective (Coleman 1988) and the external focus or structural holes perspective (Burt 1992). From Coleman (1988)'s social closure perspective, the optimal social structure is one generated by

building dense, interconnected networks. Social closure inside a group indicates the presence of relationships or the absence of structural holes within a group, and is thought to foster identification with the group (Reagans and Zuckerman 2001) and a level of mutual trust, which facilitates exchange and collective action (Coleman 1988). Social closure enables the convergence of individual interests to pursuit common initiatives and to facilitate mutual coordination (Reagans and Zuckerman 2001). From Burt (1992)'s structural holes perspective, constructing networks consisting of disconnected alters is the optimal strategy. Structural holes perspective focuses value derived from bridging gaps (i.e., structural holes) between nodes in a social network (Burt 1992). This boundary spanning structure generates information benefits since information tends to be relatively redundant within a given group (Burt 1992). As a result, actors who develop ties with disconnected groups gain access to a broader range of ideas and opportunities than those who have restricted access to single group (Granovetter 1973). Although prior research on social network analysis indicated the trade-off between two contradictory perspectives, these two perspectives do not conflict with one another (Reagans and Zuckerman 2001). While the social closure perspective highlights the importance of the presence of relationships in local interactions (i.e., internal cohesion), the external focus perspective highlights information benefits created by structural holes that divide a social network globally (i.e., external cohesion).

Ahuja (2000) studied the impact of social network structures on innovation in terms of direct ties, indirect ties, and structural holes. The debate on structural holes suggests that an accurate understanding of the role of structural holes in the collaboration network must account for both Coleman's and Burt's variants of the argument (Ahuja

2000). Similarly, direct and indirect ties may vary in their content, which highlights the importance of decomposing the firm's ego network into distinct and separate elements and identifying the contents transmitted through each type of tie (Ahuja 2000). According to Ahuja (2000), network ties are associated with two distinct kinds of network benefits. First, they can provide the benefit of resource sharing, allowing firms to combine knowledge, and skills. Second, collaborative linkages can provide access to knowledge spillovers, serving as information conduits through which news of technical breakthroughs, new insights to problems, or failed approaches travels from one firm to another. In distinguishing between the resource-sharing and knowledge-spillover benefits of collaboration, it is important to distinguish between know-how and information (Kogut and Zander 1992). Know-how entails accumulated skills and expertise in some activity. Information refers primarily to facts that can be transmitted through communication (Kogut and Zander 1992, Szulanski 1996). The resource-sharing benefits of collaboration relate primarily to the transfer and sharing of know-how while the knowledge-spillover benefits are likely to involve predominantly information. Ahuja (2000) found that direct and indirect ties both have a positive impact on innovation but that the impact of indirect ties is moderated by the number of a firm's direct ties. Direct ties potentially provide both resource sharing and knowledge spillover benefits. However, indirect ties do not entail formal resource sharing benefits but can provide access to knowledge spillovers. Structural holes influence both resource sharing and access to novel information (Ahuja 2000). Structural holes have both positive and negative influences on innovation. Specifically, increasing structural holes has a negative effect on innovation, so the optimal structure of networks depends on the objectives of the network members.

Zaheer and Bell (2005) examined the impact of the network structure on the performance and innovativeness of companies by focusing on the external connectivity constructed as structural holes. They highlight the importance of connections to external sources for innovativeness. Zaheer and Bell (2005) found that firms bridging structural holes are more innovative and perform better than other firms. They also indicated that the internal connectiveness enables firms to further exploit the ideas obtained from external resources.

Jansen et al. (2006) focused on the differences of exploration and exploitation, and examined the impact of internal cohesion and centralization on exploitation and exploration. They found that internal connectedness within teams positively affects the performance of exploitation and exploration teams while centralization negatively affects exploration teams. However, Balkundi and Harrison (2006) indicated that teams that are central in their inter-group network tend to perform better.

Schilling and Phelps (2007) examined the impact of clustering on the innovative output of firms that are members of the network. Innovation is characterized as a process in which solutions are discovered via search process that leads to the creation of new knowledge or the novel recombination of known elements of knowledge, problems, or solutions (Fleming 2001). Schilling and Phelps (2007) indicated the positive association between clustering and innovation output.

Rosenkopf and Nerkar (2001) studied the impact of organization and technology domain on subsequent technological development. They stressed the importance of knowledge internally acquired from the similar technology domains on exploitation, and the importance of knowledge externally acquired from the distinct technology domains

on exploration. In other words, organizations can develop more distinctive competence and becomes more expert in their current domain if they focus on their current organizational domain and the similar technological areas. Distinctive competences can improve the performance of developer teams on exploitation (March 1991, Rosenkopf and Nerkar 2001). In contrast, organizations can develop more diverse and less distinctive competence if they focus on their external organizational domain and the distinct technological areas. More diverse and less distinctive competence can improve the performance of developer teams on exploration (March 1991, Rosenkopf and Nerkar 2001). Lazer and Friedman (2007) on their agent-based simulation model of information sharing found that a network that maintains diversity is better for exploration than other networks, supporting a more thorough search for solutions in the long run.

Social network analysis (Wasserman and Faust 1994) has been used in a variety of contexts to study the relationship between social entities. Based on the findings of social network research (Gnyawali and Madhavan 2001, Ahuja 2000, Uzzi 1999, Uzzi 1997, Uzzi 1996, Watts and Strogatz 1998, Krackhardt 1998, Wasserman and Faust 1994, Burt 1992, Coleman 1988, Freeman, 1979, Granovetter 1973), organizational research (Schilling and Phelps 2007, Hansen 2002, Hansen 1999, Reagans and Zuckerman 2001), and OSS development research (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006), structural properties of the networks are used to analyze the network. Many structural properties of these networks could have multiple social network measures. For example, there are different types of internal cohesion measures (clustering coefficient, repeat ties, third party ties, and structural equivalence), external connectivity measures (external cohesion, direct ties, indirect ties, and technological

diversity), and network location measures (degree centrality, betweenness centrality, and closeness centrality).

4.3. Theoretical Background and Hypotheses

In software development literature, software product after delivery is improved by correcting faults or enhanced by adding new features based on user requirements (Banker and Slaughter 2000, Banker et al. 1998, IEEE 1983). Therefore, we identified two types of OSS project activities: patch development and feature request. Patch development activities are used to correct faults in software while feature request activities are used to enhance software by adding new features. In organizational literature, exploitation and exploration have been identified as two types of activities for the development and use of knowledge in organizations (March 1991). Combining findings of organizational literature and software development literature, we propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration). In the context of OSS development, developing a patch would be an example of an exploitation activity. Requesting a new software feature would be an example of an exploration activity.

Prior research indicates that different types of tasks require different communication patterns and different amount of communication based on characteristics and nature of a task (Katz and Tushman 1979). A task can differ along several dimensions including time span, specific vs. general problem orientation, and the generation of new knowledge vs. using existing knowledge (Katz and Tushman 1979). This is consistent with the view of March (1991) who has suggested that exploitation and exploration represent fundamentally incompatible and inconsistent activities. For

example, exploitation represents activities that improve existing organizational competencies and build on the existing technological trajectory. Therefore, exploitation broadens existing knowledge and skills, improves established designs, and expands existing products and services. In contrast, exploration represents activities that changes the organizational competencies and build on a different technological trajectory. Therefore, exploration requires new knowledge, offers new designs, and creates new products and services. In addition, exploitation is related to efficiency, centralization, and tight cultures while exploration is associated with flexibility, decentralization, and loose cultures (Benner and Tushman 2003). Therefore, exploitation and exploration require different organizational structures (Benner and Tushman 2003, Levinthal and March 1993). Different organizational structures for exploitation and exploration enable exploitative teams to develop the best viable solutions, and enable exploratory teams to explore new ideas (Fang et al. 2010). In addition, software is a knowledge product (Slaughter et al. 2006) and critical inputs to the software development are skills and experience of developers (Li et al. 2010). Therefore, each project activity could require different types of expertise and network structures.

While exploitation and exploration are fundamentally incompatible and inconsistent activities (March 1991), recent research on organizational literature has stressed the importance of a balance between exploitation and exploration for organizational survival (Benner and Tushman 2003, Tushman and O'Reilly 1996). The balance between exploitation and exploration is embedded in the concept of ambidextrous organizations. Ambidextrous organizations are composed of structurally differentiated exploitative and exploratory units (Benner and Tushman 2003, Siggelkow

and Levinthal 2003, O'Reilly and Tushman 2004, Tushman and O'Reilly 1996, Levinthal and March 1993). However, the coordination and integration of exploitative and exploratory activities is a necessary step in achieving ambidexterity (Jansen et al. 2009, Gilbert 2006, Smith and Tushman 2005, Tushman and O'Reilly 1996). Ambidextrous organizations may use cross-functional interfaces (Jansen et al. 2009, Gupta and Govindarajan 2000) such as ambidextrous developers as an integration mechanism between exploitative activities (patch development) and exploratory activities (feature request). In this dissertation, we studied the effects of ambidexterity and coordination mechanisms (ambidextrous developers) on OSS project performance. We also studied the effects of social network properties of OSS developers on OSS project performance.

4.3.1. Ambidexterity

Structural differentiation is a proposed mechanism for organizations to build an ambidextrous organization (Benner and Tushman 2003, Tushman and O'Reilly 1996). Structural differentiation establishes differences across organizational units in terms of mindsets, time orientations, functions, and product/market domains (Lawrence and Lorsch 1967). Therefore, structural differentiation can help project teams to develop and maintain different competencies that are required for patch development and feature request activities (Gilbert 2005). Structural differentiation protects ongoing operations in patch development activities from interfering with emerging competences being developed in feature request activities. Patch development and feature request activities can be achieved without corrupting the internal structures and processes within each unit's area of operation. The structural differentiation of patch development and feature request activities may lead to distinct organizational capabilities and competences within

each unit (March 1991, Gilbert 2006). However, these differentiated competences must be effectively allocated, mobilized, coordinated, and integrated to achieve ambidexterity (Sirmon et al. 2007). The important task in building ambidextrous organization is not the simple organizational structural design in which patch development and feature request activities are separated, but the processes by which these units are integrated in a value enhancing way. Therefore, the coordination and integration of patch development and feature request activities is a necessary step to achieve ambidexterity (Gilbert 2006, Tushman and O'Reilly 1996, Jansen et al. 2009).

We identified a new category of developers (ambidextrous developers) in OSS projects who contribute to exploitative activities (patch development) and exploratory activities (feature request). We propose that ambidextrous developers are an integration mechanism between patch development and feature request activities. Ambidextrous developers facilitate knowledge exchange and combination between patch development and feature request activities (Kogut and Zander 1992, Jansen et al. 2009). Through combination and integration of differentiated skills and experiences, project teams are able to synchronize, maintain, and further build portfolios of patch development and feature request activities simultaneously (Tushman et al. 2006). Ambidextrous developers facilitate new value creation through linking knowledge developed by patch development and feature request teams (Cohen and Levinthal 1990). They also provide opportunities to leverage common resources and obtaining synergies across patch development and feature request activities (O'Reilly and Tushman 2007). In this way, knowledge developed by patch development teams may be revisited, reinterpreted, and applied in feature request teams, or vice versa (Garud and Nayyar 1994, Postrel 2002).

Ambidextrous developers facilitate other team members to reach a common frame of reference and to build understanding and agreement (Daft and Lengel 1986, Egelhoff 1991). Ambidextrous developers also resolve differences across patch development and feature request teams to overcome disagreement over organizational goals (Daft and Lengel 1986). We develop a theoretical construct for project ambidexterity based on the concept of ambidextrous developers. We construct ambidexterity as a measure of the ability of OSS projects to pursue both patch development and feature request activities concurrently.

Consistent with March (1991)'s model of organizational learning, Narayanan et al. (2009) examined the impacts of task specialization and task variety on the performance of software maintenance teams. Task specialization and task variety represent fundamentally contradictory perspectives. Task specialization refers to gaining cumulative experience from a specific task (Narayanan et al. 2009). In contrast, task variety refers to gaining diverse experience from a variety of different tasks (Narayanan et al. 2009). Consistent with Narayanan et al. (2009), we argue that ambidextrous developers are project developers who gain diverse experience from a variety of different tasks since they work on both patch development and feature request activities concurrently. In contrast, we argue that non-ambidextrous developers are project developers who are specialized in a specific task (either patch development or feature request activities).

Project teams can gain more and deeper experience from specializing in one task because the task becomes more routine and developers become more familiar with the task (Narayanan et al. 2009). Specialized experience can improve project teams' ability to

understand, enhance, and modify the source code through different mechanisms (Narayanan et al. 2009). First, the experience gained from previous tasks is transferred to perform the current task. Second, the experience gained from previous tasks is applied to make further adjustments in the way of performing the current task. Third, the experience gained from previous tasks enables project teams to better learn from the current task. Therefore, the higher level of experience gained from a focused task increases project performance.

Project teams can gain diverse knowledge from different types of tasks (Narayanan et al. 2009). First, diverse knowledge can improve project teams' ability to better delineate knowledge that is more relevant to the current task from knowledge that is less relevant (Narayanan et al. 2009). Therefore, it prevents situations in which project teams spend time and effort in mastering new knowledge that is not really useful to the current task. For example, project teams can better understand the various patterns of software elements, and the interdependency and relationships of software elements. It may provide project teams with a better appreciation of the software product itself and the functionality of software elements. Therefore, project teams can make more informed inferences regarding the source code with limited examination of the specific software elements that is being worked on. Second, diverse knowledge allows project teams to make correlations between tasks, and then apply them to solve a broader range of problems (Narayanan et al. 2009). For example, when project teams work across different tasks, they can develop rules regarding how to solve problems with underlying common remedies. This knowledge may enable them to use preexisting solutions to known

problems. Furthermore, exposure to other tasks can help them better anticipate and avoid problems when working within a given task.

Task specification and task variety represent the fundamentally contradictory perspectives. Task specification is associated with gaining cumulative experience from a specific task (Narayanan et al. 2009). In contrast, task variety is associated with gaining diverse experience from a variety of different tasks (Narayanan et al. 2009). Therefore, there is a trade-off between task specialization and task variety. However, Narayanan et al. (2009) indicated that task specialization and task variety jointly drive the performance of software maintenance teams, and achieving a proper balance between task specialization and task variety leads to the highest performance. Excessive exposure to task variety without adequate opportunity to specialize can lead to a lot of shallow learning that ultimately does not enhance the performance (Narayanan et al. 2009). In contrast, overspecialization on a small set of tasks can reduce the ability of project teams to absorb and integrate new knowledge that will ultimately lead to higher performance (Narayanan et al. 2009). Therefore, we argue that ambidextrous developers have access to diverse knowledge from a different task types, and exchange and integrate greater amounts of knowledge among other project developers. On the other hand, non-ambidextrous developers specialize in a specific type of task, and they may benefit from knowledge exchanged by ambidextrous developers applying to a specific task type. We argue that a moderate level of ambidexterity enables project teams to access diverse knowledge from different types of tasks, and to exchange relevant knowledge within a project team, while ensuring adequate specialization to absorb and integrate new knowledge. This leads us to the following hypothesis:

H1: A moderate level of ambidexterity results in higher project performance rather than very high or very low levels of ambidexterity.

4.3.2. Internal Cohesion

OSS development mainly depends on voluntary contributions of software developers and OSS products are developed in the collective manner (Raymond 1999). OSS development process is characterized by the lack of a relatively strong control of design and implementation (Raymond 1999) and the lack of face-to-face communication (Singh et al. 2011). Therefore, OSS teams require constructive environment to foster trust, reciprocity norms and shared identity, and to improve collaboration and cooperation among developers (Singh et al. 2011).

Internal cohesion increases the information transmission capacity of a team (Schilling and Phelps 2007). First, internal cohesion improves access to information since the same information is available via multiple paths (Schilling and Phelps 2007). Information introduced into a team will quickly reach other team members through multiple paths. Multiple paths also enhance the fidelity of information received. Developers can compare information received from multiple partners, helping them to identify whether it is distorted or incomplete (Schilling and Phelps 2007). Second, internal cohesion makes information exchange meaningful and useful (Schilling and Phelps 2007). It can increase the dissemination of alternative interpretations of problems and their potential solutions, deepening the shared understanding and stimulating collective problem solving. Shared knowledge develops over time from prior familiarity with the product being developed and team members (Espinosa et al. 2007, He et al. 2007). Shared knowledge improves coordination among team members because it

enables team members to develop more accurate explanations and expectations about tasks and other team members (Espinosa et al. 2007) because prior interactions enable developers to acquire information about skills and capabilities of other developers (Granovetter 1985) and who knows what (Faraj and Sproull 2000). In addition, shared knowledge of problems and solutions enhances further learning (Schilling and Phelps 2007). Third, internal cohesion can make developers more willing and able to improve information exchange and cooperation among team members by fostering trust, reciprocity norms, and shared identity (Coleman 1988, Uzzi and Spiro 2005, Adler and Kwon 2002, Levin and Cross 2004, Hansen 1999, Ahuja 2000). Enhanced trust, reciprocity norms, and shared identity results in a high level of cooperation and collaboration by providing self-enforcing informal governance mechanisms (Schilling and Phelps 2007). Fourth, internal cohesion fosters group identification which enables the convergence of individual interests to pursuit common initiatives and to facilitate mutual coordination (Reagans and Zuckerman 2001). Fifth, internal cohesion also helps developers to develop team cognition which promote team coordination (Espinosa et al. 2007, He et al. 2007). Team cognition refers to the mental models collectively held by a group of individuals that enable them to accomplish tasks by acting as a coordinated unit (He et al. 2007). Thus, team cognition helps developer teams effectively manage team members' knowledge, expertise, and skills as integrated assets (He et al. 2007, Espinosa et al. 2007).

Internal cohesion results in a high level of cooperation and collaboration among team members. By improving the information transmission capacity of a team, it also enables to exchange and integrate greater amounts of information and knowledge more

rapidly. Internal cohesion allows project developers to develop a deep understanding to further refine and improve existing products, and processes (Rowley et al. 2000). Therefore, we argue that internal cohesion is positively related to the performance of a project[11]. This leads us to the following hypothesis:

H2: The performance of a project will be positively related to the internal cohesion of a project.

4.3.3. External Connectivity

Although the internal cohesion of a project team provides various benefits in terms of trust and information transmission capacity, project developers have access to external resources from external relationships to other developers outside of a project team. The structure and type of these relationships affect the ability of project developers to acquire various types of information that potentially affect the success of a project (Singh et al. 2011). By following prior research, we focus on the external network structure (the cohesion of external connections), types of external connections (direct ties and indirect ties) and technological characteristics of external connections that affect the diversity of external knowledge available to a focal project.

External connections are associated with two distinct kinds of information benefits (Ahuja 2000). First, they can provide the benefit of resource sharing which allows teams to combine knowledge, and skills acquired from outside teams. Second, they can provide access to knowledge spillovers which serves as information conduits through which news of technical breakthroughs, new insights to problems, or failed approaches acquired from outside project teams. Although direct ties potentially provide

[11] Prior research hypothesized that a moderate level of internal cohesion results in higher project performance rather than very high or very low levels of internal cohesion (Singh et al. 2011). However, their results did not prove it.

both resource sharing and knowledge spillover benefits (Ahuja 2000), they more likely provide redundant information (Hansen 1999). However, indirect ties do not provide resource sharing benefits but can provide access to knowledge spillovers. Therefore, information provided by indirect ties is novel information (Hansen 1999). On the other hand, external cohesion provides both resource sharing and knowledge spillovers benefits (Ahuja 2000). Although how external contacts are connected with each other affects types of information, the characteristics of the external contacts may also affect the diversity of knowledge. External contacts with different technological expertise are more likely to provide novel information and knowledge.

4.3.3.1. External Cohesion

External cohesion is the cohesion among the external contacts of a project (Singh et al. 2011). External cohesion is based on the idea of a structural hole which means the absence of a connection between two developers who are connected to the common third parties. Therefore, structural holes are defined as gaps in information flows between actors connected to the same actor but not directly connected to each other (Burt 2000). A structural hole separates developers on either side of the hole and creates the brokerage opportunities for those developers to obtain information from disconnected developers (Burt 1992). Therefore, structural holes provide both resource sharing and knowledge spillovers benefits (Granovetter 1973).

External cohesion basically measures the extent to which external contacts of a project are connected to each other. If external contacts of a project are highly connected with each other (high external cohesion or low structural holes), a project is highly constrained to have access to novel information since too much cohesion results in

homogenization of information and external contacts of a project may have relatively redundant information (Burt 2004, Burt 1992, Granovetter 1973). However, high external cohesion also enhances trust, reciprocity norms, and shared identity (Coleman 1988, Uzzi and Spiro 2005, Adler and Kwon 2002, Levin and Cross 2004). High external cohesion also improves access to external resources by enhancing information transmission capacity of the network since the same information is available via multiple paths (Schilling and Phelps 2007). Multiple paths also enhance the fidelity of the information received (Schilling and Phelps 2007). In contrast, if external contacts of a project are not connected with each other (low external cohesion or high structural holes), a project have access to novel information from remote parts of the network such as other disconnected project groups (Burt 1992). Therefore, the level of cohesion among the external contacts of a project determines the diversity of knowledge acquired from external contacts.

OSS network is made up of distinct developer teams in which developers are highly connected with each other within each project team, but weakly connected to other developers across other project teams (Singh 2010). Project teams tend to be heterogeneous across a network in terms of the knowledge they possess and produce because each team started with the different initial conditions (Fang et al. 2010). Therefore, external resources provide new knowledge, ideas, and insights (Rosenkopf and Almeida 2003).

Knowledge is developed through combinations of existing and new knowledge (Kogut and Zander 1992). The process of sharing ideas with other projects that have novel information is to generate new knowledge, rather than merely exchanging existing information (Nahapiet and Ghoshal 1998). This idea is consistent with the idea put forth

by March (1994) that projects connected to other projects that have novel information may replicate innovative ideas and generate more new ideas which can be used to introduce new and innovative products. A project whose external contacts are not highly connected has access to new knowledge, ideas, and insights from disconnected external projects (Burt 2004, Burt 1992) and they are able to develop new knowledge through knowledge recombination (Rosenkopf and Almeida 2003). Therefore, a project whose external contacts are not highly connected is able to develop new understandings not possible to those whose external contacts are highly connected (Zaheer and Bell 2005). Combining diverse knowledge from other projects (different technology areas) also enhances the capacity for creative learning (Fleming 2001, Kogut and Zander 1992, Reagans and Zuckerman 2001). Therefore, project teams that acquire knowledge from unique parts of their network improve their performance. This view is supported by Zaheer and Bell (2005) that actors bridging structural holes have been frequently shown to perform better than other actors not so positioned.

Aforementioned discussions indicate that the impact of external cohesion on resource sharing benefits is opposite to knowledge spillover benefits (Ahuja 2000). Resource sharing benefits arise from the sharing and combination of knowledge and skills acquired from outside project teams (Ahuja 2000, Uzzi 1997, Walker et al. 1997). The development of mutual trust and shared norms are preconditions for successful resource sharing (Coleman 1988, Uzzi and Spiro 2005, Adler and Kwon 2002, Levin and Cross 2004). Without mutual trust and shared norms, the sharing and combination of knowledge and skills are difficult and unproductive (Coleman 1988). High external cohesion enhances mutual trust and shared norms (Coleman 1988, Uzzi and Spiro 2005,

Adler and Kwon 2002, Levin and Cross 2004). High external cohesion also improves access to external resources by enhancing information transmission capacity of the network through multiple paths (Schilling and Phelps 2007). Multiple paths also enhance the fidelity of the information received (Schilling and Phelps 2007). However, high external cohesion limits the ability of a project to have access to novel information since too much cohesion results in homogenization of information and external contacts of a project may have relatively redundant information (Burt 2004, Burt 1992, Granovetter 1973). In contrast, knowledge spillover benefits arises from access to novel information in the forms of information conduits through which news of technical breakthroughs, new insights to problems, or failed approaches acquired from outside project teams (Ahuja 2000, Uzzi 1997, Walker et al. 1997). Low external cohesion enables a project to have access to novel information (Burt 1992), but reduces mutual trust and shared norms among the external contacts of a project, thereby hinder the transmission of knowledge (Coleman 1988). Therefore, there is a trade-off between resource sharing and knowledge spillover benefits of external cohesion. We argue that a moderate level of external cohesion enables project teams to develop mutual trust and shared norms, and to enable successful resource sharing, while ensuring access to relatively diverse knowledge from external resources which enables knowledge spillover. This leads us to the following hypothesis:

H3: A moderate level of external cohesion results in higher project performance rather than very high or very low levels of external cohesion.

4.3.3.2. Direct Ties

Direct ties in a social network potentially provide both resource sharing and knowledge spillover benefits (Ahuja 2000). First, direct ties enable knowledge sharing. When developers collaborate to develop a technology, the resultant knowledge is available to all developers. Thus, each developer can potentially receive a greater amount of knowledge from a collaborative activity than it would obtain from a comparable research investment made independently (Ahuja 2000). Second, collaboration facilitates bringing together complementary skills from different developers. By accessing to complementary skills from different developers, a project team can enhance their own knowledge base and improve their performance. In addition, direct ties among two developers imply opportunities for repeat interactions (Singh et al. 2011). Repeat interactions allow for resource pooling and joint problem solving (Kogut and Zander 1992). However, over time, repeated interactions using the same direct ties are more likely provide redundant information to a focal team (Hansen 1999). Hence, the knowledge spillover effect could decrease over time. Thus, the resource sharing benefit of direct ties is more likely greater than knowledge spillover benefit. Direct ties allow developers to combine knowledge and skills using repeating interactions (Kogut and Zander 1992). Repeated interactions through direct ties allow for resource pooling and joint problem solving (Kogut and Zander 1992) which do not decrease due to repeated interactions. Therefore, we argue that the number of direct ties is positively related to the performance of a project. This leads us to the following hypothesis:

H4: The performance of a project will be positively related to the number of direct ties of a project.

4.3.3.3. Indirect Ties

External connection can be a channel of communication between developers through indirect contacts (Ahuja 2000). An indirect tie between two developers exists when two developers do not work together but can be reached through mutual partners. Therefore, indirect ties provide developers with access not just to knowledge held by their immediate partners but also to knowledge held by their partner's partners (Gulati and Garguilo 1999). However, indirect ties are distant and infrequent relationships (Granovetter 1973). Therefore, they are less likely to provide opportunities for repeat interactions and they are not as conducive to resource pooling as direct ties (Singh et al. 2011). They provide access to novel information by bridging otherwise disconnected developers (Granovetter 1973). Indirect ties can provide access to knowledge spillovers (Ahuja 2000), serving as information conduits through which news of technical breakthroughs, new insights to problems, or failed approaches travels from one developer to another (Ahuja 2000). Information provided by indirect ties is more likely novel information (Hansen 1999). Organizations develop knowledge through combinations of existing and new knowledge (Kogut and Zander 1992). Novel information provided by indirect ties can be useful to develop knowledge through knowledge recombination (Rosenkopf and Almeida 2003). Combining diverse knowledge enhances the capacity for creative learning (Fleming 2001, Kogut and Zander 1992, Reagans and Zuckerman 2001). Therefore, we argue that the number of indirect ties is positively related to the performance of a project. This leads us to the following hypothesis:

H5: The performance of a project will be positively related to the number of indirect ties of a project.

4.3.3.4. Direct and Indirect Tie Interaction

The degree to which a focal team benefits from indirect ties is contingent on the number of direct ties of a focal team (Ahuja 2000). Project teams with few direct ties are more likely to have greater benefits from their indirect ties than project teams with many direct ties (Ahuja 2000). The relative addition to knowledge through indirect ties is greater for teams with few direct ties than for teams with many direct ties (Ahuja 2000). For teams with limited access to the network through direct ties, information provided by indirect ties may represent a significant increment to a focal team's existing information base. Therefore, we argue that project teams with many direct ties are more likely to add to less knowledge to their existing information base through their indirect ties than teams with few direct ties. This leads us to the following hypothesis:

H6: The impact of indirect ties on the performance of a project will be moderated by the number of direct ties of a project: the greater the number of direct ties, the smaller the benefit from indirect ties.

4.3.3.5. Technological Diversity

Although how external contacts are connected with each other affects types of information, the characteristics of external contacts may also affect the diversity of knowledge since they may vary in terms of technological areas (Rosenkopf and Nerkar 2001). External contacts in different technological areas are more likely to provide novel information and knowledge (Fleming 2001, Kogut and Zander 1992).

There is the trade-off between two contradictory perspectives in terms of the effect of diversity on team performance (Reagans and Zuckerman 2001). According to the first perspective, diversity provides creative learning benefits. In contract, according

to the second perspective, diversity creates coordination problems. Recent studies indicated that combining diverse knowledge from different technology areas has a positive impact on team performance (Fleming 2001, Kogut and Zander 1992). Therefore, teams which draw their members from different technological areas perform better since team members have different technical skills and expertise (Reagans and Zuckerman 2001). These teams enhance their capacity for creative learning since diverse ideas provide alternative ways of thinking, more options for creating new combinations which enhance both problem solving and innovation (Reagans and Zuckerman 2001). However, diversity introduces social divisions that hinder effective teamwork (Reagans and Zuckerman 2001) or create tensions among team members (Pfeffer 1983). Therefore, homogeneous teams may be expected to perform better since they can coordinate their members more easily than diverse teams (McCain et al. 1983, O'Reilly et al. 1989, Zenger and Lawrence 1989). However, the performance of homogeneous teams is restricted by relatively redundant information of team developers (Ancona and Caldwell 1992, Pelled et al. 1999). In addition, teams vary widely in their capability to develop, understand, or use knowledge based on their technological base and their prior knowledge (Cohen and Levinthal 1990). Absorptive capacity of teams reflects their ability to exploit novel knowledge (Zahra and George 2002) and determines their ability to utilize and benefit from novel and unfamiliar ideas (Cohen and Levinthal 1990). Teams can recognize and absorb knowledge close to their existing knowledge base (Cohen and Levinthal 1990). When teams seek to expand their knowledge base, the resultant search processes are restricted to familiar and proximate areas (Rosenkopf and Almeida 2003). Therefore, we argue that a moderate level of technological diversity

enables project teams to access diverse knowledge from different technological areas and provide creative learning benefits, while ensuring to absorb and integrate new knowledge as well as to eliminate coordination problems. This leads us to the following hypothesis:

H7: A moderate level of technological diversity results in higher project performance rather than very high or very low levels of technological diversity.

4.3.4. Network Location

Centrality is defined as the extent to which an actor occupies a central position in the network (Wasserman and Faust 1994). Developers who are more active in the network act as a central actor in the network and are viewed as major channels of information in the network (Singh et al. 2011, Singh et al. 2007). High centrality enables greater amounts of information and knowledge to be exchanged and integrated more rapidly. First, high centrality allows developers to have a broad range of knowledge, including an understanding where such knowledge is located and how to obtain it (Hansen 2002), which is unavailable to peripheral developers (Lin et al. 2007). Central developers occupy a structurally advantageous position to see a more complete picture of all the alternatives available in the network than the peripheral developers, so they have a broad range of opportunities unavailable to those in the periphery (Lin et al. 2007). A central developer has access to unique knowledge, including an understanding where such knowledge is located and how to obtain it (Hansen 2002). With such information, centrality enables a developer to make better decisions (Balkundi and Harrison 2006). Second, high centrality also allows developers to have quick access to knowledge in the network (Uzzi 1997, Powell and Smith-Doerr 1994). High centrality also allows developers to rapidly disseminate knowledge in the network (Powell and Smith-Doerr

1994). Third, high centrality allows developer to control (Wasserman and Faust 1994, Pfeffer and Salancik 1978), and regulate information flow among other developers (Wasserman and Faust 1994, Krackhardt 1996), dispensing what is needed to other team members (Balkundi and Harrison 2006). Thus, high centrality enhances a developer's ability to be central to the flow of information and resources in the network. Therefore, we argue that the centrality of a project is positively related to the performance of a project. This leads us to the following hypothesis:

H8: The performance of a project will be positively related to the centrality of a project.

4.3.5. Network Location of Ambidextrous Developers

Ambidextrous developers facilitate knowledge exchange and combination among other developers (Kogut and Zander 1992, Jansen et al. 2009). We assume that ambidextrous developers play an integration role by speeding up information flow and allowing information and knowledge to be exchanged and integrated more rapidly among other developers. Ambidextrous developers also play a control role to control and regulate information flow among other developers. A node in a structurally advantageous position in the network tends to receive benefits of information exchange and control (Burt 1992). Centrality measures the extent to which an actor occupies a central position in the network (Wasserman and Faust 1994). Therefore, central developers are viewed as major channels of information in the network (Singh et al. 2011, Singh et al. 2007). High centrality enables greater amounts of information and knowledge to be exchanged and integrated more rapidly. Central developers occupy a central position in the flow of information and resources in the network, which allows them to control and regulate

information flow among other developers. First, high centrality allows developers to have a broad range of knowledge, including an understanding where such knowledge is located and how to obtain it (Hansen 2002), which is unavailable to peripheral developers (Lin et al. 2007). Central developers occupy a structurally advantageous position to see a more complete picture of all the alternatives available in the network than the peripheral developers, so they have a broad range of opportunities unavailable to those in the periphery (Lin et al. 2007). A central developer has access to unique knowledge, including an understanding where such knowledge is located and how to obtain it (Hansen 2002). With such information, centrality enables a developer to make better decisions (Balkundi and Harrison 2006). Second, high centrality also allows developers to have quick access to knowledge in the network (Uzzi 1997, Powell and Smith-Doerr 1994). High centrality also allows developers to rapidly disseminate knowledge in the network (Powell and Smith-Doerr 1994). Therefore, high centrality more likely improves an integration role of ambidextrous developers by speeding up information flow and allowing information and knowledge to be exchanged and integrated more rapidly among other developers. Third, high centrality allows developer to control (Wasserman and Faust 1994, Pfeffer and Salancik 1978), and regulate information flow among other developers (Wasserman and Faust 1994, Krackhardt 1996), dispensing what is needed to other team members (Balkundi and Harrison 2006). Therefore, high centrality enhances ambidextrous developers' ability to be central to the flow of information and resources in the network. High centrally more likely improves a control role of ambidextrous developers to control and regulate information flow among other developers. Therefore,

we argue that the centrality of ambidextrous developers is positively related to the performance of a project.

The centrality of ambidextrous developers may be higher when they work on more projects. Thus, high centrality may imply that ambidextrous developers are working on more projects and may be exposed to too much information. However, individuals have the cognitive limitations to learning (Simon 1991). Exposure to too much information may lead to cognitive overload and poorer performance, which results in lower performance (Jansen et al. 2006). If ambidextrous developers are exposed to too much information, they may spend much time and effort to deal with overloaded information, which reduces their learning and performance (Narayanan et al. 2009). Therefore, we argue that the impact of the centrality of ambidextrous developers on the performance of a project will be moderated by the number of projects on which ambidextrous developers work. This leads us to the following hypothesis:

H9: The performance of a project will be positively related to the centrality of ambidextrous developers.

H10: The impact of the centrality of ambidextrous developers on the performance of a project will be moderated by the number of projects on which ambidextrous developers work: the greater number of projects on which ambidextrous developers work, the lower impact of the centrality of ambidextrous developers on the performance of a project.

4.4. Data

4.4.1. Data Sources and Collection

OSS network data required for this study has been collected from the SourceForge database (SourceForge.net). The SourceForge database is the primary repository for OSS projects and accounts for about 90% of all open source projects (Singh et al. 2011). Although all OSS projects are not hosted at the SourgeForge database and there are other OSS hosting websites such as BerliOS Developer and GNU Savannah, the SourgeForge database is the largest OSS development and collaboration website (Xu et al 2005). It can be considered as the most representative of the OSS community because the large number of projects and developers registered the SourgeForge database (Singh et al. 2011, Grewal et al. 2006, Xu et al 2005). Researchers analyzing issues related to OSS development phenomenon have predominantly used SourceForge data (Singh et al. 2011, Singh 2010, Singh 2007, Grewal et al 2006). The SourceForge database provides storage space and services to OSS projects in order to organize and coordinate software development activities by providing project web servers, trackers, mailing lists, discussion boards, and software releases (Xu et al 2005). This database contains software for download as well as statistics related to OSS projects. Researchers can create database programs to download statistics that are of interest.

Our research objective is to study the effect of social network properties of OSS developers and ambidexterity on project performance. Therefore, we need to collect affiliation network data in order to construct the network of OSS developers. Given a set of projects and developers, there are two methods to collect affiliation network data: Snowball method and Whole network method (Hanneman and Riddle 2005). The whole

network method yields maximum information, but it can also be difficult to execute while the snowball method yields considerably less information about network structure, but it is often less difficult to implement (Hanneman and Riddle 2005).

The snowball method begins with a focal actor or set of actors. Then, all the actors connected to a focal actor or set of actors are tracked down. The snowball process continues until no new actors are identified, or a large enough number of observations is collected for analysis. However, there are major potential limitations of the snowball method (Hanneman and Riddle 2005). First, actors who are not connected (i.e. actors in different components) are not reached through this method. The snowball method may tend to overstate the connectedness and solidarity of populations of actors based on the starting actors and their connectivity to other actors. Therefore, there is no guaranteed way of finding all of the connected individuals in the population.

The whole network method requires that we collect information about each developer's ties with all other developers. Because we collect information about ties between all developer-project pairs, full network data give a complete picture of relations in the population (Hanneman and Riddle 2005). Whole network data is necessary to properly define and measure many of the structural concepts of network analysis (Hanneman and Riddle 2005). Whole network data also allows for very powerful descriptions and analyses of social structures (Hanneman and Riddle 2005). However, whole network data can also be very difficult to collect. The data collection task is made more manageable by determining an appropriate boundary around the network since the whole network method examines actors that are regarded as bounded social collectives (Marsden 2005, Singh et al. 2011). This is the predominant method used in situation

where an appropriate network boundary is established. Prior studies on OSS development used software development platforms called project foundries as a network boundary. Project foundries are mainly built on programming languages, thereby project foundry and programming language are similar concepts. For example, Singh et al. (2011) used participation in Python foundry (uses Python programming language) and Grewal et al. (2006) used participation in Perl foundry (uses Perl programming language) as a network boundary. However, foundry data associated with OSS projects was not available at the SouceForce database after 2005. Therefore, there is no way for us to associate projects with foundries.

We used the whole network method to collect affiliation network data and selected the C programming language as a network boundary. The selection of the C programming language as a network boundary is acceptable for several reasons. First, it is the system implementation language for the UNIX operating system and UNIX/Linux operating system is dominant in OSS community (Subramanian et al. 2009). Second, it is one of the preferred languages of OSS developers for codes that require portability, need faster processing, have real-time requirements, or are tightly coupled to the UNIX/Linux kernel (Subramanian et al. 2009). Third, developers who are familiar with the programming language are able to understand the source code easily (Subramanian et al. 2009), thereby more efficient knowledge sharing may be possible within a project or across projects written in the same programming language. Fourth, we analyzed the number of projects and associated developers across programming languages and found that the C language is in the top three languages used by the large number of software developers at SourgeForge.

Data collection started by identifying developer-project pairs since OSS developers may work on multiple projects simultaneously if they are members of different project teams. A relationship exists between any two developers if they are members of different project teams and consequently work together on the same project. These kinds of relationships between developers and projects can be represented by an affiliation network (Wasserman and Faust 1994). Affiliation data for projects and developers has been collected from the SourceForge database for projects registered from January 1999 to December 2008 at the SourceForge website. We have set December 2008 as a cutoff date for our study for several reasons. First, constructing a network and calculating a variety of social network measures are extremely computation intensive especially for larger networks. We used social network software (UCINET) (Borgatti et al. 2002) to perform calculations and wrote our own code when required to construct a network as well as to perform some calculations. We analyzed the number of developers for projects written in the C language for each year from 2003 to 2011. We found that networks (especially project developers' network used in Chapter 4) have large number of developers (≥15,000) after December 2008 as shown in Table 14. This results in extremely large networks that are challenging to process with UCINET. Second, the first data snapshot of the SourceForge database is available for January 2003. The difference between our cutoff date and the first data snapshot date of the SourceForge data is 5 years which provides sufficient variation in network characteristics. Third, we had a concern for data availability of our dependent variables (the number of versions) because the SourceForge database provides data for our dependent variables until December 2008.

TABLE 14: Project Statistics across Years

Years	Number of Projects	Number of Developers
Jan 31, 2003	741	4,371
Dec 31, 2004	1,271	6,999
Dec 31, 2005	1,532	8,400
Dec 31, 2006	1,830	9,935
Dec 31, 2007	2,117	11,330
Dec 31, 2008	2,374	12,665
Dec 31, 2009	2,515	14,933
Dec 31, 2010	2,608	15,564
Dec 31, 2011	2,665	15,950

In order to identify developer-project pairs, we identified all projects that match following criteria. First, we included the projects which are written in the C language (our network boundary). Second, we excluded projects which have neither patch nor feature request activities in order to ensure the calculation of project ambidexterity. Prior research also indicated that a large proportion of projects hosted at the SourceForge database show no activity (Singh et al. 2011, Singh 2010, Chengalur-Smith and Sidorova 2003). These projects would be dead nodes in the network and the relationships involving them would not facilitate any knowledge transfers or spillovers (Singh et al. 2011). Therefore, including such projects in the network may lead to misleading results. By following prior research (Singh et al. 2011, Singh 2010), we excluded those projects. If a project has neither patch nor feature request activities, we considered those projects as inactive because we assume that they showed no sign of activity since their inception until December 2008. Third, among the projects matching previous criteria, we selected the projects whose patch development and feature request activities have been successfully closed. In this way, our data collection procedure is the consistent with our data collection procedure in Chapter 3. This is important since project ambidexterity is calculated based on ambidextrous developers identified in Chapter 3. For the projects that

match our criteria, we identified the developers who joined to projects. This allows us to collect affiliation network data (developer-project pairs) and construct an affiliation network for projects.

4.4.2. Network Construction

OSS network data analyzed in this study is the affiliation data between developers and projects. Social network of the OSS community is represented by an affiliation network such as a two-mode network based on a developer-project pair. However, in order to analyze the structure of OSS networks, we need a one-mode network at the developer level. Therefore, we construct a project network in two steps.

We construct an affiliation network for projects based on the developer-project pairs. In this affiliation network, the actors are unique developers, and the events are projects. A relationship exists between two developers if they work together on the same project. Figure 5 illustrates the process of developer affiliation network construction. In Figure 5a, each project has its own set of developers. A square node represents a unique project and a circular node represents a unique developer. A link between any two developers exists if they work on the same project. Figure 5b shows the developer network for individual projects. However, some developers (D5 and D10) work on more than one project simultaneously. Thus, they belong to more than one project team and they are used to connect the individual teams in the network as shown in Figure 5c (which shows the network of developers across projects). In Figure 5c, a node represents a unique developer.

A binary adjacency matrix (the matrix *P*) of affiliation networks represents the relationships between projects and developers in the network in Figure 6a. The adjacency

matrix of affiliation networks lists unique developers across multiple projects. A row represents developers, and a column represents projects. When a developer belongs to a project, the corresponding matrix element gets a value of one, and zero otherwise. The transpose (the matrix P^T) of an adjacency matrix of affiliation networks represents the relationships between projects and developers in the network in Figure 6b. A row represents projects, and a column represents developers. We converted two-mode network data to one-mode network data by multiplying an adjacency matrix (the matrix P) of affiliation networks with the dot product of the transpose (the matrix P^T) of an adjacency matrix of affiliation networks expressed as follows:

$$X^P = PP^T \quad [3]$$

An adjacency matrix (the matrix X^P) of a project network represents the relationships between any two developers in Figure 6c. The row and the column represent unique developers. A value of one or more corresponding to the pair of two developers in the network indicates a presence of a relationship between them, and a value of zero indicates the absence of relationship. The adjacency matrix is undirected because relationship among two developers is mutual. We converted all values greater than one to one which simply indicates a presence of a relationship between two developers. This final adjacency matrix is our final network which is used in our analysis. The final network includes 2,374 projects and 12,665 developers.

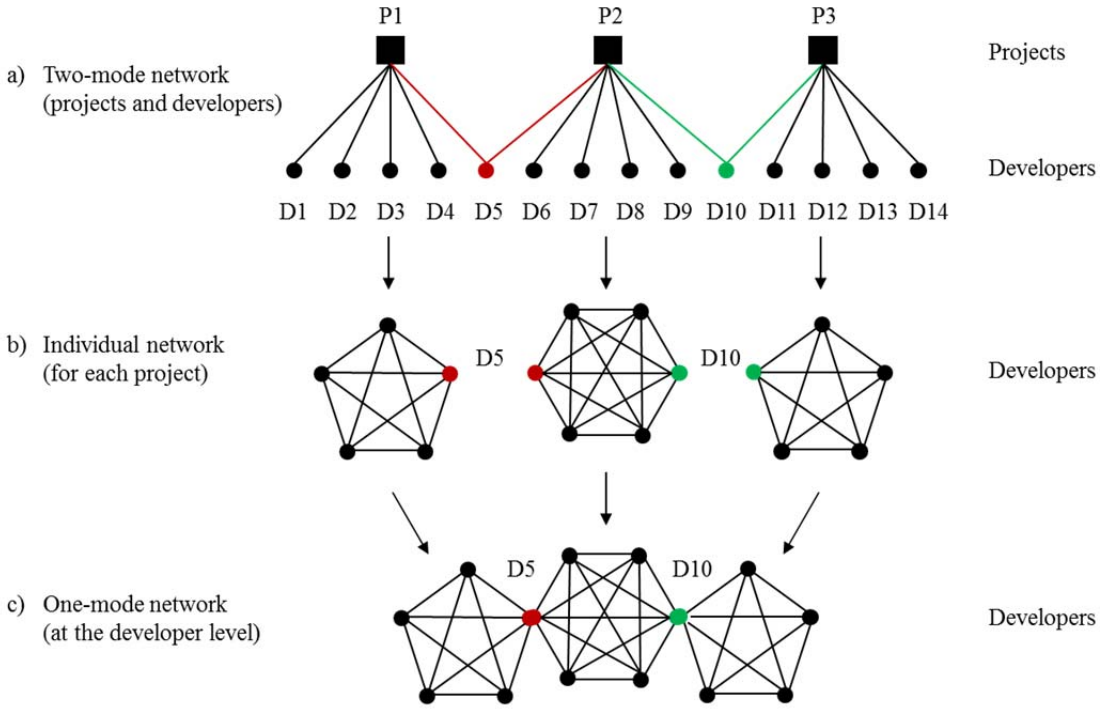

FIGURE 5: OSS Network Construction at the Project Level

$$P = \begin{array}{c} \\ D1 \\ D2 \\ D3 \\ D4 \\ D5 \\ D6 \\ D7 \\ D8 \\ D9 \\ D10 \\ D11 \\ D12 \\ D13 \\ D14 \end{array} \begin{array}{c} \begin{array}{ccc} P1 & P2 & P3 \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{array}\right] \end{array}$$

a) Two-Mode Adjacency Matrix of Projects and Developers

FIGURE 6: Matrix Representations of OSS Project Network at the Project Level

$$P^T = \begin{array}{c} \\ P1 \\ P2 \\ P3 \end{array} \begin{array}{c} \begin{array}{cccccccccccccc} D1 & D2 & D3 & D4 & D5 & D6 & D7 & D8 & D9 & D10 & D11 & D12 & D13 & D14 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

b) Transpose of Two-Mode Adjacency Matrix of Projects and Developers

$$X^P = \begin{array}{c} \\ D1 \\ D2 \\ D3 \\ D4 \\ D5 \\ D6 \\ D7 \\ D8 \\ D9 \\ D10 \\ D11 \\ D12 \\ D13 \\ D14 \end{array} \begin{array}{c} \begin{array}{cccccccccccccc} D1 & D2 & D3 & D4 & D5 & D6 & D7 & D8 & D9 & D10 & D11 & D12 & D13 & D14 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

c) One-Mode Adjacency Matrix of OSS Projects

FIGURE 6: Cont'd

4.5. Variable Definitions and Operationalization

4.5.1. Dependent Variables

Our theoretical background and associated hypotheses rely on the effect of social network properties of OSS developers and ambidexterity on project performance. Recent studies on OSS project performance measured OSS project performance in terms of the technical performance of a project which represents the rate of knowledge creation by a project (Singh et al. 2011, Singh 2010, Greval at al. 2006), and the commercial

performance of a project (Singh 2010, Greval at al. 2006). We measured the technical performance (knowledge creation) of OSS projects with two types of dependent variables. First, by following the prior research on OSS project performance, we measured the technical performance of OSS projects with the Concurrent Versions System (CVS) which commonly used in the OSS literature (Singh et al. 2011, Singh 2010, Grewal et al. 2006, Rai et al. 2002). Second, we measured the technical performance of OSS projects with the Subversion System (SVN). CVS and SVN are commonly used Source Code Management (SCM) tools to manage and track changes in software source code (SourceForge.net). The number of the CVS commits and SVN commits are our technical performance measures. Although we did not expect two types of technical performance measures, in the data collection, we found that there are two types of technical performance measures. Therefore, we focused on the technical performance of a project. The commercial performance model is the topic of future research since significant time involves collecting and analyzing data for the commercial performance of a project.

4.5.1.1. Technical Performance of a Project

Based on Grant (1996)'s knowledge-based theory, a project is a structure to turn team members' knowledge into products. Therefore, software is a knowledge product (Slaughter et al. 2006) and the amount of knowledge created by a project measures the performance of a project (Singh et al. 2011). The dependent variable should represent the amount of knowledge created by a project.

Extant research on software development has suggested the use of modification requests (MR's) as a measure of the rate of knowledge creation by a project that follows

an incremental software development approach (Singh et al. 2011, Singh 2010, Boh et al. 2007, Grewal et al. 2006). The MR measure represents the addition of new functionality as well as the modification or repair of old functionality (Singh et al. 2011). In OSS development literature, the CVS and SVN commit transactions measure a basic addition of functionality similar to that taken into account by the MR measure in a commercial development environment (Mockus et al. 2002). Therefore, the MR measure is equivalent to CVS and SVN commits (Van Antwerp and Madey 2008).

Software developers use the CVS and SVN to manage the software development process. The CVS and SVN enable project teams to store source code at a central location, thus enabling team members to retrieve the source code to make changes. The CVS and SVN also help project teams to keep track of every change, including what was changed, when it was changed, and who made the change, and help in blending changes made by different developers, including ensuring that developers do not accidentally overwrite each other's alterations. CVS and SVN commits occur when a developer uploads the altered source code file, and the CVS and SVN tool updates the changed files automatically.

Recent studies on OSS development have used CVS commits as a measure of the technical performance of a project, and have not use SVN commits since the SVN is relatively new (Van Antwerp and Madey 2008). After the SVN has been made available, the adoption of the SVN was widespread throughout the OSS community and many projects have migrated from the CVS to the SVN (Van Antwerp and Madey 2008). Therefore, many projects have used the CVS in their first years and then started using the SVN by migrating from the CVS to the SVN. The CVS and SVN may not be used in

parallel mode because this potentially creates difficulties for projects to track the same changes in different systems. Projects may use the CVS and SVN for different modules of their projects. Therefore, the same file (changes in source code) may not be submitted to both systems. The overlapping between the CVS and SVN commits is very low. Based on our assumptions, the marginal error to use the CVS and SVN commits as a combined measure may be very low.

As CVS and SVN commits reflect changes to source code, we used the number of CVS commits as well as the sum of CVS and SVN commits (combined score) as measures of the technical performance of a project (the rate of knowledge creation by a project). These measures of the technical performance of a project are consistent with the literature on information system success (DeLone and McLean 1992). Recent studies which are closely related to our study have used the number of CVS commits as a measure of the technical performance of a project (Singh et al. 2011, Singh 2010, Grewal et al. 2006, Rai et al. 2002).

4.5.2. Independent Variables

Based on the finding of organizational literature (Jansen et al. 2009, Gupta and Govindarajan 2000), we identified a new category of developers (ambidextrous developers) in OSS projects. We develop a new theoretical construct for OSS project ambidexterity based on the concept of ambidextrous developers.

Social network analysis (Wasserman and Faust 1994) has been used in a variety of contexts to study the relationship between social entities. Structural properties of the networks are used to analyze the network. Many structural properties of these networks could have multiple social network measures. For example, there are different types of

internal cohesion measures (clustering coefficient, repeat ties, third party ties, and structural equivalence), external connectivity measures (external cohesion, direct ties, indirect ties, and technological diversity), and network location measures (degree centrality, betweenness centrality, and closeness centrality). Consistent with previous studies on social network research (Gnyawali and Madhavan 2001, Ahuja 2000, Uzzi 1999, Uzzi 1997, Uzzi 1996, Watts and Strogatz 1998, Krackhardt 1998, Wasserman and Faust 1994, Burt 1992, Coleman 1988, Freeman, 1979, Granovetter 1973), organizational research (Schilling and Phelps 2007, Hansen 2002, Hansen 1999, Reagans and Zuckerman 2001), and OSS development research (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006), we categorized our social network variables into three categories: internal cohesion, external connectivity, and network location. In the following section, we describe our variables used in this study along with the construction of their measures.

4.5.2.1. Ambidexterity

Recent research on organizational performance has realized the importance of ambidexterity and begun to study ambidexterity based on perceptual (survey) data in the context of formal organizations (Jansen et al. 2009, Jansen et al. 2006, Jansen et al. 2005, Lin et al. 2007). These studies have used ambidexterity as a measure of the ability of organizations to pursue both exploitative and exploratory activities concurrently.

Based on the finding of organizational literature (Jansen et al. 2009, Gupta and Govindarajan 2000), some developers are expected to be members of teams involved in exploitative activities (patch development) and members of teams involved in exploratory activities (feature request). Consistent with the finding of organizational literature, we

identified a new category of developers (ambidextrous developers) in OSS projects who are members of teams involved in exploitative activities (patch development) and members of teams involved in exploratory activities (feature request), and contribute to both types of OSS activities. We develop a theoretical construct for project ambidexterity based on the concept of ambidextrous developers. We assume that the contribution of ambidextrous developers to patch development activities is independent from their contribution to feature request activities, or vice versa.

We measured project ambidexterity as the percentage of ambidextrous developers in a project. We calculated ambidexterity for a project as follows. First, we identified ambidextrous developers from their memberships to exploitative activities (patch development) and exploratory activities (feature request) for each project in Chapter 3. Second, for each project, we identified project developers from their memberships to projects. Third, we calculated the percentage of ambidextrous developers in a project as a measure of ambidexterity for a project. Therefore, the measure of ambidexterity ranges from 0 to 1. A high score of project ambidexterity indicates a project is mostly consisted of ambidextrous developers rather than non-ambidextrous developers. The square of ambidexterity is also included as an independent variable to capture the curvilinear relationship as hypothesized in Hypothesis 1.

4.5.2.2. Internal Cohesion

We measured internal cohesion for a project with clustering coefficient, repeated ties, third party ties, and structural equivalence (Jaccard similarity and correlation similarity).

Clustering Coefficient: The clustering coefficient captures the degree to which the overall network contains localized pockets of dense connectivity (Watts and Strogatz 1998, Watts 1999). The clustering coefficient mainly measures the extent to which two related developers share a relationship with a common third.

We measured the clustering coefficient for a project by following Watts and Strogatz (1998). For each project developer, we calculated the clustering coefficient (see Appendix B for the calculation of clustering coefficient). We took an average of each project developer's clustering coefficient over all the project developers to calculate a measure of the clustering coefficient for a project.

The clustering coefficient lies strictly in the range from 0 to 1. The value of 1 indicates that all developers in the network share a direct relationship with each other. That means each developer is directly connected to all other developers in the network, which results in extreme clustering. In contrast, the value of 0 indicates that any two connected developers do not share a relationship with a common third. A high score of the clustering coefficient indicates greater clustering.

Repeat Ties: Repeated collaboration among project members captures the strength of interpersonal connections among team members (Uzzi 1996, Uzzi 1999, Singh et al. 2011). Strong interpersonal connections indicate the presence of repeat collaborations among project members (Uzzi 1997). As developers interact more frequently, the strength of the collaborative tie increases, and they develop more closer and cohesive relationships (Granovetter 1973, Hansen 1999). Team members rely on repeated ties developed through joint participation in past teams because they are motivated to continue to work with those with whom they have collaborated in the past (Hahn et al.

2008). Repeated ties from past interactions may result in greater trust and knowledge for developers (Uzzi and Spiro 2005).

We measured the number of repeated ties for a project by following Singh et al. (2011). We counted the total number of projects on which each pair of project developers have worked together. We divided this number by the total number of pairs that exist in a project to calculate a measure of repeat ties for a project. A high score of repeat ties indicates that project developers have worked together on several projects.

Third Party Ties: Third party ties support direct relationships and imply that a project team is composed of developers who work with many of the same collaborators (Szulanski 1996, Coleman 1988, Singh et al. 2011). Third part ties are important for the existence of effective norms and the trustworthiness in social structures (Coleman 1988). Similarly, the concept of simmelian ties are the same with third party ties (Krackhardt 1998). Two people are simmelian tied to one another if they are reciprocally and strongly tied to each other and to another one in common (Krackhardt 1998). Simmelian ties enhance the conflict resolution and group norms (Krackhardt 1998).

We measured the number of third party ties for a project by following Singh et al. (2011). We counted the total number of third party ties of all pairs of project developers around the members of a project team (besides the focal team members). We divided this number by the total number of pairs that exist in a project to calculate a measure of third party ties for a project. A high score of third party ties indicates that project developers have worked together with other developers on several projects.

Structural Equivalence: The structural equivalence measures to the extent to which two actors have identical relationships to all other actors, i.e. they jointly occupy

the structurally equivalent position in the network (Wasserman and Faust 1994). Thus, the structural equivalence is a pair-level measure of how similar the actors' network patterns are. Structurally equivalent actors have a similar pattern of relationships to other actors in the network (Wasserman and Faust 1994, Gnyawali and Madhavan 2001). Structurally equivalent actors tend to have similar profiles and behaviors (Gnyawali and Madhavan 2001). Structurally equivalent actors tend to interact with similar others in similar ways, which results in similar attitudes, resources, and behaviors (Gnyawali and Madhavan 2001). Therefore, structurally equivalent actors may have similar asset, information, and resources (Gnyawali and Madhavan 2001).

We measured the structural equivalence for a project with two measures: Jaccard similarity, and Correlation similarity (Wasserman and Faust 1994). Jaccard similarity measures the similarity of the relationships of two developers by comparing the size of the overlap against the size of the relationships of two developers (Wasserman and Faust (1994). Correlation similarity measures the similarity of the relationships of two developers by calculating Pearson's correlation of the relationships of two developers (Wasserman and Faust (1994). Correlation similarity measures the strength of the relationship between two developers and it is based on the similarity in pattern of ties whereas Jaccard similarity accounts for the identity of ties between two developers.

We calculated Jaccard similarity and correlation similarity as follows. We calculated the total of Jaccard similarity and correlation similarity of all pairs of project developers. We divided these numbers by the total number of pairs that exist in a project to calculate measures of Jaccard similarity and correlation similarity for a project. Jaccard similarity and correlation similarity lie strictly in the range from 0 to 1. A value of one

represents perfect structural equivalence whereas a value of zero represents no structural equivalence. A high score of structural equivalence indicates that project developers worked with many of the same developers.

4.5.2.3. External Connectivity

We measured external connectivity for a project with external cohesion, direct ties, indirect ties, and technological diversity.

External Cohesion: We measured the external cohesion with Burt's (1992) network constraint. Network constraint measures the extent to which a project member's external contacts share relationships with each other.

We calculated the external cohesion for a project as follows. For each project developer, we calculated the network constraint (see Appendix B for the calculation of external cohesion). We took an average of each project developer's network constraint over all the project developers to calculate a measure of the network constraint for a project. Higher values of external cohesion indicate that external contacts of a project are more directly connected with each other, which indicates greater external cohesion. In contrast, lower values of external cohesion indicate that external contacts of a project are less directly connected with each other, which indicates smaller external cohesion. The square of the external cohesion is also included as an independent variable to capture the curvilinear relationship as hypothesized in Hypothesis 2.

Direct Ties: We measured direct ties by following Ahuja (2000). Direct ties measure the extent to which project members are directly connected to external contacts. Direct ties are also associated with the capacity of a project to acquire tacit knowledge from outside (Singh et al. 2011).

We calculated direct ties for a project as follows. For each project developer, we counted the number of developers who a project developer has ties with other than the other team members of the project. We took an average of this number over all the project developers to calculate a measure of direct ties for a project. Higher values of direct ties indicate that a project is more directly connected to external contacts.

Indirect Ties: Indirect ties are ties that provide access to external developers at path distances of two or greater (local project developers' partner's partners), which excluded direct ties. Indirect ties measure the extent to which project members are indirectly connected to external partner's partners. Indirect ties are also associated with the capacity of a project to acquire explicit knowledge from outside (Singh et al. 2011).

We used two measures for indirect ties. The first measure is the number of indirect ties. For each project developer, we counted the number of developers with whom a project developer does not have a direct tie but can reach through others (at path distances of two or greater, which excluded direct ties). We took an average of this number over all the project developers to calculate a measure of indirect ties for a project.

This measure does not account for the weakening or decay of tie strength as distance between two developer's increases (Ahuja 2000). Burt (1992) provided a frequency decay measure for indirect ties that accounts for this decline in tie strength across distant ties (see Appendix B for the calculation of indirect ties with frequency decay function). Thus, our second measure for indirect ties is a frequency decay measure proposed by Burt (1992). The argument for the frequency decay function is that the rate at which the strength of a relation decreases with the increasing length of its corresponding path distance should vary with the social structure in which it occurs (Burt

1992). The larger the number of developers to which the focal project developer must devote their time and energy, the weaker the relationship that the focal project developer can sustain with any individual developer. Thus, decay in the strength of a relationship is related to the number of other developers reached at each path distance.

For each project developer, we calculated a frequency decay function for indirect ties. We took an average of this number over all the project developers to calculate a measure of indirect ties with a frequency decay function for a project. Higher values of indirect ties indicate that a project is more indirectly connected to external partner's partners at path distances of two or greater. The interaction terms of the number of direct ties with the number of indirect ties and indirect ties with frequency decay function are also included to capture the interaction effect between direct and indirect ties as hypothesized in Hypothesis 6.

Technological Diversity: Technological diversity measures the extent to which two projects are different in terms of the angular distance of their technological positions. In order to calculate the technological diversity for a project, we defined the technological position of a project. The technological position of a project can be defined in terms of different dimensions such as the type of the project, programming language, user interface, and operating system (Singh et al. 2011). Each of these dimensions represents different type of technical expertise. Project type represents the application domain knowledge whereas the other three dimensions represent the tools knowledge and expertise that comprise the knowledge of process, data and functional architecture (Kim and Stohr 1998, Singh et al. 2011). The similarity of domain and tools affect the amount of knowledge that can be reused from one project to another (Singh et al. 2011).

Following Jaffe (1986), we characterized a project's technological position by a vector $F_p = (F_1 \ldots F_k)$, where k is the total number of categories under the four dimensions, and F_k is an indicator variable that equals to 1 if the project p falls under the category k. A project can fall under several categories within a single dimension. Technological diversity between the two projects p and q is then calculated by the angular separation or uncentered correlation of their vectors (see Appendix B for the calculation of technological diversity).

We calculate the technological diversities of all pairs of a focal project with all of the projects with which it shares a developer. We summed these measures and divided it by the number of projects (the total number of project pairs) to calculate a measure of technological diversity for a project. Technological diversity lies in the range from 0 to 1. A value of one represents the greatest technological diversity between two projects. The square of technological diversity is also included as an independent variable to capture the curvilinear relationship as hypothesized in Hypothesis 7.

4.5.2.4. Network Location

We measured network location for a project with network centralities: degree centrality, closeness centrality, and betweenness centrality.

Degree Centrality: We measured the degree centrality with Freeman's (1979) degree centrality. Degree centrality is the measure of how many an actor is connected to other actors in the network through direct connections (Freeman 1979, Wasserman and Frost 1994). Degree centrality of a developer reflects the activeness of a developer in the network. Developers who are more active in the network act as a central actor in the

network and are viewed as major channels of information in the network (Singh et al. 2011, Singh et al. 2007).

We calculated the degree centrality for a project as follows. For each project developer, we calculated the degree centrality (see Appendix B for the calculation of degree centrality). We took an average of each project developer's degree centrality over all the project developers to calculate a measure of the degree centrality for a project.

The degree centrality is normalized by dividing by the maximum possible degree in the network which is that one actor is connected to all other actors in the network. This calculation results in that the degree centrality lies in the range from 0 to 1. However, UCINET reports the normalized degree centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of degree centrality for a project ranges from 0 to 100. A high score of the degree centrality indicates a project is comprised of developers who are connected to many developers in the network.

Betweenness Centrality: We measured the betweenness centrality with Freeman's (1979) betweenness centrality. Betweenness centrality is the measure of how often a developer falls on the shortest path between pairs of other developers (Freeman 1979, Wasserman and Faust 1994). Developers with a high betweenness centrality lie in the shortest path of information flow between other developers. These developers can exert control over information flow among other developers, and potentially may have some control over the interactions between other developers (Wasserman and Faust 1994). Thus, betweenness centrality signifies a developer's ability to be central to the flow of information and resources in the network. These developers can be important to the

network-wide information diffusion process by occupying a central position on the shortest path between other developers in a network.

We calculated the betweenness centrality for a project as follows. For each project developer, we calculated the betweenness centrality (see Appendix B for the calculation of betweenness centrality). We took an average of each project developer's betweenness centrality over all the project developers to calculate a measure of the betweenness centrality for a project.

The betweenness centrality is normalized by dividing by the maximum possible betweenness in the network which is the number of pairs of actors not including a focal actor (the maximum possible paths passing through a focal actor). This calculation results in that the betweenness centrality lies in the range from 0 to 1. However, UCINET reports the normalized betweenness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of betweenness centrality for a project ranges from 0 to 100. A high score of the betweenness centrality indicates a project is comprised of developers who fall on many shortest paths between other developers.

Closeness Centrality: We measured the closeness centrality with Freeman's (1979) closeness centrality. Closeness centrality is the measure of how close an actor is to all other actors in the network through direct and indirect connections (Freeman 1979, Wasserman and Frost 1994). It basically measures the inverse of the sum of geodesic distances between actors in the network, thereby an actor with high closeness centrality has minimum geodesic distances to other actors. Closeness centrality signifies a developer's ability to reach resources in the network (Gulati and Gargiulo1999).

Information would have to travel over shorter distances to reach a developer who is more central in the network (Wasserman and Faust 1994). A developer who is close to many developers can quickly interact and communicate with them without passing through many intermediaries (Wasserman and Faust 1994).

We calculated the closeness centrality for a project as follows. For each project developer, we calculated the closeness centrality (see Appendix B for the calculation of closeness centrality). We took an average of each project developer's closeness centrality over all the project developers to calculate a measure of the closeness centrality for a project.

The closeness centrality is normalized by multiplying by the maximum possible path distance in the network which is that one actor is connected to another one actor passing through all other actors in the network. This calculation results in that the closeness centrality lies in the range from 0 to 1. However, UCINET reports the normalized closeness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of closeness centrality for a project ranges from 0 to 100. A high score of the closeness centrality indicates a project is comprised of developers who are very close to all other developers in the network via shortest paths.

4.5.2.5. Network Location of Ambidextrous Developers

We measured network location of ambidextrous developers for a project with their network centralities: degree centrality, closeness centrality, and betweenness centrality.

Degree Centrality of Ambidextrous Developers: We measured the degree centrality of ambidextrous developers with Freeman's (1979) degree centrality. Degree centrality is the measure of how many an actor is connected to other actors in the network through direct connections (Freeman 1979, Wasserman and Frost 1994). We calculated the degree centrality ambidextrous developers for a project as follows. For each ambidextrous developer, we calculated the degree centrality (see Appendix B for the calculation of degree centrality). We took an average of each ambidextrous developer's degree centrality over all ambidextrous project developers to calculate a measure of the degree centrality of ambidextrous developers for a project.

The degree centrality is normalized by dividing by the maximum possible degree in the network which is that one actor is connected to all other actors in the network. This calculation results in that the degree centrality lies in the range from 0 to 1. However, UCINET reports the normalized degree centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of degree centrality for a project ranges from 0 to 100. A high score of the degree centrality indicates a project is comprised of ambidextrous developers who are connected to many developers in the network.

Betweenness Centrality of Ambidextrous Developers: We measured the betweenness centrality of ambidextrous developers with Freeman's (1979) betweenness centrality. Betweenness centrality is the measure of how often a developer falls on the shortest path between pairs of other developers (Freeman 1979, Wasserman and Faust 1994).

We calculated the betweenness centrality of ambidextrous developers for a project as follows. For each ambidextrous developer, we calculated the betweenness centrality (see Appendix B for the calculation of betweenness centrality). We took an average of each ambidextrous project developer's betweenness centrality over all ambidextrous project developers to calculate a measure of the betweenness centrality of ambidextrous developers for a project.

The betweenness centrality is normalized by dividing by the maximum possible betweenness in the network which is the number of pairs of actors not including a focal actor (the maximum possible paths passing through a focal actor). This calculation results in that the betweenness centrality lies in the range from 0 to 1. However, UCINET reports the normalized betweenness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of betweenness centrality for a project ranges from 0 to 100. A high score of the betweenness centrality of ambidextrous developers indicates a project is comprised of ambidextrous developers who fall on many shortest paths between other developers.

Closeness Centrality of Ambidextrous Developers: We measured the closeness centrality of ambidextrous developers with Freeman's (1979) closeness centrality. Closeness centrality is the measure of how close an actor is to all other actors in the network through direct and indirect connections (Freeman 1979, Wasserman and Frost 1994).

We calculated the closeness centrality of ambidextrous developers for a project as follows. For each ambidextrous developer, we calculated the closeness centrality (see Appendix B for the calculation of closeness centrality). We took an average of each

ambidextrous developer's closeness centrality over all ambidextrous project developers to calculate a measure of the closeness centrality of ambidextrous developers for a project.

The closeness centrality is normalized by multiplying by the maximum possible path distance in the network which is that one actor is connected to another one actor passing through all other actors in the network. This calculation results in that the closeness centrality lies in the range from 0 to 1. However, UCINET reports the normalized closeness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of closeness centrality for a project ranges from 0 to 100. A high score of the closeness centrality indicates a project is comprised of ambidextrous developers who are very close to all other developers in the network via shortest paths.

4.5.2.6. Number of Projects which Ambidextrous Developers Work

The number of projects is the measure of how many projects ambidextrous developers work on. We calculated the number of projects for a project as follows. For each ambidextrous developer, we counted the number of projects on which that ambidextrous developer works. We took an average of this number over all ambidextrous project developers to calculate a measure of the number of projects for a project. The interaction terms of the number of projects with the degree, betweenness and closeness centralities of ambidextrous developers are also included to capture the interaction effect the number of projects and ambidextrous developers' centralities as hypothesized in Hypothesis 9.

4.5.3. Control Variables

Consistent with prior research, we included control variables in order to control effects of factors other than independent variables. We categorized control variable into following categories: team human capital and ability, user input and market potential, project age, and project characteristics (Singh et al. 2011, Singh 2010, Greval at al. 2006).

Team Human Capital and Ability: We included the number of developers (project team size) associated with a project to account for human capital actively involved in the project.

User Input and Market Potential: Although all projects use the C programming language, the software developed by the project team may differ in terms of market potential and extent of user participation (Singh et al. 2011). Users often play a critical role in the development and evolution of an open source product (Von Hippel and Von Krogh 2003). Activities such as bug reports, bug fixes, and user support requests are user-driven activities since bugs and support requests represent user inputs to OSS projects (Greval at al. 2006). Bugs play an important role to identify defects in software (SourceForge.net). Support request made by users are associated with specific questions and offered solutions which represent the collection of feedbacks (SourceForge.net). Following Singh et al. (2011) and Greval at al. (2006), we controlled user inputs to OSS projects by constructing two variables: the number of support requests and the number of bugs. The number of support requests is constructed as the cumulative number of support requests answered. The number of bugs is constructed as the cumulative number of bugs closed.

Page views directly signals the general interest of users in the project and its market potential (Greval at al. 2006). Following Singh et al. (2011) and Greval at al. (2006), we controlled the general interest of users in the project and its market potential with the number of page views which is constructed as the cumulative number of project pages viewed.

Project Life-Cycle Effects: The software life cycle may also affect the dependent variables (the number of CVS commits and the sum of CVS and SVN commits) since the dependent variables are more likely to increase with the age of a project (Singh et al. 2011 and Greval at al. 2006). We controlled the effect of project life-cycle on the dependent variables with a project age which is constructed as the number of months since a project's inception at SourceForge by network construction date. However, projects are more likely to see a relatively higher CVS and SVN commits rate close to the inception of the project as compared to later stages where the complexity of the software increases with growth, making it harder to make improvements (Singh et al. 2011). To control for potential nonlinear effect of project age on the dependent variables, we also included the square of the project age. This accounts for the potential complexity associated with software as the code grows.

Project Characteristics: Following Singh et al. (2011), we construct a broad range of variables to control the effects of software characteristics on the dependent variables. We control for a project type, intended audience, and user interface, language, and development status by constructing dummy variables.

The type of the project may indicate the potential market size of the software. We controlled the type of the project with 18 measures: Communications, Database,

Education, Formats Protocols, Games and Entertainment, Internet, Mobile, Multimedia, Office Business, Printing, Religion Philosophy, Scientific Engineering, Security, Sociology, Software Development, System, Terminals, Text Editors. A project can fall under several categories. The measure of the project type takes a value 1 if the project is categorized under that project type and 0 otherwise.

Intended audience may influence the quality of developers that are attracted towards a project. For example, software that is aimed towards system administrators is likely to attract more sophisticated developers (Lerner and Tirole 2002, Roberts et al. 2006). We controlled intended audience with 7 measures: Advanced End Users, Developers, Desktop End Users, Industry and Sector Users, Quality Engineers, System Administrators, Other Audience. A project can fall under several categories. The measure of intended audience takes a value 1 if the project is categorized under that intended audience and 0 otherwise.

User interface may also have an influence on the market size of a project. For example, software with graphical user interface is easier to use and is likely to be adopted more widely. We controlled user interface with 7 measures: Graphical Interface, Grouping and Descriptive Interface, Non-interactive Interface, Plugins, Textual Interface, Toolkits Libraries, Web Based Interface. A project can fall under several categories. The measure of user interface takes a value 1 if the project is categorized under that user interface and 0 otherwise.

Project's whose language is not English restrict both the number of users and developers that can participate in it. We construct the language as a dummy variable for

English. The measure of the language takes a value 1 if the project language is English and 0 otherwise.

We also controlled the development status of software. Software can be at one of the development stage: Planning, Pre-Alpha, Alpha, Beta, Production/Stable, and Mature. The measure of the development status of software takes a value of 1 for a planning stage, 2 for a pre-alpha stage, 3 for an alpha stage, 4 for a beta stage, 5 for a production/stable stage, and 6 a mature stage.

4.6. Research Methodology

Our research objective is to study the effect of social network properties of OSS developers and ambidexterity on OSS project performance. This study also examines the impact of coordination mechanisms (ambidextrous developers) on OSS project performance. In order to accomplish these research objectives and test the hypothesis developed in the previous sections, we employed the Ordinary Least Squares (OLS) regression.

OSS network data analyzed in this study is the affiliation network data of developers at the project level. We created a list of 2,374 projects from the SourceForge database for projects registered from January 1999 to December 2008. One of important issues for the OLS regression is the absence of outliers. An outlier is an observation with large residual and the multivariate statistics such as standardized residuals can be used to detect outliers (Tabachnick and Fidell 2007, Pedhazur 1997, Myers 1990). Standardized residuals that exceed ±3 indicate possible outliers (Tabachnick and Fidell 2007). We detected outliers for each individual model and then removed all observations from our data set. We identified 14 observations as outliers, and hence they were removed from

our data set. We tested our hypotheses by using the final data set including 2,360 observations (projects).

The required sample size for the OLS regression depends on a number of factors including the power (1-β), the alpha level (α), and the number of predictors (Tabachnick and Fidell 2007, Green 1991). When the alpha (α) is set at the 0.05 level, Cohen (1988) assumes that the risk of failure to find the beta (β) may be about four times less serious than the risk of finding what does not exist (α). The test with the power greater than 0.80 level is considered statistically powerful at the 0.05 alpha level (Green 1991, Cohen 1988). Given the significance alpha level (α=0.05) and the power level (1-β=0.80), Green (1991) suggested that at least $N > 50 + 8m$ (where m is the number of independent variables) observations are required for testing the significance of the multiple correlation, and at least $N > 104 + m$ observations are required for testing the significance of the individual predictors. Tabachnick and Fidell (2007) also recommended to choose the larger number of observations required by these two rules. In this study, we have 26 independent variables including the squares of independent variables and the interaction terms, and 40 control variables. As explained later, we also incorporated 3 interaction terms that are missing in the models since some independent variables may not be observable by themselves due to the interaction effect with other variables. We have 69 variables in total. Therefore, the sample size of 2360 is considered adequate when compared to the required sample size of 603 (50+8*69) for testing the multiple correlation. The sample size of 2360 is also considered adequate when compared to the required sample size of 173 (104+69) for testing the individual predictors.

The OLS regression assumes that the relationship between the dependent variable and each independent variable should be linear. However, we expect the curvilinear relationship (inverse U-shaped relationship) for some of our independent variables (ambidexterity, external cohesion, and technological diversity), and one of our control variables (project age). Therefore, these variables violate the linearity assumption. One of the remedies for this violation is to add the quadratic component of these variables as an independent variable (Myers 1990, Allison 1999). Thus, we included the squares of ambidexterity, external cohesion, technological diversity, and project age as independent variables in the model in order to capture the curvilinear relationship.

The OLS regression also assumes that the error term should follow the normal distribution. We tested this normality assumption with the Kolmogorov-Smirnov and Shapiro-Wilk tests of normality. We found that the error terms are normally distributed.

We found that the dependent variables (the number of CVS commits and the sum of CVS and SVN commits), some of the independent variables (repeat ties, third party ties, direct ties, indirect ties, indirect ties with frequency decay function, and the number of projects which ambidextrous developers work) and some of control variables (the number of support requests answered, the number of bugs closed, and the number of page views) were not normally distributed. In such a case, the OLS regression may yield biased parameter estimates that cannot be easily interpreted (Gelman and Hill 2007). Therefore, as suggested by Gelman and Hill (2007), we performed a logarithmic transformation on dependent, non-normally distributed independent and control variables. In addition, following Singh et al. (2011), we scaled down the number of direct ties, indirect ties, and frequency decayed indirect ties by the factor of 100 before performing a

logarithmic transformation. Based on a review of prior OSS literature, we identified two types of theoretical models in terms of analyzing of OSS project performance. We term them as the forecasting model (Singh et al. 2011 and Singh 2010) and the cumulative model (Grewal et al. 2006). We first reviewed the forecasting model used by Singh et al. (2011) and Singh (2010) which based on the use of a lag between network variables and the project performance measure. For example, Singh et al. (2011) measured the dependent variable (CVS Commits) as the number of CVS commits for a project in one year subsequent to network construction date. Thus, the dependent variable leads the independent variables by one year. However, the choice of network construction date is arbitrary. In addition, they used the cumulative number of CVS commits (the presample CVS in Singh et al. 2011) until the network construction date as a control variable in order to gauge the prior capacities of the project teams. However, there are major potential limitations of the forecasting model. First, network data used in the forecasting model is based on the cumulative social network interactions of software developers. The SourceForge database provides monthly snapshots of projects hosted at the SourceForge website. Each monthly data snapshot includes data associated with the current month and all previous months (from the inception date of a project). Therefore, the forecasting model implicitly tests the effect of cumulative network characteristics on project performance over a short future period of time, i.e., one year in Singh et al. (2011) and Singh (2010). Second, we analyzed the number of CVS commits for each year from 2003 to 2011, and found that the distribution of CVS commits is not uniform over years. For example, a project may perform well for some periods and may not perform well for other periods. Therefore, forecasting for short future time periods based on long

cumulative performance is sensitive choice of network construction date. Therefore, the forecasting model is vulnerable to the arbitrary selection of time for both network structure and project performance. Third, we analyzed the correlations among the cumulative number of CVS commits (a control variable in Singh et al. 2011), and the number of CVS commits in a subsequent year. We found that they are highly correlated, hence, the cumulative number of CVS commits as a control variable may suppress the importance of other independent variables since most of the variation in CVS commits may explained by only one variable. In other words, if a project team has performed well over a long period of time in the past, it is more likely to perform well in the future time period. However, past performance is not a good predictor for performance if future time periods are short.

The second model is a cumulative model used by Grewal et al. (2006). This model is based on the project performance measures over the life span of a project. They measured the dependent variable (CVS Commits) as the number of CVS commits for a project over the life span of a project until network construction date. We used the cumulative model for several reasons. First, both network data and performance data used in the cumulative model are based on the cumulative social network interactions of software developers. Therefore, the cumulative model measures the cumulative performance of resulting developer teams. At any point in time, a social network is the result of cumulative interactions of software developers. It makes sense to measure the cumulative effects of network structures until network construction date. The cumulative model tests the effect of cumulative network characteristics on project performance over a long period of time (compared to the forecasting model). Second, the cumulative model

is not vulnerable to the arbitrary selection of time for network construction date. Third, the cumulative model captures the final results of social network interactions of software developers. Fourth, the cumulative model has been used in prior research (Grewal et al. 2006).

We present multiple technical performance models. With the first set of models, we test the impacts of independent variables on the technical performance of a project measured with the number of CVS commits. With the second set of models, we test the impacts of independent variables on the technical performance of a project measured with the sum of CVS and SVN commits. These models are variants of the following models:

$$\text{CVS} = f(\beta_0 + \beta_1\ \text{Ambi} + \beta_2\ \text{SQ\_Ambi} + \beta_3\ \text{CCoeff} + \beta_4\ \text{RT} + \beta_5\ \text{TPT} + \beta_6\ \text{JS} + \beta_7\ \text{CS} + \beta_8\ \text{EC} + \beta_9\ \text{SQ\_EC} + \beta_{10}\ \text{DT} + \beta_{11}\ \text{IT} + \beta_{12}\ (\text{DT x IT}) + \beta_{13}\ \text{ITFD} + \beta_{14}\ (\text{DT x ITFD}) + \beta_{15}\ \text{TD} + \beta_{16}\ \text{SQ\_TD} + \beta_{17}\ \text{DC} + \beta_{18}\ \text{BC} + \beta_{19}\ \text{CC} + \beta_{20}\ \text{Ambi\_DC} + \beta_{21}\ \text{Ambi\_BC} + \beta_{22}\ \text{Ambi\_CC} + \beta_{23}\ \text{NP} + \beta_{24}\ (\text{Ambi\_DC x NP}) + \beta_{25}\ (\text{Ambi\_BC x NP}) + \beta_{26}\ (\text{Ambi\_CC x NP}))$$

$$\text{CVS and SVN} = f(\beta_0 + \beta_1\ \text{Ambi} + \beta_2\ \text{SQ\_Ambi} + \beta_3\ \text{CCoeff} + \beta_4\ \text{RT} + \beta_5\ \text{TPT} + \beta_6\ \text{JS} + \beta_7\ \text{CS} + \beta_8\ \text{EC} + \beta_9\ \text{SQ\_EC} + \beta_{10}\ \text{DT} + \beta_{11}\ \text{IT} + \beta_{12}\ (\text{DT x IT}) + \beta_{13}\ \text{ITFD} + \beta_{14}\ (\text{DT x ITFD}) + \beta_{15}\ \text{TD} + \beta_{16}\ \text{SQ\_TD} + \beta_{17}\ \text{DC} + \beta_{18}\ \text{BC} + \beta_{19}\ \text{CC} + \beta_{20}\ \text{Ambi\_DC} + \beta_{21}\ \text{Ambi\_BC} + \beta_{22}\ \text{Ambi\_CC} + \beta_{23}\ \text{NP} + \beta_{24}\ (\text{Ambi\_DC x NP}) + \beta_{25}\ (\text{Ambi\_BC x NP}) + \beta_{26}\ (\text{Ambi\_CC x NP}))$$

where the CVS is the dependent variable of the technical performance model which is measures as the number of CVS commits. The CVS and SVN is the another dependent variable of the technical performance model which is measures as the sum of CVS and SVN commits. The Ambi is project ambidexterity, the SQ_Ambi is the square

of project ambidexterity, the CCoeff is clustering coefficient for a project, the RT is the number of repeat ties for a project, the TPT is the number of third part ties for a project, the JS is Jaccard similarity for a project, the CS is correlation similarity for a project, the EC is the external cohesion for a project, SQ_EC is the square of external cohesion, the DT is the number of direct ties for a project, the IT is the number of indirect ties for a project, the (DT x IT) is the interaction term between the number of direct ties and indirect ties, the ITFD is the number of indirect ties calculated with frequency decay function, the (DT x ITFD) the interaction term between the number of direct ties and frequency decayed indirect ties, the TD is the technological diversity of a project, the SQ_TD is the square of technological diversity, the DC is the degree centrality of a project, the BC is the betweenness centrality of a project, the CC is the closeness centrality of a project, the Ambi_DC is the degree centrality of ambidextrous developers for a project, the Ambi_BC the betweenness centrality of ambidextrous developers for a project, the Ambi_CC is the closeness centrality of ambidextrous developers for a project, the NP is the number of projects on which ambidextrous developers work , the (Ambi_DC x NP) is the interaction term between the degree centrality of ambidextrous developers and the number of projects, the (Ambi_BC x NP) is the interaction term between the betweenness centrality of ambidextrous developers and the number of projects, the (Ambi_CC x NP) is the interaction term between the closeness centrality of ambidextrous developers and the number of projects. In Table 15, the model variables, their notations, and transformations applied to dependent and independent variables are shown. In Table 16, the descriptive statistics of the untransformed dependent and independent variables are shown.

Another important issue the OLS regression is the absence of multicollinearity among independent variables. High multicollinearity results in reduced stability of the corresponding parameter estimates, increased standard errors associated with coefficients of predictors, and reduced power to measure effects (Cohen et al. 2003, Stevens 1992, Myers 1990). We examined the correlations among independent variables including the squares of independent variables and the interaction terms by using the Pearson Correlation analysis (Allison 1999) and the Variance Inflation Factor (VIF) (Myers 1990, Stevens 1992). Allison (1999) indicated that the cut-off value of a correlation coefficient can be 0.70 for the OLS regression although the correlation coefficient greater than 0.60 may pose difficulties in testing and interpreting regression coefficients (Tabachnick and Fidell 2007). Thus, we set the cut-off value of a correlation coefficient as 0.60. We report correlation coefficients among untransformed independent variables in Table 17, and correlation coefficients among transformed independent variables in Table 18. Pearson Correlation analysis indicates statistically significant correlation among some independent variables including the squares of independent variables and the interaction terms. Therefore, we tested the seriousness of high correlations with the VIF. The VIF values greater than 10 for the OLS regression indicate high multicollinearity problem (Myers 1990, Stevens 2002).

We found that the squares of ambidexterity, external cohesion, technological diversity, and project age were highly correlated with ambidexterity, external cohesion, technological diversity, and project age respectively. Aiken and West (1991) and Myers (1990) suggested that when the squared variables are included in the model, the independent variables should be centered in order to reduce the correlation between them

to acceptable levels. Therefore, we mean centered ambidexterity, external cohesion, technological diversity, and project age before taking their squares. Our approach is also consistent with Singh et al. (2011) since they found a curvilinear relationship for external cohesion, technological diversity, and project age. They included the squares of non-linear variables (external cohesion, technological diversity, and project age), and mean centered non-linear variables before taking their squares. After mean centering, the VIF values for ambidexterity, external cohesion, technological diversity and project age, and their squares are lower than 10. Thus, the VIF analysis does not indicate any multicollinearity problems for these variables.

We also found that ambidextrous developers' degree, betweenness and closeness centralities, and the number of projects are highly correlated with their interaction terms. Aiken and West (1991) and Myers (1990) suggested that when the interaction terms are included in the model, the independent variables should be centered in order to reduce the correlation between them to acceptable levels. Therefore, we mean centered ambidextrous developers' degree, betweenness and closeness centralities before calculating their interaction terms with the number of projects. However, after mean centering, the VIF values for ambidextrous developers' degree, betweenness and closeness centralities, the number of projects, and their interaction terms are found higher than 10. Therefore, the VIF analysis still indicates multicollinearity problems. We chose to report results with multicollinearity problems.

TABLE 15: Transformations Applied to Dependent and Independent Variables

Variable Type	Variable Name	Notation	Transformations Applied to Key Variables
Dependent Variables			
Technical Performance	CVS Commits	CVS	Log transformed
	CVS and SVN Commits	CVS and SVN	Log transformed
Independent Variables			
Ambidexterity	Ambidexterity	Ambi	Mean centered
	Ambidexterity Squared	SQ_Ambi	Square of Mean-centered Ambidexterity
Internal Cohesion	Clustering Coefficient	CCoeff	No transformation
	Repeat Ties	RT	Log transformed
	Third Party Ties	TPT	Log transformed
	Jaccard Similarity	JS	No transformation
	Correlation Similarity	CS	No transformation
External Connectivity	External Cohesion	EC	Mean centered
	External Cohesion Squared	SQ_EC	Square of Mean-centered External Cohesion
	Direct Ties	DT	Log transformed and scaled down by a factor of 100
	Indirect Ties	IT	Log transformed and scaled down by a factor of 100
	Direct Ties x Indirect Ties Term	DT x IT	Product of Log transformed DT and IT
	Indirect Ties FD	ITFD	Log transformed and scaled down by a factor of 100
	Direct Ties x Indirect Ties FD	DT x ITFD	Product of Log transformed DT and ITFD
	Technological Diversity	TD	Mean centered
	Technological Diversity Squared	SQ_TD	Square of Mean-centered Technological Diversity
Network Location	Degree Centrality	DC	Mean centered
	Betweenness Centrality	BC	Mean centered
	Closeness Centrality	CC	Mean centered
Network Location of Ambidextrous Developers	Ambi Degree Centrality	Ambi_DC	Mean centered
	Ambi Betweenness Centrality	Ambi_BC	Mean centered
	Ambi Closeness Centrality	Ambi_CC	Mean centered
	Number of Projects	NP	Log transformed
	Ambi DC x NP Term	Ambi_DC x NP	Product of Mean-centered Ambi DC and Log transformed NP
	Ambi BC x NP Term	Ambi_BC x NP	Product of Mean-centered Ambi BC and Log transformed NP
	Ambi CC x NP Term	Ambi_CC x NP	Product of Mean-centered Ambi CC and Log transformed NP

TABLE 16: Descriptive Statistics of Untransformed Dependent and Independent Variables (N=2360)

Variable Type	Variable Name	Mean	Std. Dev.	Std. Error Mean
Dependent Variables				
Technical Performance	CVS Commits	802.902	2,466.375	50.770
	CVS and SVN Commits	929.130	2,608.244	53.690
Independent Variables				
Ambidexterity	Ambidexterity	0.254	0.357	0.007
	Ambidexterity Squared	0.192	0.333	0.007
Internal Cohesion	Clustering Coefficient	0.597	0.448	0.009
	Repeat Ties	0.804	0.490	0.010
	Third Party Ties	0.331	3.812	0.078
	Jaccard Similarity	0.498	0.445	0.009
	Correlation Similarity	0.656	0.425	0.009
External Connectivity	External Cohesion	0.511	0.390	0.008
	External Cohesion Squared	0.413	0.436	0.009
	Direct Ties	7.463	12.315	0.254
	Indirect Ties	671.736	1,149.184	23.656
	Direct Ties x Indirect Ties Term	12,104.947	31,896.333	656.576
	Indirect Ties FD	286.799	506.890	10.434
	Direct Ties x Indirect Ties FD	5,522.912	15,346.078	315.894
	Technological Diversity	0.214	0.271	0.006
	Technological Diversity Squared	0.119	0.170	0.003
Network Location	Degree Centrality	0.05893	0.09724	0.00200
	Betweenness Centrality	0.01633	0.06845	0.00141
	Closeness Centrality	0.00929	0.00238	0.00005
Network Location of Ambidextrous Developers	Ambi Degree Centrality	0.05678	0.12812	0.00264
	Ambi Betweenness Centrality	0.03273	0.12815	0.00264
	Ambi Closeness Centrality	0.00465	0.00557	0.00011
	Number of Projects	0.848	1.275	0.026
	Ambi DC x NP	0.15566	0.57760	0.01189
	Ambi BC x NP	0.11494	0.56197	0.01157
	Ambi CC x NP	0.00969	0.01642	0.00034

TABLE 17: Pearson Correlations among Untransformed Independent Variables (N=2360)

No	Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	Ambi	1	.965	.113	.099	.065	-.030	-.069	-.061	-.112	.189	.281	.197	.294	.208	.348	.316	.189	.178	.283	.139	.197	.240	.094	.713	.524	.139	.074	.437	.171	.050
2	SQ_Ambi	.965	1	.045	.009	.053	-.083	-.111	-.066	-.093	.131	.183	.138	.199	.155	.258	.237	.131	.130	.184	.122	.138	.145	.031	.558	.395	.076	.020	.316	.090	.006
3	CCoeff	.113	.045	1	.516	.038	.899	.572	.217	.068	.361	.275	.200	.258	.182	.310	.266	.361	.061	.273	.036	.302	.243	.092	.291	.190	.115	.060	.183	.233	.372
4	RT	.099	.009	.516	1	.201	.517	.732	.496	.353	.325	.270	.236	.266	.233	.292	.266	.325	.122	.268	.090	.296	.265	.138	.285	.308	.256	.147	.318	.245	.255
5	TPT	.065	.053	.038	.201	1	.004	-.004	-.083	-.076	.484	.123	.476	.149	.480	.106	.095	.484	.095	.127	.128	.501	.447	.061	.114	.212	.359	.094	.226	.135	.044
6	JS	-.030	-.083	.899	.517	.004	1	.717	.252	.145	.244	.059	.072	.035	.047	.062	.052	.244	-.061	.054	-.042	.177	.119	.001	.114	.003	.013	-.020	-.003	.204	.410
7	CS	-.069	-.111	.572	.732	-.004	.717	1	.625	.527	.136	-.042	.003	-.062	-.017	-.078	-.073	.136	-.099	-.048	-.066	.082	.041	-.038	.030	-.079	-.027	-.049	-.082	.138	.315
8	EC	-.061	-.066	.217	.496	-.083	.252	.625	1	.962	-.275	-.246	-.264	-.259	-.261	-.194	-.169	-.275	-.150	-.247	-.117	-.277	-.254	-.157	-.165	-.162	-.169	-.123	-.191	-.241	-.211
9	SQ_EC	-.112	-.093	.068	.353	-.076	.145	.527	.962	1	-.326	-.347	-.286	-.349	-.277	-.329	-.290	-.326	-.176	-.347	-.118	-.317	-.294	-.186	-.266	-.247	-.191	-.147	-.270	-.265	-.250
10	DT	.189	.131	.361	.325	.484	.244	.136	-.275	-.326	1	.501	.902	.542	.884	.426	.360	1.000	.322	.504	.341	.984	.836	.341	.441	.442	.540	.299	.474	.704	.772
11	IT	.281	.183	.275	.270	.123	.059	-.042	-.246	-.347	.501	1	.647	.965	.611	.656	.559	.501	.402	.997	.241	.572	.562	.434	.708	.595	.394	.347	.676	.473	.281
12	DT x IT	.197	.138	.200	.236	.476	.072	.003	-.264	-.286	.902	.647	1	.681	.975	.423	.353	.902	.377	.648	.373	.964	.856	.405	.502	.493	.577	.350	.550	.725	.618
13	ITFD	.294	.199	.258	.266	.149	.035	-.062	-.259	-.349	.542	.965	.681	1	.688	.635	.539	.542	.494	.969	.321	.613	.603	.506	.697	.624	.451	.415	.704	.492	.275
14	DT x ITFD	.208	.155	.182	.233	.480	.047	-.017	-.261	-.277	.884	.611	.975	.688	1	.400	.331	.884	.445	.616	.464	.945	.847	.448	.482	.514	.617	.396	.569	.689	.545
15	TD	.348	.258	.310	.292	.106	.062	-.078	-.194	-.329	.426	.656	.423	.635	.400	1	.966	.426	.246	.657	.144	.434	.438	.260	.607	.565	.287	.208	.549	.374	.250
16	SQ_TD	.316	.237	.266	.266	.095	.052	-.073	-.169	-.290	.360	.559	.353	.539	.331	.966	1	.360	.188	.560	.106	.365	.365	.195	.528	.488	.236	.155	.467	.311	.209
17	DC	.189	.131	.361	.325	.484	.244	.136	-.275	-.326	1.000	.501	.902	.542	.884	.426	.360	1	.322	.504	.341	.984	.836	.341	.441	.442	.540	.299	.474	.704	.772
18	BC	.178	.130	.061	.122	.095	-.061	-.099	-.150	-.176	.322	.402	.377	.494	.445	.246	.188	.322	1	.409	.816	.354	.375	.754	.343	.428	.358	.643	.469	.222	.060
19	CC	.283	.184	.273	.268	.127	.054	-.048	-.247	-.347	.504	.997	.648	.969	.616	.657	.560	.504	.409	1	.246	.575	.564	.438	.709	.598	.398	.350	.679	.468	.277
20	DC x BC	.139	.122	.036	.090	.128	-.042	-.066	-.117	-.118	.341	.241	.373	.321	.464	.144	.106	.341	.816	.246	1	.365	.350	.564	.215	.309	.351	.479	.335	.191	.073
21	DC x CC	.197	.138	.302	.296	.501	.177	.082	-.277	-.317	.984	.572	.964	.613	.945	.434	.365	.984	.354	.575	.365	1	.865	.376	.477	.474	.570	.328	.517	.724	.720
22	Ambi_DC	.240	.145	.243	.265	.447	.119	.041	-.254	-.294	.836	.562	.856	.603	.847	.438	.365	.836	.375	.564	.350	.865	1	.555	.611	.659	.747	.525	.686	.788	.574
23	Ambi_BC	.094	.031	.092	.138	.061	.001	-.038	-.157	-.186	.341	.434	.405	.506	.448	.260	.195	.341	.754	.438	.564	.376	.555	1	.400	.534	.530	.892	.580	.436	.162
24	Ambi_CC	.713	.558	.291	.285	.114	.114	.030	-.165	-.266	.441	.708	.502	.697	.482	.607	.528	.441	.343	.709	.215	.477	.611	.400	1	.810	.395	.320	.782	.531	.275
25	NP	.524	.395	.190	.308	.212	.003	-.079	-.162	-.247	.442	.595	.493	.624	.514	.565	.488	.442	.428	.598	.309	.474	.659	.534	.810	1	.719	.588	.980	.547	.152
26	Ambi_DC x NP	.139	.076	.115	.256	.359	.013	-.027	-.169	-.191	.540	.394	.577	.451	.617	.287	.236	.540	.358	.398	.351	.570	.747	.530	.395	.719	1	.740	.758	.548	.223
27	Ambi_BC x NP	.074	.020	.060	.147	.094	-.020	-.049	-.123	-.147	.299	.347	.350	.415	.396	.208	.155	.299	.643	.350	.479	.328	.525	.892	.320	.588	.740	1	.631	.386	.096
28	Ambi_CC x NP	.437	.316	.183	.318	.226	-.003	-.082	-.191	-.270	.474	.676	.550	.704	.569	.549	.467	.474	.469	.679	.335	.517	.686	.580	.782	.980	.758	.631	1	.562	.165
29	Team Size	.171	.090	.233	.245	.135	.204	.138	-.241	-.265	.704	.473	.725	.492	.689	.374	.311	.704	.222	.468	.191	.724	.788	.436	.531	.547	.548	.386	.562	1	.740
30	NP x Team Size	.050	.006	.372	.255	.044	.410	.315	-.211	-.250	.772	.281	.618	.275	.545	.250	.209	.772	.060	.277	.073	.720	.574	.162	.275	.152	.223	.096	.165	.740	1

TABLE 18: Pearson Correlations among Transformed Independent Variables (N=2360) (Variables are Transformed as in Table 15)

No	Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	Ambi	1	.848	.113	.099	.065	-.030	-.069	-.061	-.204	.223	.287	.236	.290	.236	.348	.202	.189	.178	.283	.111	.118	.240	.094	.713	.672	.088	.026	.486	.199	.050
2	SQ_Ambi	.848	1	-.033	-.091	.035	-.134	-.147	-.066	-.012	.065	.062	.075	.069	.081	.138	.098	.058	.068	.061	.091	.054	.029	-.040	.344	.318	-.050	-.073	.176	.016	-.043
3	CCoeff	.113	-.033	1	.516	.038	.899	.572	.217	-.473	.431	.283	.234	.269	.219	.310	.141	.361	.061	.273	-.002	.051	.243	.092	.291	.245	.157	.056	.233	.255	.372
4	RT	.099	-.091	.516	1	.201	.517	.732	.496	-.365	.377	.276	.268	.269	.262	.292	.173	.325	.122	.268	.052	.128	.265	.138	.285	.290	.257	.122	.308	.221	.255
5	TPT	.065	.035	.038	.201	1	.004	-.004	-.083	-.001	.400	.126	.409	.136	.412	.106	.059	.484	.095	.127	.090	.536	.447	.061	.114	.154	.458	.067	.184	.098	.044
6	JS	-.030	-.134	.899	.517	.004	1	.717	.252	-.307	.294	.062	.079	.047	.064	.062	.024	.244	-.061	.054	-.058	-.016	.119	.001	.114	.052	.053	-.017	.033	.238	.410
7	CS	-.069	-.147	.572	.732	-.004	.717	1	.625	-.158	.166	-.042	.000	-.054	-.012	-.078	-.051	.136	-.099	-.048	-.069	-.036	.041	-.038	.030	-.037	.002	-.045	-.056	.166	.315
8	EC	-.061	-.066	.217	.496	-.083	.252	.625	1	.175	-.299	-.251	-.298	-.253	-.290	-.194	-.094	-.275	-.150	-.247	-.082	-.201	-.254	-.157	-.165	-.148	-.223	-.134	-.218	-.240	-.211
9	SQ_EC	-.204	-.012	-.473	-.365	-.001	-.307	-.158	.175	1	-.350	-.454	-.214	-.432	-.199	-.549	-.298	-.270	-.139	-.441	.007	.043	-.222	-.154	-.418	-.407	-.134	-.111	-.398	-.165	-.207
10	DT	.223	.065	.431	.377	.400	.294	.166	-.299	-.350	1	.586	.878	.594	.864	.496	.218	.974	.362	.576	.267	.719	.819	.387	.504	.484	.716	.335	.567	.712	.742
11	IT	.287	.062	.283	.276	.126	.062	-.042	-.251	-.454	.586	1	.750	.993	.733	.676	.301	.513	.406	.998	.157	.390	.569	.436	.714	.620	.471	.359	.809	.454	.288
12	DT x IT	.236	.075	.234	.268	.409	.079	.000	-.298	-.214	.878	.750	1	.767	.995	.502	.204	.874	.433	.749	.311	.870	.849	.464	.578	.535	.764	.404	.685	.710	.566
13	ITFD	.290	.069	.269	.269	.136	.047	-.054	-.253	-.432	.594	.993	.767	1	.759	.654	.285	.522	.442	.995	.183	.419	.582	.466	.710	.621	.494	.390	.816	.454	.279
14	DT x ITFD	.236	.081	.219	.262	.412	.064	-.012	-.290	-.199	.864	.733	.995	.759	1	.480	.191	.859	.458	.734	.345	.875	.843	.481	.566	.531	.773	.423	.682	.687	.530
15	TD	.348	.138	.310	.292	.106	.062	-.078	-.194	-.549	.496	.676	.502	.654	.480	1	.735	.426	.246	.657	.080	.215	.438	.260	.607	.615	.319	.190	.614	.371	.250
16	SQ_TD	.202	.098	.141	.173	.059	.024	-.051	-.094	-.298	.218	.301	.204	.285	.191	.735	1	.180	.051	.289	-.003	.065	.172	.044	.294	.304	.105	.011	.272	.151	.100
17	DC	.189	.058	.361	.325	.484	.244	.136	-.275	-.270	.974	.513	.874	.522	.859	.426	.180	1	.322	.504	.248	.810	.836	.341	.441	.423	.720	.297	.499	.732	.772
18	BC	.178	.068	.061	.122	.095	-.061	-.099	-.150	-.139	.362	.406	.433	.442	.458	.246	.051	.322	1	.409	.727	.301	.375	.754	.343	.380	.390	.720	.473	.178	.060
19	CC	.283	.061	.273	.268	.127	.054	-.048	-.247	-.441	.576	.998	.749	.995	.734	.657	.289	.504	.409	1	.159	.396	.564	.438	.709	.612	.469	.361	.807	.446	.277
20	DC x BC	.111	.091	-.002	.052	.090	-.058	-.069	-.082	.007	.267	.157	.311	.183	.345	.080	-.003	.248	.727	.159	1	.289	.262	.478	.143	.184	.315	.456	.227	.081	.006
21	DC x CC	.118	.054	.051	.128	.536	-.016	-.036	-.201	.043	.719	.390	.870	.419	.875	.215	.065	.810	.301	.396	.289	1	.778	.318	.313	.302	.710	.288	.399	.656	.525
22	Ambi_DC	.240	.029	.243	.265	.447	.119	.041	-.254	-.222	.819	.569	.849	.582	.843	.438	.172	.836	.375	.564	.262	.778	1	.555	.611	.628	.915	.508	.698	.771	.574
23	Ambi_BC	.094	-.040	.092	.138	.061	.001	-.038	-.157	-.154	.387	.436	.464	.466	.481	.260	.044	.341	.754	.438	.478	.318	.555	1	.400	.467	.595	.977	.571	.357	.162
24	Ambi_CC	.713	.344	.291	.285	.114	.114	.030	-.165	-.418	.504	.714	.578	.710	.566	.607	.294	.441	.343	.709	.143	.313	.611	.400	1	.935	.423	.296	.892	.535	.275
25	NP	.672	.318	.245	.290	.154	.052	-.037	-.148	-.407	.484	.620	.535	.621	.531	.615	.304	.423	.380	.612	.184	.302	.628	.467	.935	1	.512	.387	.912	.494	.197
26	Ambi_DC x NP	.088	-.050	.157	.257	.458	.053	.002	-.223	-.134	.716	.471	.764	.494	.773	.319	.105	.720	.390	.469	.315	.710	.915	.595	.423	.512	1	.609	.626	.597	.395
27	Ambi_BC x NP	.026	-.073	.056	.122	.067	-.017	-.045	-.134	-.111	.335	.359	.404	.390	.423	.190	.011	.297	.720	.361	.456	.288	.508	.977	.296	.387	.609	1	.499	.288	.115
28	Ambi_CC x NP	.486	.176	.233	.308	.184	.033	-.056	-.218	-.398	.567	.809	.685	.816	.682	.614	.272	.499	.473	.807	.227	.399	.698	.571	.892	.912	.626	.499	1	.516	.229
29	Team Size	.199	.016	.255	.221	.098	.238	.166	-.240	-.165	.712	.454	.710	.454	.687	.371	.151	.732	.178	.446	.081	.656	.771	.357	.535	.494	.597	.288	.516	1	.834
30	NP x Team Size	.050	-.043	.372	.255	.044	.410	.315	-.211	-.207	.742	.288	.566	.279	.530	.250	.100	.772	.060	.277	.006	.525	.574	.162	.275	.197	.395	.115	.229	.834	1

We found that the pair of log transformed direct ties and indirect ties, and the pair of log transformed direct ties and frequency decayed indirect ties were highly correlated with their interaction terms. However, the VIF values for direct ties, indirect ties, frequency decayed indirect ties, and their interaction terms are lower than 10. Thus, the VIF analysis does not indicate any multicollinearity problems, and we did not use any additional transformation such as mean centered transformation for them.

The correlation analysis also indicates the high correlation among variables within and between variable groups (internal cohesion, external connectivity, network location of projects, and network location of ambidextrous developers). For example, there are high correlations among the pairs of internal cohesion variables: clustering coefficient and Jaccard similarities, correlation similarities and repeat ties, correlation similarities and Jaccard similarities. There are high correlations among the pairs of external connectivity variables: indirect ties and frequency decayed indirect ties, indirect ties and technological diversity. External connectivity variable are also correlated with centrality variables of projects and centrality variables of ambidextrous developers. For example, there are high correlations among the pairs of following variables: direct ties and projects' degree centrality, direct ties and ambidextrous developers' degree centrality, indirect ties and projects' closeness centrality, indirect ties and ambidextrous developers' closeness centrality, indirect ties and the number of projects, frequency decayed indirect ties and projects' closeness centrality, frequency decayed indirect ties and ambidextrous developers' closeness centrality, frequency decayed indirect ties and the number of projects, technological diversity and projects' closeness centrality, technological diversity and ambidextrous developers' closeness centrality, technological diversity and the

number of projects. In addition, centrality variables of projects and centrality variables of ambidextrous developers are also correlated with each other. For example, there are high correlations among the pairs of following variables: projects' degree centrality and ambidextrous developers' degree centrality, projects' betweenness centrality and ambidextrous developers' betweenness centrality, projects' closeness centrality and ambidextrous developers' closeness centrality. Because of high correlations among independent variables within and between variable groups, we cannot test all variables in one model. Therefore, we tested each independent variable along with ambidexterity in separate models. We also created illustrative combined models in order to show that further combined models are possible. However, there is no basis for which an independent variable should be used for a representative for each variable group (internal cohesion, external connectivity, network location of projects, and network location of ambidextrous developers). Prior studies use one variable for internal cohesion as well as one variable for external connectivity to test their hypotheses (Singh et al. 2011). Consistent with prior studies, we select one variable for internal cohesion as well as one variable for external connectivity along with ambidexterity to create our illustrative combined models.

In the data analysis, we found that projects' degree, betweenness and closeness centralities were not statistically significant. The results are contrary to our expectations and do not support our hypotheses regarding projects' network locations. These unexpected results lead us to analyze the interaction terms that are missing in the models. We used backward stepwise regression starting with all two-way interactions in order to reach a final model, in which the interaction of degree centrality with closeness centrality

and the interaction of degree centrality with betweenness centrality were found significant. We analyzed the data set after including the interaction between degree centrality with closeness centrality, and the interaction between degree centrality with betweenness centrality. However, we found that projects' degree, betweenness and closeness centralities were highly correlated with their interaction terms. Aiken and West (1991) and Myers (1990) suggested that when the interaction terms are included in the model, the independent variables should be centered in order to reduce the correlation between them to acceptable levels. Therefore, we mean centered projects' degree, betweenness and closeness centralities before calculating their interaction terms. After mean centering, the VIF values for a projects' degree, betweenness and closeness centralities, and their interaction terms are lower than 10. Thus, the VIF analysis does not indicate any multicollinearity problems for them.

In the data analysis, we also found that project ambidexterity and its square were not statistically significant in testing hypothesis regarding to the network locations of ambidextrous developers. The results are contrary to our expectations for project ambidexterity. These unexpected results lead us to analyze the interaction terms that are missing in the models. We used backward stepwise regression starting with all two-way interactions in order to reach a final model, in which the interaction of the number of projects with a project team size was found significant. We analyzed the data set after including the interaction between the numbers of projects with a project team size.

4.6.1. Technical Performance Models

4.6.1.1. Results of Independent Variables

The significance of an overall regression model is tested with the analysis of variance (Tabachnick and Fidell 2007, Myers 1990). Therefore, the *F* statistic is used to assess the significance of the proposed model against the null model which assumes that that all effects of the independent variables are zero (all regression coefficients are zero). We measured the technical performance of a project using two measures (the number of CVS commits and the sum of CVS and SVN commits). In Table 20, we report the results of regression analyses for technical performance models in which the dependent variable is the number of CVS commits. In Table 21, we report the results of regression analyses for technical performance models in which the dependent variable is the sum of CVS and SVN commits. In each table, Model 1 presents the base model with only ambidexterity and control variables. Model 2.1 through Model 2.5 add internal cohesion measures to Model 1 (clustering coefficient, repeat ties, third party ties, Jaccard similarity, and correlation similarity respectively). Model 3.1 through Model 3.4 add external connectivity measures to Model 1 (external cohesion, direct ties/indirect ties, direct ties/frequency decayed indirect ties, and technological diversity respectively). Model 4 adds projects' centralities measures to Model 1 (degree centrality, betweenness centrality, and closeness centrality together). Model 5 adds ambidextrous developers' centralities measures and the number of projects to Model 1 (the degree centrality of ambidextrous developers, the betweenness centrality of ambidextrous developers, the closeness centrality of ambidextrous developers, and the number of projects together). The *F* statistics of all models across two technical performance measures are significant at the

0.01 alpha level. We rejected the null hypotheses that the effects of the independent variables are zero, and, hence, all models are found to be statistically significant. We summarize the results of our hypotheses in Table 19.

TABLE 19: Summary of Hypotheses

Variable Type	Hypotheses	Tested with Variable	Results	Comments
Ambidexterity	Hypothesis 1	Ambidexterity	Supported	
Internal Cohesion	Hypothesis 2	Clustering Coefficient	Supported	
	Hypothesis 2	Repeat Ties	Supported	
	Hypothesis 2	Third Party Ties	Not Supported	Not significant
	Hypothesis 2	Jaccard Similarity	Supported	
	Hypothesis 2	Correlation Similarity	Supported	
External Connectivity	Hypothesis 3	External Cohesion	Supported	
	Hypothesis 4	Direct Ties	Supported	
	Hypothesis 5	Indirect Ties	Supported	
	Hypothesis 6	Direct Ties x Indirect Ties	Supported	
	Hypothesis 5	Indirect Ties FD	Supported	
	Hypothesis 6	Direct Ties x Indirect Ties FD	Supported	
	Hypothesis 7	Technological Diversity	Supported	
Network Location	Hypothesis 8	Degree Centrality	Supported	
	Hypothesis 8	Betweenness Centrality	Not Supported	Opposite of hypothesis
	Hypothesis 8	Closeness Centrality	Supported	
Network Location of Ambidextrous Developers	Hypothesis 9	Ambi Degree Centrality	Not Supported	Opposite of hypothesis
	Hypothesis 9	Ambi Betweenness Centrality	Supported	
	Hypothesis 9	Ambi Closeness Centrality	Supported	
	Hypothesis 10	Ambi DC x NP	Not Supported	Opposite of hypothesis
	Hypothesis 10	Ambi BC x NP	Supported	
	Hypothesis 10	Ambi CC x NP	Supported	

TABLE 20: Results of Technical Performance Model (Dependent Variable: CVS Commits, N=2360)

	CVS Model 1	CVS Model 2.1	CVS Model 2.2	CVS Model 2.3	CVS Model 2.4	CVS Model 2.5	CVS Model 3.1	CVS Model 3.2	CVS Model 3.3	CVS Model 3.4	CVS Model 4	CVS Model 5
Independent Variables												
Ambidexterity												
Ambidexterity	5.143 ***	4.092 ***	3.540 ***	4.937 ***	4.897 ***	5.124 ***	3.437 ***	3.709 ***	3.768 ***	3.439 ***	4.119 ***	-2.372 **
Ambidexterity Squared	-4.850 ***	-3.752 ***	-3.029 ***	-4.745 ***	-4.046 ***	-4.322 ***	-3.273 ***	-3.749 ***	-3.813 ***	-3.697 ***	-4.115 ***	.818
Internal Cohesion												
Clustering Coefficient		5.677 ***										
Repeat Ties			5.244 ***									
Third Party Ties				0.959								
Jaccard Similarity					5.857 ***							
Correlation Similarity						5.326 ***						
External Connectivity												
External Cohesion							2.079 **					
External Cohesion Squared							-4.731 ***					
Direct Ties								3.913 ***	3.638 ***			
Indirect Ties								2.724 ***				
Direct x Indirect Term								-4.031 ***				
Indirect Ties FD									2.269 **			
Direct x Indirect Ties FD									-3.626 ***			
Tech. Diversity										3.437 ***		
Tech. Diversity Squared										-2.625 ***		
Network Location												
Degree Centrality											2.804 ***	
Betweenness Centrality											-2.494 **	
Closeness Centrality											1.737 *	
DC x BC											2.021 **	
DC x CC											-3.861 ***	

TABLE 20: Cont'd

	CVS Model 1	CVS Model 2.1	CVS Model 2.2	CVS Model 2.3	CVS Model 2.4	CVS Model 2.5	CVS Model 3.1	CVS Model 3.2	CVS Model 3.3	CVS Model 3.4	CVS Model 4	CVS Model 5
Network Location												
Ambi Degree Centrality												-2.205 **
Ambi Betweenness Centrality												2.284 **
Ambi Closeness Centrality												3.96 ***
Number of Projects												1.935 **
Ambi DC x NP												2.541 **
Ambi BC x NP												-2.366 **
Ambi CC x NP												-3.024 **
NP x Project Team Size												-3.428 **
Control Variables												
Team Human Capital and Ability												
Project Team Size	7.981 ***	6.299 ***	7.080 ***	7.939 ***	5.936 ***	6.705 ***	7.843 ***	3.869 ***	3.563 ***	7.085 ***	3.719 ***	6.707 ***
User Input and Market Potential												
Bugs Closed	10.277 ***	9.573 ***	9.536 ***	10.245 ***	9.206 ***	9.433 ***	9.873 ***	9.642 ***	9.694 ***	10.315 ***	9.678 ***	9.638 ***
Support Requests Answered	-.675	-.659	-.430	-.609	-.786	-.693	-.600	-.535	-.511	-.523	-.604	-.450
Page Views	4.827 ***	4.586 ***	4.986 ***	4.865 ***	4.514 ***	4.750 ***	4.821 ***	4.793 ***	4.813 ***	4.879 ***	4.755 ***	4.703 ***
Project Life-Cycle Effects												
Project Age (in months)	27.532 ***	27.230 ***	27.235 ***	27.523 ***	27.576 ***	27.480 ***	27.030 ***	27.005 ***	27.069 ***	27.002 ***	27.146 ***	26.949 ***
Project Age Squared	6.598 ***	6.453 ***	6.656 ***	6.599 ***	6.468 ***	6.570 ***	6.542 ***	6.529 ***	6.551 ***	6.464 ***	6.547 ***	6.473 ***
Development Status	-1.685 *	-1.613	-1.738 *	-1.737 *	-1.584	-1.539	-1.669 *	-1.729 *	-1.740 *	-1.745 *	-1.738 *	-1.629
English	2.668 ***	2.667 ***	2.799 ***	2.680 ***	2.634 ***	2.703 ***	2.717 ***	2.750 ***	2.723 ***	2.752 ***	2.598 ***	2.915 ***
Results												
F statistics	46.217 ***	46.500 ***	46.298 ***	45.162 ***	46.589 ***	46.335 ***	45.092 ***	43.398 ***	43.765 ***	44.573 ***	42.056 ***	40.129 ***
Degree of Freedom	42	43	43	43	43	43	44	45	45	44	47	50
R	0.675	0.681	0.680	0.675	0.681	0.680	0.679	0.679	0.678	0.677	0.679	0.682
R Square	0.456	0.463	0.462	0.456	0.464	0.462	0.462	0.461	0.460	0.459	0.461	0.465
Adj. R Square	0.446	0.453	0.452	0.446	0.454	0.452	0.451	0.450	0.449	0.448	0.450	0.453
Std. Error of Estimate	2.306	2.291	2.293	2.306	2.290	2.293	2.295	2.298	2.299	2.301	2.298	2.291
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level												

TABLE 21: Results of Technical Performance Model (Dependent Variable: CVS and SVN Commits, N=2360)

	CVS/SVN Model 1	CVS/SVN Model 2.1	CVS/SVN Model 2.2	CVS/SVN Model 2.3	CVS/SVN Model 2.4	CVS/SVN Model 2.5	CVS/SVN Model 3.1	CVS/SVN Model 3.2	CVS/SVN Model 3.3	CVS/SVN Model 3.4	CVS/SVN Model 4	CVS/SVN Model 5
Independent Variables												
Ambidexterity												
Ambidexterity	4.773 ***	3.402 ***	2.624 ***	4.373 ***	4.473 ***	4.758 ***	2.484 **	2.618 ***	2.687 ***	2.664 ***	3.139 ***	-3.080 ***
Ambidexterity Squared	-4.614 ***	-3.179 ***	-2.169 **	-4.395 ***	-3.616 ***	-3.953 ***	-2.492 **	-2.922 ***	-2.995 ***	-3.169 ***	-3.368 ***	1.472
Internal Cohesion												
Clustering Coefficient		7.659 ***										
Repeat Ties			7.385 ***									
Third Party Ties				2.291 **								
Jaccard Similarity					7.439 ***							
Correlation Similarity						6.836 ***						
External Connectivity												
External Cohesion							2.609 ***					
External Cohesion Squared							-6.754 ***					
Direct Ties								5.969 ***	5.848 ***			
Indirect Ties								4.422 ***				
Direct x Indirect Term								-5.975 ***				
Indirect Ties FD									3.908 ***			
Direct x Indirect Ties FD									-5.752 ***			
Tech. Diversity										3.916 ***		
Tech. Diversity Squared										-1.906 *		
Network Location												
Degree Centrality											4.459 ***	
Betweenness Centrality											-2.716 ***	
Closeness Centrality											2.735 ***	
DC x BC											1.966 **	
DC x CC											-5.858 ***	

TABLE 21: Cont'd

	CVS/SVN Model 1	CVS/SVN Model 2.1	CVS/SVN Model 2.2	CVS/SVN Model 2.3	CVS/SVN Model 2.4	CVS/SVN Model 2.5	CVS/SVN Model 3.1	CVS/SVN Model 3.2	CVS/SVN Model 3.3	CVS/SVN Model 3.4	CVS/SVN Model 4	CVS/SVN Model 5
Network Location												
Ambi Degree Centrality												-1.673 *
Ambi Betweenness Centrality												1.416
Ambi Closeness Centrality												3.864 ***
Number of Projects												2.539 **
Ambi DC x NP												2.067 **
Ambi BC x NP												-1.51
Ambi CC x NP												-2.535 **
NP x Project Team Size												-5.447 ***
Control Variables												
Team Human Capital and Ability												
Project Team Size	9.265 ***	7.087 ***	8.063 ***	9.180 ***	6.741 ***	7.680 ***	9.078 ***	3.740 ***	3.259 ***	8.168 ***	3.673 ***	8.912 ***
User Input and Market Potential												
Bugs Closed	13.588 ***	12.715 ***	12.626 ***	13.528 ***	12.284 ***	12.555 ***	13.078 ***	12.729 ***	12.762 ***	13.630 ***	12.777 ***	12.904 ***
Support Requests Answered	-.864	-.846	-.523	-.708	-1.008	-.890	-.765	-.641	-.597	-.683	-.723	-.494
Page Views	6.701 ***	6.414 ***	6.963 ***	6.802 ***	6.336 ***	6.631 ***	6.712 ***	6.699 ***	6.721 ***	6.766 ***	6.653 ***	6.790 ***
Project Life-Cycle Effects												
Project Age (in months)	14.245 ***	13.826 ***	13.835 ***	14.240 ***	14.224 ***	14.125 ***	13.618 ***	13.618 ***	13.700 ***	13.703 ***	13.788 ***	13.616 ***
Project Age Squared	11.857 ***	11.747 ***	12.023 ***	11.871 ***	11.769 ***	11.888 ***	11.846 ***	11.820 ***	11.849 ***	11.713 ***	11.830 ***	11.892 ***
Development Status	-1.229	-1.132	-1.304	-1.357	-1.099	-1.042	-1.218	-1.333	-1.346	-1.352	-1.328	-1.189
English	1.668 *	1.664 *	1.851 *	1.699 *	1.620	1.709 *	1.742 *	1.839 *	1.798 *	1.744 *	1.661 *	1.952 *
Results												
F statistics	32.572 ***	33.970 ***	33.818 ***	31.995 ***	33.848 ***	33.529 ***	32.807 ***	32.074 ***	31.952 ***	31.686 ***	30.611 ***	29.107 ***
Degree of Freedom	42	43	43	43	43	43	44	45	45	44	47	
R	0.609	0.622	0.621	0.610	0.621	0.619	0.620	0.620	0.619	0.613	0.619	0.622
R Square	0.371	0.387	0.386	0.373	0.386	0.384	0.384	0.384	0.383	0.376	0.384	0.387
Adj. R Square	0.360	0.375	0.374	0.361	0.375	0.372	0.372	0.372	0.371	0.364	0.371	0.373
Std. Error of Estimate	2.269	2.241	2.243	2.267	2.243	2.247	2.247	2.247	2.249	2.262	2.249	2.245
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level												

The adjusted R^2 statistics indicate a reasonable fit for all models with the adjusted R^2 around 0.450 for the number of CVS commits, and 0.370 for the sum of CVS and SVN commits. The overall fit of models with the sum of CVS and SVN commits is lower than the overall fit of models with the number of CVS commits. One possible explanation for the relatively lower support for the CVS and SVN commits is that the SVN was not mature enough to fully capture project performance until our network construction date (December 2008) since it started being used around the end of 2006. After the SVN has been available, the adoption of the SVN was widespread throughout the OSS community and many projects have migrated from the CVS to the SVN (Van Antwerp and Madey 2008). Therefore, many projects have used the CVS in their first years and then started using the SVN by migrating from the CVS system to the SVN system. However, the CVS and SVN may not be used in parallel mode because this potentially creates difficulties for projects to track the same changes in different systems. Projects may use the CVS and SVN for different modules of their projects. Projects may still use the CVS to track changes in core project modules which have been developed at the first years of projects and submitted to the CVS system. Projects may not migrate these core modules from the CVS system to the SVN system because of the following reasons. First, there are difficulties to migrate these core modules from the CVS system to the SVN system. Second, there are difficulties to track changes in both systems because of the impact of variety and interdependency of project elements on software development process. In Chapter 2, we analyzed the impact of variety and interdependency of project elements (i.e., software complexity) on the software development process. The variety and interdependency of project elements affect the effectiveness of development teams

(Roberts et al. 2004). The more complex the software product, the more effort required to mitigate the negative impact of software complexity on software development process. As the number of project elements increases, software development becomes more difficult to control. Software complexity also determines the maintenance cost of software since high level of software complexity interferes with the process of comprehending the application and makes it difficult for developers to efficiently and correctly modify the application (Banker and Slaughter 2000). Therefore, migrating core modules from the CVS system to the SVN system and tracking changes in both systems may be very challenging for project teams to effectively manage the interdependencies among project modules. Projects may use the SVN system for non-core modules which may not have interdependency to core project modules. Therefore, the SVN system may be used to track changes which may have relatively lower impact on project performance. Therefore, SVN commits should be analyzed over a long time period with very recent data.

The *t* statistic is used to check the significance of each individual regression coefficients, and hence, to assess the support for the relevant hypotheses. The results of regression coefficients are consistent across two technical performance measures. We found strong support for all of our ten hypotheses across two technical performance measures as shown in Table 19.

Our first hypothesis states that *ambidexterity has a curvilinear effect on project performance, i.e. a moderate level of ambidexterity results in higher project performance rather than very high or very low levels of ambidexterity*. We found that ambidexterity has a curvilinear effect on project performance. The coefficient for ambidexterity is

positive and significant (CVS commits: 5.143, $p < 0.01$; CVS and SVN commits: 4.773, $p < 0.01$) whereas the coefficient for the square of ambidexterity is negative and significant (CVS commits: –4.850, $p < 0.01$; CVS and SVN commits: -4.614, $p < 0.01$). Therefore, our hypothesis *H1* is supported. We propose ambidexterity as a measure of the ability of projects to pursue both exploitative and exploratory activities concurrently. It is based on the concept of ambidextrous developers who contribute to exploitative and exploratory activities concurrently. Ambidextrous developers play important roles for projects. First, they integrate exploitative and exploratory project teams. Second, they facilitate knowledge exchange and combination between exploitative and exploratory project teams, in turn they facilitate new value creation through linking knowledge sources held by exploitative and exploratory project teams. Therefore, their important roles are based on the idea that exploitative and exploratory project teams develop and maintain distinct capabilities and competences. This view is consistent with the idea of task specialization and task variety (Narayanan et al. 2009). Project teams can gain diverse knowledge from different types of tasks since ambidextrous developers work on both exploitative and exploratory activities (Narayanan et al. 2009). In contrast, project teams can gain more and deeper experience from specializing in one task since non-ambidextrous developers work on either exploitative or exploratory activities, and become more familiar with the task (Narayanan et al. 2009). Therefore, ambidextrous developers have access to diverse knowledge from exploitative and exploratory activities, and quickly exchange and integrate greater amounts of knowledge with other project developers. On the other hand, non-ambidextrous developers specialize in either exploitative or exploratory activities, and they may benefit from knowledge exchanged

by ambidextrous developers. Therefore, ambidextrous developers play the same or similar roles in both exploitative and exploratory activities. On the other hand, non-ambidextrous developers play unique roles depending on their specializations on either exploitative or exploratory activities. In addition, their roles are also different from the roles of ambidextrous developers. Therefore, the results show that a moderate level of ambidexterity enables project teams to access diverse knowledge from different types of tasks, and to exchange relevant knowledge within a project team, while ensuring adequate specialization to absorb and integrate new knowledge.

Our second hypothesis states that *the performance of a project will be positively related to the internal cohesion of a project*. We measured internal cohesion for a project with clustering coefficient, repeated ties, third party ties, Jaccard similarity, and correlation similarity. We found support for our second hypothesis for clustering coefficient (CVS commits: 5.677, $p < 0.01$; CVS and SVN commits: 7.659, $p < 0.01$), repeated ties (CVS commits: 5.244, $p < 0.01$; CVS and SVN commits: 7.385, $p < 0.01$), Jaccard similarity (CVS commits: 5.857, $p < 0.01$; CVS and SVN commits: 7.439, $p < 0.01$), and correlation similarity (CVS commits: 5.326, $p < 0.01$; CVS and SVN commits: 6.836, $p < 0.01$). Their coefficients are positive and significant. However, we found that the number of third party ties is insignificant for the CVS measure, but significant for the CVS and SVN measures. The results of repeat ties and third party ties merit further discussion. Repeat ties and third party ties are based on social interactions among developers. One possible explanation for the insignificance of third party ties is that there may be a few social interactions for the pairs of developers with common third parties, and these interactions may not be have enough strength to support third party ties. In

addition, third party ties measure the number of relationship of a pair of developers to common third parties outside the focal project team. Therefore, third party ties do not measure the strong relationship between two developers, but measure the relative relationship of already connected two developers to the common third Thus, they may represent relatively loose connections. The common third developer is an outside developer of a focal team, and thereby, that developer may not directly foster trust, reciprocity norms and shared identity within a focal project team which facilitate collaboration and cooperation among focal project team members. In contrast, repeat ties capture the strength and deepness of the relationship between two developers. The strength and deepness of relationship indicates two developers interact more frequently, and they develop more closer and cohesive relationships. Repeat ties from past interactions also result in greater trust within a focal team. This contributes more to project performance. Therefore, our hypothesis *H2* is supported by results of the clustering coefficient, repeated ties, Jaccard similarity, and correlation similarity.

Our third hypothesis states that *external cohesion has a curvilinear effect on project performance, i.e. a moderate level of external cohesion results in higher project performance rather than very high or very low levels of external cohesion*. We found that external cohesion has a curvilinear effect on project performance. The coefficient for external cohesion is positive and significant (CVS commits: 2.079, $p < 0.01$; CVS and SVN commits: 2.609, $p < 0.01$) whereas the coefficient for the square of external cohesion is negative and significant (CVS commits: –4.731, $p < 0.01$; CVS and SVN commits: -6.754, $p < 0.01$). Therefore, our hypothesis *H3* is supported. Although external cohesion has resource sharing and knowledge spillovers benefits, the impact of external

cohesion on the resource sharing benefits is opposite to the impact of external cohesion on knowledge spillover benefits. From the perspective of resource sharing benefits, a moderate level of external cohesion enables project teams to access to a greater amount of external knowledge by enhancing information transmission capacity of the external network of a project, but does not limit the ability of project teams to access to novel information. From the perspective of knowledge spillovers benefits, a moderate level of external cohesion enables project teams to access to novel information in the forms of information conduits, but does not reduce mutual trust and shared norms which facilitate collaboration and cooperation among developers. Therefore, we found that a moderate level of external cohesion facilitates both the access to and the diversity of external knowledge resources available to a project team.

Our fourth hypothesis states that *the performance of a project will be positively related to the number of direct ties of a project*. Our fifth hypothesis states that *the performance of a project will be positively related to the number of indirect ties of a project*. In the hypothesis *H6*, we expect that *the impact of indirect ties on the performance of a project will be moderated by the number of direct ties of a project, i.e., the greater the number of direct ties, the smaller the benefit from indirect ties*. We used two measures for indirect ties: the number of indirect ties and the number of indirect ties calculated with the frequency decay function. Therefore, direct ties have been tested with each measure of indirect ties in separate models. We found that direct ties have a positive effect on project performance. The coefficient for direct ties is positive and significant (CVS commits: 3.913, $p < 0.01$; CVS and SVN commits: 5.969, $p < 0.01$) when we consider its interaction with the number of indirect ties. The coefficient for direct ties is

also positive and significant (CVS commits: 3.638, $p < 0.01$; CVS and SVN commits: 5.848, $p < 0.01$) when we consider its interaction with the number of frequency decayed indirect ties. We found that indirect ties have a positive effect on project performance. The coefficient for indirect ties is positive and significant (CVS commits: 2.724, $p < 0.01$; CVS and SVN commits: 4.422, $p < 0.01$). The coefficient for frequency decayed indirect ties is also positive and significant (CVS commits: 2.269, $p < 0.01$; CVS and SVN commits: 3.908, $p < 0.01$). Regarding the interaction of indirect ties with direct ties, we found that the impact of indirect ties on project performance is moderated by the number of direct ties. The coefficient for the interaction term of direct and indirect ties is negative and significant (CVS commits: -4.031, $p < 0.01$; CVS and SVN commits: -5.975, $p < 0.01$). The coefficient for the interaction term of direct and frequency decayed indirect ties is negative and significant (CVS commits: -3.626, $p < 0.01$; CVS and SVN commits: -5.752, $p < 0.01$). Although direct ties have both resource sharing and knowledge spillover benefits, the resource sharing benefit of direct ties is greater than knowledge spillover benefit, hence they facilitate resource pooling by enabling project teams to combine knowledge and skills with repeating interactions. Indirect ties provide novel information by enabling project teams to access to knowledge spillovers. However knowledge spillover benefits provided by indirect ties are contingent on the number of direct ties. We found that the ability of project teams to access novel knowledge is constrained by many direct ties. Therefore, project teams with few direct ties enjoy greater knowledge spillovers benefits from their indirect ties than teams with many direct ties.

Our seventh hypothesis states that *technological diversity has a curvilinear effect on project performance, i.e. a moderate level of technological diversity results in higher project performance rather than very high or very low levels of technological diversity*. We found that technological diversity has a curvilinear effect on project performance. The coefficient for technological diversity is positive and significant (CVS commits: 3.437, $p < 0.01$; CVS and SVN commits: 3.916, $p < 0.01$) whereas the coefficient for the square of external cohesion is negative and significant (CVS commits: –2.625, $p < 0.01$; CVS and SVN commits: -1.906, $p < 0.01$). Therefore, our hypothesis *H7* is supported. OSS developer may work on multiple projects concurrently. When they join to another project, they choose to work on projects that are moderately technologically diverse from each other since they can recognize and absorb knowledge close to their existing knowledge base (Cohen and Levinthal 1990). Therefore, a moderate level of technological diversity between two projects improves the performance of developers on each project.

Our eighth hypothesis states that *the performance of a project will be positively related to the centrality of a project*. We measured network location for a project with degree centrality, betweenness centrality, and closeness centrality. In the data analysis, we found that degree centrality has interaction with closeness centrality and betweenness centrality. During the analysis, we considered the interaction of degree centrality with closeness centrality and betweenness centrality. In support for the hypothesis 8, we found that degree centrality has a positive effect on project performance. The coefficient for degree centrality is positive and significant (CVS commits: 2.804, $p < 0.01$; CVS and SVN commits: 4.459, $p < 0.01$) when we consider its interaction with betweenness

centrality and closeness centrality. We also found that closeness centrality has a positive effect on project performance. The coefficient for closeness centrality is positive and significant (CVS commits: 1.737, $p < 0.1$; CVS and SVN commits: 2.735, $p < 0.01$) when we consider its interaction with degree centrality. Therefore, we found support for our eighth hypothesis for degree centrality and closeness centrality. Degree centrality is the measure of how many an actor is connected to other actors in the network through direct connections (Freeman 1979, Wasserman and Frost 1994). If a developer is connected to many other developers through direct ties, a developer may access to greater amounts of information and knowledge (Hansen 2002). However, direct ties more likely provide relatively redundant information (Hansen 1999). Therefore, high degree centrality allows a developer to have access to greater amounts of (relatively redundant) knowledge (Hansen 2002). Therefore, a developer with high degree centrality may have access to greater amounts of (relatively redundant) knowledge. As explained later, the results of our ninth and tenth hypothesis indicates that ambidextrous developers play an integration role. The integration role of ambidextrous developers on project performance depends on access to novel information through indirect ties or from multiple projects: the greater the access to novel information, the higher the impact on project performance. While high degree centrality is undesirable for ambidextrous developers, it could be desirable for a project as a whole. This is because, if every developer has access to greater amounts of knowledge, the project, as a whole, could be positively affected. Closeness centrality is the measure of how close an actor is to all other actors in the network through direct and indirect connections (Freeman 1979, Wasserman and Frost 1994). If a developer is very close to many other developers through direct and indirect ties, a developer may have

quick access to knowledge (Uzzi 1997, Powell and Smith-Doerr 1994). However, indirect ties more likely provide access to novel information (Hansen 1999). Therefore, high closeness centrality may allow a developer to have quick access to both redundant and novel knowledge (Uzzi 1997, Powell and Smith-Doerr 1994). We found that high closeness centrality provides quick access to knowledge, and in turn improves project performance. However, we found that betweenness centrality has a negative effect on project performance. The coefficient for betweenness centrality is negative and significant (CVS commits: -2.494, $p < 0.05$; CVS and SVN commits: -2.716, $p < 0.01$) when we consider its interaction with degree centrality. The effect of betweenness centrality on project performance seems contrary to our expectations. Betweenness centrality is the measure of how often a developer falls on the shortest path between pairs of other developers (Freeman 1979, Wasserman and Faust 1994). Therefore, high between centrality allows a developer to control (Wasserman and Faust 1994, Pfeffer and Salancik 1978), and regulate information flow among other developers (Wasserman and Faust 1994, Krackhardt 1996). Therefore, a developer with high betweenness centrality may control and regulate too much information passing through him. As explained later, the results of our ninth and tenth hypothesis indicates that ambidextrous developers play a control role. The control role of ambidextrous developers on project performance depends on the level of control on information flow among other developers: the greater the level of control on information flow, the higher the impact on project performance. While high betweenness centrality (high control) is desirable for ambidextrous developers, it could be undesirable for a project as a whole. This is because, if every

developer has a high degree of control, the project, as a whole, could be negatively affected.

Regarding the interaction of degree centrality with closeness centrality, we found that the impact of closeness centrality on project performance is moderated by degree centrality. The coefficient for the interaction term of closeness centrality and degree centrality is negative and significant (CVS commits: -3.861, $p < 0.01$; CVS and SVN commits: -5.858, $p < 0.01$). The ability of project teams to have quick access to the novel information is constrained by high degree centrality. Degree centrality considers direct connections whereas closeness centrality considers direct and indirect connections (Freeman 1979, Wasserman and Frost 1994). Project teams with few direct ties add a significant increment to their existing information base through indirect ties (Ahuja 2000). Project teams with many direct ties may be more constrained in their ability to absorb new information (Cohen and Levinthal 1990). Therefore, the effect of closeness centrality of project teams with lower degree centrality is greater than the effect of closeness centrality of project teams with higher degree centrality. The result indicates that project teams with high closeness centrality have quick access to more novel information if their degree centrality is low, which improves project performance. Regarding the interaction of degree centrality with betweenness centrality, we found that the impact of betweenness centrality on project performance is moderated by degree centrality. The coefficient for the interaction term of betweenness centrality and degree centrality is positive and significant (CVS commits: 2.021, $p < 0.05$; CVS and SVN commits: 1.966, $p < 0.05$). The ability of project teams to control and regulate information flow is constrained by degree centrality. Therefore, the effect of betweenness

centrality of project teams with higher degree centrality is greater than the effect of betweenness centrality of project teams with lower degree centrality. The results indicate that project teams can more easily control and regulate information acquired from immediate contacts through direct ties. The results also indicate that information acquired from remote contacts may be more challenging to be controlled and regulated since remote developers more likely provide novel information. Therefore, project teams can more easily control and regulate information acquired from immediate developers than information acquired from remote developers. People vary widely in their capability to develop, understand, or use knowledge based on their technological base and their prior knowledge (Cohen and Levinthal 1990). People recognize and absorb knowledge close to their existing knowledge base (Cohen and Levinthal 1990). Therefore, too much novel information acquired from remote contacts restricts the capability of project teams to develop, understand, or use knowledge. This result is also consistent with the results of technological diversity.

Our ninth hypothesis states that *the performance of a project will be positively related to the centrality of ambidextrous developers*. We measured network location for ambidextrous developers with degree centrality, betweenness centrality, and closeness centrality. We expect the interaction of the number of projects with ambidextrous developers' degree centrality, betweenness centrality, and closeness centrality. We found that ambidextrous developers' betweenness centrality has a positive effect on project performance. The coefficient for ambidextrous developers' betweenness centrality is positive and significant for the dependent variable of CVS commits, but not significant for the dependent variable of CVS and SVN commits (CVS commits: 2.284, $p < 0.05$;

CVS and SVN commits: 1.416, $p < 0.1$) when we consider its interaction with the number of projects. Therefore, we found support for our ninth hypothesis for betweenness centrality. High betweenness centrality allows a developer to control (Wasserman and Faust 1994, Pfeffer and Salancik 1978), and regulate information flow among other developers (Wasserman and Faust 1994, Krackhardt 1996). In our eighth hypothesis, we found that high betweenness centrality of a project negatively affect project performance since project developers do not take advantage to control and regulate information flow among other developers. However, we expect that ambidextrous developers play an important role to control and regulate information flow among other developers. On the other hand, non-ambidextrous developers play unique roles depending on their specializations on either exploitative or exploratory activities. In addition, their roles are also different from the roles of ambidextrous developers. They may benefit from knowledge exchanged by ambidextrous developers. The result of ambidextrous developers' betweenness centrality indicates that ambidextrous developers play an important role to control and regulate information flow among other developers. Combined with the results of our tenth hypothesis, our results indicate that the control role of ambidextrous developers on project performance depends on the level of control on information flow among other developers: the greater the level of control on information flow, the higher the impact on project performance. Therefore, an ambidextrous developer with high betweenness centrality performs better than an ambidextrous developer with low betweenness centrality. We found that ambidextrous developers' closeness centrality has a positive effect on project performance. The coefficient for ambidextrous developers' closeness centrality is positive and significant

(CVS commits: 3.960, $p < 0.1$; CVS and SVN commits: 3.864, $p < 0.01$) when we consider its interaction with the number of projects. Therefore, we found support for our ninth hypothesis for closeness centrality. High closeness centrality allows a developer to have quick access to information (Uzzi 1997, Powell and Smith-Doerr 1994). Moreover, high closeness centrality may allow a developer to have quick access to both redundant and novel knowledge (Uzzi 1997, Powell and Smith-Doerr 1994). We expect that ambidextrous developers play an integration role by speeding up information flow and allowing information to be exchanged and integrated more rapidly among other developers. The result of ambidextrous developers' closeness centrality indicates that ambidextrous developers play an integration role by speeding up information flow and allowing information and knowledge to be exchanged and integrated more rapidly among other developers. Therefore, an ambidextrous developer with high closeness centrality performs better than an ambidextrous developer with low closeness centrality. However, we found that degree centrality has a negative effect on project performance. The coefficient for degree centrality is negative and significant (CVS commits: -2.205, $p < 0.05$; CVS and SVN commits: -1.673, $p < 0.1$) when we consider its interaction with the number of projects. The effect of degree centrality on project performance seems contrary to our expectations. High degree centrality may allow a developer to have access to greater amounts of (relatively redundant) knowledge from immediate contacts (Hansen 2002). We expect that ambidextrous developers play an integration role by allowing greater amounts of information and knowledge to be exchanged and integrated among other developers. The result of ambidextrous developers' degree centrality indicates that ambidextrous developers with high degree centrality may have access to relatively

redundant information from immediate contacts which negatively affects their integration role since most information exchanged by ambidextrous developers are redundant held by all project developers. However, we expect that ambidextrous developers perform well since they have access to diverse knowledge from both exploitative and exploratory activities. Therefore, an ambidextrous developer with low degree centrality performs better than an ambidextrous developer with high degree centrality since they may exchange more novel information with other developers. Combined with the results of our tenth hypothesis, our results indicate that the integration role of ambidextrous developers on project performance depends on access to novel information through indirect ties or from multiple projects: the greater the access to novel information, the higher the impact on project performance.

Our tenth hypothesis states that that *the impact of the centrality of ambidextrous developers on the performance of a project will be moderated by the number of projects on which ambidextrous developers work, i.e., the greater number of projects on which ambidextrous developers work, the lower impact of the centrality of ambidextrous developers on the performance of a project*. We found that the number of projects has a positive effect on project performance. The coefficient for the number of projects is positive and significant (CVS commits: 1.935, $p < 0.05$; CVS and SVN commits: 2.539, $p < 0.01$) when we consider its interaction with ambidextrous developers' degree centrality, betweenness centrality and closeness centrality. The results indicate that ambidextrous developers perform better if they work on many projects since they may access to more novel information from different projects, which improve the integration role of ambidextrous developers by allowing them to exchange more novel information

with other developers. In the data analysis, we also found that the number of projects has interaction with a project team size. When we considered the interaction of the number of projects with a project team size, the coefficient for the interaction term is negative and significant (CVS commits: -3.428, $p < 0.05$; CVS and SVN commits: -5.447, $p < 0.01$). Therefore, the impact of the number of projects on project performance is constrained by a project team size. The results indicate that the impact of the number of projects on project performance decreases if ambidextrous developers work on many large projects since they may spend more time and effort in maintaining connections with many developers (Hansen 1999, Hansen 2002, Shane and Cable 2002). In contrast, if ambidextrous developers work on many small projects, they may not spend too much time and effort in maintaining connections with many developers while they access to more novel information from different projects. In support for the hypothesis 10, we found that the impact of betweenness centrality on project performance is moderated by the number of projects. The coefficient for the interaction term of betweenness centrality and the number of projects is negative and significant (CVS commits: -2.366, $p < 0.05$; CVS and SVN commits: -1.510, $p < 0.05$). Therefore, we found support for tenth hypothesis for betweenness centrality. If ambidextrous developers work on small number of projects, the effect of their betweenness centrality on project performance becomes higher. Regarding the interaction of the number of projects with closeness centrality, we found that the impact of closeness centrality on project performance is moderated by the number of projects. The coefficient for the interaction term of closeness centrality and the number of projects is negative and significant (CVS commits: -3.024, $p < 0.05$; CVS and SVN commits: -2.535, $p < 0.05$). Therefore, we also found support for tenth hypothesis

for closeness centrality. If ambidextrous developers work on small number of projects, the effect of their closeness centrality on project performance becomes higher. Regarding the interaction of the number of projects with degree centrality, the coefficient for the interaction term of degree centrality and the number of projects is positive and significant (CVS commits: 2.541, $p < 0.05$; CVS and SVN commits: 2.067, $p < 0.05$). Although the impact of degree centrality on project performance is moderated by the number of projects, the effect of the interaction term of degree centrality and the number of projects seems contrary to our expectations. In our hypothesis 9, we found that high degree centrality negatively affects the integration role of ambidextrous developers since most information exchanged by ambidextrous developers are redundant held by all project developers. However, we expect that ambidextrous developers perform well since they have access to diverse knowledge from both exploitative and exploratory activities. If ambidextrous developers work on many projects simultaneously, they may access to more novel information from different projects, which improve the integration role of ambidextrous developers by allowing them to exchange more novel information with other developers.

4.6.1.2. Results of Control Variables

Consistent with prior research, we controlled effects of team human capital and ability, user input and market potential, project life-cycle effects on the project technical performance. The results for our control variables are consistent for all models across two technical performance measures, and hence we discuss the results for the base model (Model 1).

Regarding team human capital and ability, we found that a project team size has a positive effect on the project technical performance. The coefficient for a project team size is positive and significant (CVS commits: 7.981, $p < 0.01$; CVS and SVN commits: 9.265, $p < 0.01$). The results showed that projects with large teams perform better than projects with small teams.

Regarding user input and market potential, we found that the number of bugs and page views have a positive effect on the project technical performance. The coefficient for bugs is positive and significant (CVS commits: 10.277, $p < 0.01$; CVS and SVN commits: 13.588, $p < 0.01$). The coefficient for page views is positive and significant (CVS commits: 4.827, $p < 0.01$; CVS and SVN commits: 6.701, $p < 0.01$). However, the number of support requests does not have an effect on the project technical performance. Bugs play an important role to identify defects in software, while support request are associated with specific user questions and offered solutions. Thus, the number of defects detected by users directly affects the project technical performance.

Regarding project life-cycle effects, we found that project age and project language (English) have a positive effect on the project technical performance. The coefficient for a project age is positive and significant (CVS commits: 27.532, $p < 0.01$; CVS and SVN commits: 14.245, $p < 0.01$) and the coefficient for the square of project age is also positive and significant (CVS commits: 6.598, $p < 0.01$; CVS and SVN commits: 11.857, $p < 0.01$). Therefore, the results showed that a project age does not have a curvilinear effect on the project technical performance. We found the development status of software has a negative effect on the project technical performance for some models, but not significant for other models. We may say that project teams may perform

better at the early stages of software development, and their performance decreases when a project becomes stable or mature.

4.6.1.3. Illustrative Combined Models

In the previous section, we presented the results of hypotheses tested with an individual model for each independent variable along with ambidexterity. In this section, we present illustrative combined models in order to show that further combined models are possible. We use the number of CVS commits as the dependent variable for our illustrative combined models consistent with prior research (Singh et al. 2011, Singh 2010, Grewal et al. 2006, Rai et al. 2002). However, there is no basis for which an independent variable should be used as representative for each variable group (internal cohesion, external connectivity, network location of projects, and network location of ambidextrous developers). Prior studies use one variable for internal cohesion as well as one variable for external connectivity to test their hypotheses (Singh et al. 2011). However, we have centrality measures for projects and ambidextrous developers. The correlation analysis indicates the high correlation among variables within and between variable groups as shown in Table 17 and Table 18. For example, there are high correlations among the pairs of internal cohesion variables. There are also high correlations among the pairs of external connectivity variables. External connectivity variable are also correlated with centrality variables of projects and centrality variables of ambidextrous developers. In addition, centrality variables of projects and centrality variables of ambidextrous developers are also correlated with each other. Therefore, we cannot create a combined model which includes external connectivity variables, centrality variables of projects and centrality variables of ambidextrous developers.

However, ambidexterity and internal cohesion variables are not correlated with other variable groups (external connectivity, network location of projects, and network location of ambidextrous developers).

We have 5 internal cohesion variables (clustering coefficient, repeat ties, third party ties, Jaccard similarity, and correlation similarity). We also have 4 external connectivity variables (external cohesion, direct ties/indirect ties, direct ties/frequency decayed indirect ties, and technological diversity). We have 2 groups of centrality variables for projects and ambidextrous developers. First, we included each internal cohesion variable along with ambidexterity and control variables. Second, we created all possible combined models by adding each external connectivity variable and each centrality variable group to ambidexterity, one internal cohesion variable and control variables. Therefore, we ran all possible combined models (30=5*[4+2]). We selected correlation similarity as a representative variable for internal cohesion. In Table 22, we report the results of illustrative combined models which include ambidexterity, correlation similarity and control variables along with each external connectivity variable and each centrality variable group. We report the results of other combined models in Appendix D. In Table 22, Model 1 presents the base model with only ambidexterity and control variables. Model 2.1 through Model 2.4 add correlation similarity and one external connectivity measure (external cohesion, direct ties/indirect ties, direct ties/frequency decayed indirect ties, and technological diversity respectively) to Model 1. Model 3 adds correlation similarity and projects' centralities measures to Model 1 (degree centrality, betweenness centrality, and closeness centrality together). Model 4 adds correlation similarity, ambidextrous developers' centralities measures and the

number of projects to Model 1 (the degree centrality of ambidextrous developers, the betweenness centrality of ambidextrous developers, the closeness centrality of ambidextrous developers, and the number of projects together).

The *F* statistics of all illustrative combined models are significant at the 0.01 alpha level. We rejected the null hypotheses that the effects of the independent variables are zero, and, hence, all models are found to be statistically significant. The adjusted R^2 statistics indicate a reasonable fit for all combined models with the adjusted R^2 ranged from 0.455 to 0.459. However, the contribution of internal cohesion, external connectivity and centrality variables to the base model is marginal considering the adjusted R^2 statistic of the base model (0.446). The results indicate that ambidexterity is significant in almost all illustrative models. Therefore, the results of ambidexterity are stronger than the results of social network measures. We found that the result of each variable in illustrative models is almost consistent with the result of the same variable in individual technical performance models presented in the previous section. In a few cases, the results of some variables in illustrative models are not significant. Therefore, we concluded that the results of individual technical performance models and illustrative models are generally the same. Additional analysis and model development may be possible in future research.

TABLE 22: Results of Illustrative Combined Models for Technical Performance
(Internal Cohesion Measure: Correlation Similarity, Dependent Variable: CVS Commits, N=2360)

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Independent Variables							
Ambidexterity							
Ambidexterity	5.143 ***	4.070 ***	3.399 ***	3.448 ***	2.846 ***	3.740 ***	-2.158 **
Ambidexterity Squared	-4.850 ***	-3.304 ***	-3.049 ***	-3.096 ***	-2.685 ***	-3.358 ***	1.039
Internal Cohesion							
Clustering Coefficient							
Repeat Ties							
Third Party Ties							
Jaccard Similarity							
Correlation Similarity		3.304 ***	5.05 ***	5.164 ***	6.137 ***	5.07 ***	4.864 ***
External Connectivity							
External Cohesion		2.962 ***					
External Cohesion Squared		-2.976 ***					
Direct Ties			3.278 ***	3.067 ***			
Indirect Ties			2.773 ***				
Direct x Indirect Term			-3.003 ***				
Indirect Ties FD				2.428 **			
Direct x Indirect Ties FD				-2.718 ***			
Tech. Diversity					4.581 ***		
Tech. Diversity Squared					-3.095 ***		
Network Location							
Degree Centrality						2.657 ***	
Betweenness Centrality						-2.166 **	
Closeness Centrality						2.256 **	
DC x BC						1.753 *	
DC x CC						-3.090 ***	

TABLE 22: Cont'd

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Network Location							
Ambi Degree Centrality							-1.245
Ambi Betweenness Centrality							2.026 **
Ambi Closeness Centrality							2.866 ***
Number of Projects							2.000 **
Ambi DC x NP							1.612
Ambi BC x NP							-2.125 **
Ambi CC x NP							-2.008 **
NP x Project Team Size							-2.809 ***
Control Variables							
Team Human Capital and Ability							
Project Team Size	7.981 ***	4.171 ***	2.896 ***	2.680 ***	5.328 ***	2.643 ***	5.075 ***
User Input and Market Potential							
Bugs Closed	10.277 ***	8.527 ***	8.964 ***	8.995 ***	9.340 ***	8.989 ***	9.067 ***
Support Requests Answered	-.675	-.730	-.547	-.530	-.486	-.593	-.341
Page Views	4.827 ***	4.448 ***	4.745 ***	4.756 ***	4.806 ***	4.727 ***	4.735 ***
Project Life-Cycle Effects							
Project Age (in months)	27.532 ***	27.210 ***	26.835 ***	26.894 ***	26.779 ***	26.952 ***	26.870 ***
Project Age Squared	6.598 ***	6.449 ***	6.494 ***	6.515 ***	6.381 ***	6.522 ***	6.484 ***
Development Status	-1.685 *	-1.621	-1.641	-1.649 *	-1.619	-1.655 *	-1.566
English	2.668 ***	2.854 ***	2.846 ***	2.831 ***	2.816 ***	2.709 ***	2.993 ***
Results							
F statistics	46.217 ***	45.304 ***	43.953 ***	43.868 ***	45.109 ***	42.155 ***	40.192 ***
Degree of Freedom	42	45	46	46	45	48	51
R	0.675	0.684	0.683	0.683	0.684	0.683	0.686
R Square	0.456	0.468	0.466	0.466	0.467	0.467	0.470
Adj. R Square	0.446	0.458	0.456	0.455	0.457	0.456	0.459
Std. Error of Estimate	2.306	2.281	2.286	2.287	2.283	2.286	2.280
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level							

4.7. Discussions and Contributions

We empirically study the effect of ambidexterity and social network properties of OSS developers on OSS project performance. We also examine the effect of ambidextrous developers who participate in patch development and feature request activities on OSS project performance. We develop technical performance models for OSS projects and measure technical performance (knowledge creation) with two measures. We measure technical performance using the number of CVS commits which is commonly used in the OSS literature (Singh et al. 2011, Singh 2010, Grewal et al. 2006, Rai et al. 2002). We also measure technical performance using the sum of CVS and SVN commits. The overall fit of models with the number of CVS commits is greater than the overall fit of models with the sum of CVS and SVN commits. This could be due to the fact that the SVN was not mature enough to fully capture project performance until our network construction date (December 2008). Therefore, SVN commits should be analyzed over a long time period with very recent data.

We found that ambidexterity has a curvilinear effect on project performance. Our result indicates a balanced pursuit of both exploitative and exploratory activities concurrently has a positive impact on project success. In addition, it shows the importance of different roles and specializations of ambidextrous and non-ambidextrous developers for project success. Ambidextrous developers have access to diverse knowledge from exploitative and exploratory activities, and quickly exchange and integrate greater amounts of knowledge with other project developers. On the other hand, non-ambidextrous developers play unique roles depending on their specializations on

either exploitative or exploratory activities. They also benefit from knowledge exchanged by ambidextrous developers.

Our results illustrate the roles of ambidextrous developers as coordination mechanisms between patch development and feature request activities on project performance. Ambidextrous developers play an integration role by speeding up information flow and allowing information and knowledge to be exchanged and integrated more rapidly among other developers. Our results indicate that the integration role of ambidextrous developers on project performance depends on ambidextrous developers' access to novel information through indirect ties or from multiple projects: the greater the access to novel information, the higher the impact on project performance. Ambidextrous developers also play a control role to control and regulate information flow among other developers. Our results indicate that the control role of ambidextrous developers on project performance depends on the level of control on information flow among other developers: the greater the level of control on information flow, the higher the impact on project performance.

Our results for social network measures are consistent with the findings of prior research on OSS development. However, the results of ambidexterity are stronger than the results of social network measures whose contributions are relatively marginal.

Our results for projects' centrality indicate that project performance is positively related to degree and closeness centrality of a project. However, the result of betweenness centrality of a project is opposite to our expectations since project performance is negatively related to betweenness centrality of a project. Our results for ambidextrous developers' centrality indicate that project performance is positively related to

betweenness and closeness centrality of ambidextrous developers. However, the result of degree centrality of ambidextrous developers is opposite to our expectations since project performance is negatively related to degree centrality of ambidextrous developers. Combined results of projects' centrality and ambidextrous developers' centrality show interesting differences between the effects of ambidextrous developers' centrality and projects' centrality. Contrary to our expectations, degree and betweenness centrality measures do not behave in the same way for projects and ambidextrous developers.

Degree centrality of a project positively affects project performance whereas degree centrality of ambidextrous developers negatively affects project performance. Therefore, while high degree centrality is undesirable for ambidextrous developers, it could be desirable for a project as a whole. Our results for the interaction between degree centrality and the number of projects indicate that the impact of ambidextrous developers' degree centrality on project performance is moderated by the number of projects. If ambidextrous developers work on many projects simultaneously, they may access to more novel information from multiple projects. Therefore, the number of projects positively affects the impact of ambidextrous developers' degree centrality on project performance. The integration role of ambidextrous developers on project performance is facilitated by low degree centrality or access to multiple projects which enables ambidextrous developers to access to novel information.

Betweenness centrality of a project negatively affects project performance whereas betweenness centrality of ambidextrous developers positively affects project performance. Therefore, while high betweenness centrality is desirable for ambidextrous developers, it could be undesirable for a project as a whole. The control role of

ambidextrous developers on project performance is facilitated by high betweenness centrality which enables ambidextrous developers to control and regulate information flow among other developers. Our results indicate that high closeness centrality provides quick access to information, thereby it is desirable for both ambidextrous developers as well as projects.

We found that project performance is positively related to the internal cohesion of a project. As measured by clustering coefficient, repeat ties, Jaccard similarity, and correlation similarity, our findings indicate that different measures of internal cohesion are consistent and have a positive impact on project performance. However, the result of internal cohesion measured by third party ties is not significant. This could be because, although repeat ties and third party ties are based on social interactions among developers, repeat interactions between two developers are much stronger than third party interactions with common third parties since repeat interactions result in greater trust within a focal team.

We found that external cohesion has a curvilinear effect on project performance. A moderate level of external cohesion facilitates both the access to and the diversity of external knowledge resources available to a project team.

We found that project performance is positively related to the number of direct and indirect ties. Our results show that the resource sharing benefit of direct ties is greater than knowledge spillover benefit, hence direct ties facilitate resource pooling by enabling project teams to combine knowledge with repeating interactions. Indirect ties provide novel information by enabling project teams to access to knowledge spillovers. However, we found that knowledge spillovers provided by provided by indirect ties are not equally

accessible to or appropriated by everyone since knowledge spillover benefits provided by indirect ties are contingent on the number of direct ties.

We found that technological diversity has a curvilinear effect on project performance. Our results indicate that when developers join to another project, they perform better if they work on projects that are moderately technologically diverse from each other since they can recognize and absorb knowledge close to their existing knowledge base. Therefore, OSS developers who work on multiple projects simultaneously should choose work on projects that are moderately technologically diverse from one another. OSS project leaders should encourage developers to work on projects that are moderately technologically diverse.

By providing a more comprehensive understanding of the effects of ambidexterity and social network structure of OSS developers combined with the effect of coordination mechanisms (ambidextrous developers) on project performance, this dissertation makes several important theoretical and practical contributions.

From a theoretical perspective, we develop the theory for and then empirically test how ambidexterity affects project performance. Recent research on OSS development has studied the social network structure of software developers as determinant of project success (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006). However, this stream of research has focused on the project level, and has not recognized the fact that projects could consist of different types of activities, each of which could require different types of expertise. We propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration) based on organizational theory (March 1991). We identified a new category

of developers (ambidextrous developers) in OSS projects who contribute to exploitative activities (patch development) and exploratory activities (feature request). We develop a theoretical construct for project ambidexterity based on the concept of ambidextrous developers. We construct ambidexterity as a measure of the ability of OSS projects to pursue both exploitative and exploratory activities concurrently. To the best of our knowledge, this is the first research to study ambidexterity in OSS development. Recent research in organizational science has begun to study ambidexterity based on perceptual (survey) data in the context of formal organizations (Jansen et al. 2009, Jansen et al. 2006, Lin et al. 2007), In contrast, we used real-world project data to study ambidexterity. Our results illustrate the roles of ambidextrous developers as coordination mechanisms between patch development and feature request activities on project performance. Our results also illustrate ambidextrous developers' differences compared to other developers in terms of roles played by ambidextrous developers.

We replicated recent research on OSS development that has studied the effect of social network structure of software developers on project performance (Singh et al. 2011, Singh 2010, Singh et al. 2007, Grewal et al. 2006). However, we used larger and more recent data from the SourceForge database. Our data is also different from data used in recent research since we used a different foundry (programming language) to select projects. Our findings associated with the social network structure of software developers are consistent with the findings of recent studies on OSS development. Thus, we provide greater reliability to their findings and increase the generability of their findings.

This dissertation also makes several important contributions to practice. We empirically illustrated how ambidexterity affects project performance. We found that a moderate level of ambidexterity results in the higher performance of a project rather than very high or very low levels of ambidexterity. A moderate level of ambidexterity enables project teams to access diverse knowledge from different types of tasks, and to exchange relevant knowledge within a project team. A moderate level of ambidexterity also enables project teams to access more and deeper experience by ensuring adequate specialization to absorb and integrate new knowledge. Team composition is often a central concern for OSS project leaders. We illustrate the importance of team composition to the success of a project in terms of the optimal mix of ambidextrous and non-ambidextrous developers.

We found that non-ambidextrous developers play unique roles depending on their specializations on specific tasks (either exploitative or exploratory tasks). On the other hand, ambidextrous developers play the same or similar roles based on a variety of different tasks (both exploitative and exploratory tasks) on which they work. Therefore, we suggest that projects should be composed of both ambidextrous and non-ambidextrous developers for the following reasons. First, projects should be composed of ambidextrous who work on both exploitative and exploratory activities in order to gain diverse knowledge from different types of tasks. Second, we also suggest that projects should be composed of non-ambidextrous developers who specialize in either exploitative or exploratory activities in order to gain more and deeper experience from their specializations. Therefore, we provide OSS project leaders with a way to optimize their team compositions. OSS project leaders should identify, recruit, and retain both

ambidextrous and non-ambidextrous developers while trying to maintain a moderate level of ambidexterity.

4.9. Limitations and Future Research

We measure the effects of social network structure of OSS developers on project performance which represents the rate of knowledge creation by a project. We assume that network structure affects knowledge transfer. However, we did not observe knowledge transfer directly but rather infer it from the relationship between network structure and project performance. Knowledge may flow to projects through other mechanisms. For example, a developer may acquire knowledge from unconnected projects by using their software or by analyzing their software's source code. In this dissertation, we did not consider other mechanisms for knowledge flow. We did not analyze characteristics of individual team members such as their experiences and motivations which may also influence the extent to which knowledge is transferred or absorbed (Cohen and Levinthal 1990). These aspects of relationships can be analyzed in order to understand the mechanism through which network structure affects project performance. These limitations have been recognized in prior research on OSS social networks (Singh et al. 2011, Sing 2010).

We focus on the technical performance of a project measured as the rate of knowledge creation by a project. While we have presented several models, additional analysis and model development may be possible in future research.

We select one programming language as a network boundary. Therefore, our data is restricted to projects using the same programming language. Future research can collect data for multiple programming languages.

We measure project performance using the cumulative number of CVS and SVN commits over the life span of a project. Grewal et al. (2006) indicated the existence of multiple regimes each with possibly different models. Analyzing the effects of ambidexterity and social network structure of OSS developers on project performance in different regimes can produce interesting results.

Future research can investigate the commercial performance of a project measured as the number of downloads which represents user acceptance.

REFERENCES

Adler, P. S., and Kwon, S. W. (2002) Social Capital: Prospects for a New Concept. Academy of Management Review 27 17-40.

Ahuja, G. (2000) Collaboration Networks, Structural Holes, and Innovation: A Longitudinal Study. Administrative Science Quarterly, 45, 425-455.

Aiken, L. S., and West, S. G. (1991) Multiple Regression: Testing and Interpreting Interactions. Sage Publications, Thousand Oaks, California.

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

Allison, P. D. (1995) Survival Analysis using the SAS System: A Practical Guide. Cary, NC: SAS publishing.

Anand, S. (2008) Information Security Implications of Sarbanes-Oxley. Information Security Journal: A Global Perspective, 17, 2, 75-79.

Ancona, D. G., and Caldwell, D. F. (1992) Demography and design: Predictors of new product team productivity. Organizational Science, 3, pp. 321-341.

Antwerp, M.V.; and Madey, G. (2005) Advances in the SourceForge Research Data Archive (SRDA), The 4th International Conference on Open Source Systems (WoPDaSD 2008), Milan, Italy, available at http://www.cse.nd.edu/~oss/Papers/papers.html

Arbaugh, W. A., Fithen, W. L., and McHugh, J. (2000) Windows of vulnerability: a case study analysis. Computer, 33, 12, 52-58.

Arora, A., Krishnan, R., Telang, R., and Yang, Y. (2010a) An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. Information Systems Research, 21, 1, 115-132.

Arora, A., Forman, C., Nandkumar, A., and Telang, R. (2010b) Competition and patching of security vulnerabilities: An empirical analysis. Information Economics and Policy, 22, 164-177.

Arora, A., Telang, R., and Xu, H. (2008) Optimal policy for software vulnerability disclosure. Management Science, 54, 4, 642-656.

Arora A., Caulkins, J. P., and Telang, R. (2006a) Sell first, fix later: Impact of patching on software quality. Management Science, 52, 3, 465-471.

Arora, A., Nandkumar, A., and Telang, R. (2006b) Does information security attack frequency increase with vulnerability disclosure? An empirical analysis. Information Systems Frontiers, 8, 5, 350-362.

Atuahene-Gima, K. (2003) The Effects of Centrifugal and Centripetal Forces on Product Development Speed and Quality: How Does Problem Solving Matter? The Academy Management Journal, 46, 3, 359-373.

Avizienis, A., Laprie, J. C., Randell, B., and Landwehr, C. (2004) Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, 1, 1, 11-33.

Baker, F. B., and Hubert, L.J. (1981) The Analysis of Social Interaction Data: A Nonparametric Technique. Sociological Methods and Research, 9, 3, 339-361.

Balkundi, P., and Harrison, D. A. (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.

Banker, R. D., Bardhan, I., and Asdemir, O. (2006) Understanding the impact of collaboration software on product design and development. Information Systems Research, 17, 4, 352-373.

Banker, R. D., and Slaughter, S. A. (2000) The moderating effects of structure on volatility and complexity in software enhancement. Information Systems Research, 11, 3, 219-240.

Banker, R. D., Davis, G. B., and Slaughter, S. A. (1998) Software development practices, software complexity, and software maintenance performance: A field study. Management Science, 44, 4, 433-450.

Barrick, M., Stewart, G., Neubert, M. J., and Mount, M. (1998) Relating member ability and personality to work-team processes and team effectiveness, Journal of Applied Psychology, 83, 377-391.

Baron, D. P. (2001) Private Politics, Corporate Social Responsibility, and Integrated Strategy. Journal of Economics & Management Strategy, 10, 1, 7-45.

Beattie, S., Arnold, s., Cowan, C., Wagle, P., and Wright, C. (2002) Timing the Application of Security Patches for Optimal Uptime. Proceedings of LISA: Sixteenth Systems Admininistration Conference, USENIX Association, Berkeley, CA, 233-242.

Benner, M. J. and Tushman, M. L. (2003) Exploitation, Exploration, and Process Management: The Productivity Dilemma Revisited. Academy of Management Review, 28, 2, 238-256.

Biskup, J. (2009) Security in Computing Systems: Challenges, Approaches and Solutions. Springer-Verlag, Berlin, Germany.

Boh, W., S. Slaughter, A. Espinosa. 2007. Learning from experience in software development: A multi-level analysis. Management Science 53(8) 1315–1331.

Borgatti, S. P. (2005) Centrality and Network Flow. Social Networks, 27, 1, 55-71.

Borgatti, S.P. and Halgin, D. (2011) Analyzing Affiliation Networks. In Carrington, P. and Scott, J. (eds.) The Sage Handbook of Social Network Analysis, Sage Publications.

Borgatti, S. P., Everett, M. G., and Freeman, L. C. (2002) Ucinet for Windows: Software for Social Network Analysis. Harvard, MA: Analytic Technologies.

Brigham, E. F. and Ehrhardt, M. C. (2008) Financial Management: Theory and Practice, Thomson South-Western.

Burt, R. S. (2000) The Network Structure of Social Capital. Research in Organizational Behavior, 22, 345-423.

Burt, R.S. (1992) Structural Holes: The Social Structure of Competition. Harvard University Press, Cambridge, MA.

Burt, R.S. (2004) Structural Holes and Good Ideas. American Journal of Sociology 110, 2, 349-399.

Cardinal, L. B. (2001) Technological innovation in the pharmaceutical industry: The use of organizational control in managing research and development. Organizational Science, 12, pp. 19-36.

Carroll, A. B. A (1979) Three-Dimensional Conceptual Model of Corporate Social Performance. Academy of Management Review, 4, 497-505.

Cavusoglu, H., Cavusoglu, H., and Raghunathan, S. (2007) Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. IEEE Transactions on Software Engineering, 33, 3, 171-184.

Chandramouli, R., Grance, T., Kuhn, R., and Landau, S. (2006) Common Vulnerability Scoring System. IEEE Security & Privacy, 4, 6, 85-89.

Chengalur-Smith, S.; and Sidorova, A. (2003) Survival of Open-Source Projects: A Population Ecology Perspective. ICIS 2003 Proceedings, Paper 66.

Christensen, C. M. (1998) The Innovator's Dilemma. Harvard Business School Press, Boston.

Cohen, J., Cohen, P., West, S. G., and Aiken, L. S. (2003) Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences. 3rd edition, Lawrence Erlbaum Associates, Mahwah, New Jersey.

Cohen, J., and Cohen, P. (1983) Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Cohen, W., and Levinthal, D. (1990) Absorptive capacity: A new perspective on learning and innovation. Administrative Science Quarterly, 35, 1, 128-152.

Cohen, J. (1988) Statistical Power Analysis for the Behavioral Sciences. 2nd edition, Lawrence Erlbaum Associates, Hillsdale, NJ, England.

Cohen, J. (1977) Statistical Power Analysis for the Behavioral Sciences. 1st edition, Lawrence Erlbaum Associates, Hillsdale, NJ, England.

Coleman, J. S. (1988) Social capital in the creation of human capital. American Journal Sociology, 94, 95-120.

Computer Emergency Response Team/Coordination Center (CERT/CC) Statistics, available at http://www.cert.org/stats/

Control Objectives for Information and Related Technology (COBIT), governed by Information Systems Audit and Control Association (ISACA), www.isaca.org/COBIT

Cox, D.R. (1972) Regression models and life tables. Journal of the Royal Statistical Society, B34, 187-220.

Crowston, K., Annabi, H. and Howison, J. (2003) Defining Open Source Software Project Success. Twenty-Fourth International Conference on Information Systems.

Daft, R. L., and Lengel, R. H. (1986) Organizational Information Requirements, Media Richness and Structural Design. Management Science, 32, 5, 554-571.

DeLone, W. H., and McLean, E. R. (1992) Information Systems Success: The Quest for the Dependent Variable. Information Systems Research, 3, 1, 60-95.

Eisenhardt, K. M., and Brown, S. L. (1999) Patching. Restitching business portfolios in dynamic markets. Harvard Business Review, 77, 3, 72-82.

Egelhoff, W. G. (1991) Information-Processing Theory and the Multinational Enterprise. Journal of International Business Studies, 22, 3, 341-368.

Espinosa, J. A., Slaughter, S. A., Kraut, R. E., and Herbsleb, J. D. (2007) Team knowledge and coordination in geographically distributed software development, Journal of Management Information Systems, 24, 1, 135-169.

Fang, C., Lee, J., and Schilling, M. A., (2010) Balancing Exploration and Exploitation through Structural Design: The Isolation of Subgroups and Organizational Learning. Organization Science, 21, 3, 625–642.

Faraj, S., and Sproull, L. S. (2000) Coordinating Expertise in Software Development Teams. Management Science, 46, 12, 1554-1568.

Feller, J., and Fitzgerald, B. (2002) Understanding Open Source Software Development. London, UK: Pearson Education Limited.

Fleming, L., and Marx, M. (2006) Managing Creativity in Small Worlds. California Management Review, 48, 4, 6-27.

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

Fox, J. (1991) Regression Diagnostics. Beverly Hills, CA: Sage Publications.

Frühwirth, C., and Männistö, T. (2009) Improving CVSS-based vulnerability prioritization and response with context information. 3rd International Symposium on Empirical Software Engineering and Measurement, Washington, DC: IEEE Computer Society, 535-544.

Gacek, C., and Arief, B. (2004) The many meanings of open source. IEEE Software, 21, 1, 34-40.

Garud, R., Nayyar, P. R. (1994) Transformative Capacity: Continual Structuring by Intertemporal Technology Transfers. Strategic Management Journal, 15, 5, 365-385.

Gelman, A., and Hill, J. (2007) Data Analysis Using Regression and Multilevel/Hierarchical Models. Cambridge University Press, New York.

Gilbert, C. G. (2005) Unbundling the Structure of Inertia: Resource versus Routine Rigidity. The Academy of Management Journal, 48, 5, 741-763.

Gilbert, C. G. (2006) Change in the Presence of Residual Fit: Can Competing Frames Coexist? Organization Science, 17, 1, 150-167.

Gordon, L. A., Loeb, M. P., Lucyshyn, W., and Richardson, R. (2006) CSI/FBI Computer Crime and Security Survey. Computer Security Institute.

Gramm-Leach-Bliley Act (GLBA), available at http://www.ftc.gov/privacy/privacyinitiatives/glbact.html

Grant, R. M. (1996) Toward a Knowledge-Based Theory of the Firm. Strategic Management Journal, 17, 109-122.

Granovetter, M. (1985) Economic Action and Social Structure: The Problem of Embeddedness. American Journal of Sociology, 91, 3, 481-510.

Granovetter, M. S. (1973) The Strength of Weak Ties. American Journal of Sociology, 78, 6, 1360-1380.

Greene, W. H. (2003) Econometric Analysis, 5th edition, Prentice Hall, Upper Saddle River, NJ.

Gruenfeld, D. H., Mannix, E. A., Williams, K. Y., and Neale, M. A. (1996) Group Composition and Decision Making: How Member Familiarity and Information Distribution Affect Process and Performance. Organizational Behavior and Human Decision Processes, 67, 1, 1-15.

Gupta, R., and Govindarajan, R. (2000) Knowledge flows within multinational corporations. Strategic Management Journal, 21, 4, 473-496.

Gupta, A. K., Smith, K. G., and Shalley, C. E. (2006) The Interplay Between Exploration And Exploitation. Academy of Management Journal, 49, 4, 693-706.

Gnyawali, D. R., and Madhavan, R. (2001) Cooperative Networks and Competitive Dynamics: A Structural Embeddedness Perspective. Academy of Management Review, 26, 3, 431-445.

Green, S. A. (1991). How Many Subjects Does It Take To Do A Multiple Regression Analysis. Multivariate Behavioral Research, 26, 3, 499-510.

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

Gulati, R. (1999) Network location and learning: The influence of network resources and firm capabilities on alliance formation. Strategic Management Journal, 20, 397-420.

Gulati, R., and Garguilo, M. (1999) Where do networks come from?. American Journal of Sociology, 104, 1439-1493.

Ha, R. R., and J. C. Ha (2012) Integrative Statistics for the Social and Behavioral Sciences. SAGE Publications, Washington, USA.

Hahn, J., Moon, J. Y., and Zhang, C. (2008) Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties. Information Systems Research, 19, 3, 369-391.

Hambrick, D. C. (1983) Some Tests of the Effectiveness and Functional Attributes of Miles and Snow's Strategic Types. Academy of Management Journal, 26, 1, 5-26.

Hanneman, R. A., and Riddle, M. (2005) Introduction to social network methods. Riverside, CA: University of California, Riverside (Available in digital form at http://faculty.ucr.edu/~hanneman/)

Hansen, M. T. (2002) Knowledge Networks: Explaining Effective Knowledge Sharing in Multiunit Companies. Organization Science, 13, 3, 232-248.

Hansen, M. T. (1999) The Search-Transfer Problem: The Role Of Weak Ties in Sharing Knowledge Across Organizational Subunits. Administrative Science Quarterly, 44, 82-111.

He, J., Butler, B. S., and King, W. R. (2007) Team Cognition: Development and Evolution in Software Project Teams. Journal of Management Information Systems, 24, 2, 261-292.

He, Z., and Wong, P. K. (2004) Exploration vs. exploitation: An empirical test of the ambidexterity hypothesis. Organizational Science, 15, 4, 481-494.

Henderson, R., and Cockburn, I. (1994) Measuring Competence? Exploring Firm Effects in Pharmaceutical Research. Strategic Management Journal, 15, 63-84.

Hosmer, D. W., and Lemeshow, S. (1999) Applied Survival Analysis: Regression Modeling of Time to Event Data. New York: John Wiley.

Houmb, S. H., Franqueira, V. N. L., and Engum, E. A. (2008) Estimating Impact and Frequency of Risks to Safety and Mission Critical Systems using CVSS. IEEE CS Conference Proceedings, Seattle: IEEE Computer Society Press.

Hubert, L. (1987) Assignment Methods in Combinatorial Data Analysis. Dekker, New York.

Hubert, L., and Schultz, J. (1976) Quadratic Assignment as a General Data Analysis Strategy. British Journal of Mathematical and Statistical Psychology, 29, 190-241.

IEEE Standard for Software Maintenance. IEEE, New York: The Institute of Electrical and Electronics Engineers, 1993.

Jaisingh, J., See-To, E. W. K., and Tam, K. Y. (2009) The impact of open source software on the strategic choices of firms developing proprietary software. Journal of Management Information Systems, 25, 3, 241-275.

Jaffe, A.B. (1986) Technological Opportunity and Spillovers of R&D: Evidence from Firms' Patents, Profits and Market Value. American Economic Review, 76, 5, 984-999.

Jansen, J. J. P., Tempelaar, M., Van den Bosch, F. A. J., and Volberda, H. (2009) Structural differentiation and ambidexterity: The mediating role of integration mechanisms. Organizational Science, 20, 4, 797-811.

Jansen, J. J. P., Van Den Bosch, F. A. J., and Volberda, H. W. (2006) Exploratory Innovation, Exploitative Innovation, and Performance: Effects of Organizational Antecedents and Environmental Moderators. Management Science, 52, 11, 1661-1674.

Johnson, M. E. (2008) Information risk of inadvertent disclosure: An analysis of file-sharing risk in the financial supply chain. Journal of Management Information Systems, 25, 2, 97-123.

Kannan, K., Rees, J., and Sridhar, S. (2007) Market reactions to information security breach announcements: an empirical analysis. International Journal of Electronic Commerce, 12, 1, 69-91.

Kemerer, C. F. and Slaughter, S. (1999) An Empirical Approach to Studying Software Evolution. IEEE Transactions on Software Engineering, 25, 4, 493-509.

Kemerer, C. F. (1995) Software complexity and software maintenance: A survey of empirical research. Annals of Software Engineering, 1, 1, 1-22.

Kim, Y., Stohr, E.A. (1998) Software Reuse: Survey and Research Directions. Journal of Management Information Systems, 14, 4, 113-148.

Khoshgoftaar, T. M., Allen, E. B., Jones, W. D., and Hudepohl, J. P. (2000) Classification-Tree Models of Software-Quality over Multiple Releases. IEEE Transactions on Reliability, 49, 1, 4-11.

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

Koza, M. P., and Lewin, A. Y. (1998) The co-evolution of strategic alliances. Organizational Science, 9, 3, 255-264.

Krackhardt, D. (1998) Simmelian Ties: Super Strong and Sticky. In Roderick Kramer and Margaret Neale (eds.) Power and Influence in Organizations. Thousand Oaks, CA: Sage, 21-38.

Krackhardt, D. (1996) Social networks and the liability of newness for managers. Journal of Organizational Behavior, 3, 159-173.

Krackhardt, D. (1992) The Strength of Strong Ties: The Importance of Philos in Organizations. In N. Nohria & R. Eccles (eds.), Networks and Organizations: Structure, Form and Action, Boston, MA: Harvard Business School Press, 216-239.

Krackhardt, D. (1988) Predicting with Networks: Nonparameteric Multiple Regression Analysis of Dyadic Data. Social Networks, 10, 4, 359-381.

Krackhardt, D. (1987) QAP Partialling as a Test of Spuriousness. Social Networks, 9, 171-186.

Krishnan, M. S., Mukhopadhyay, T., and Kriebel, C. H. A (2004) Decision Model for Software Maintenance. Information Systems Research, 15, 4, 396-412.

Lazer, D., and Friedman, A. (2007) The Network Structure of Exploration and Exploitation. Administrative Science Quarterly, 52, 667-694.

Lawrence, P. R., and Lorsch, J. W. (1967) Differentiation and Integration in Complex Organizations. Administrative Science Quarterly, 12, 1, 1-47.

Lavie, D., and Rosenkopf, L. (2006) Balancing exploration and exploitation in alliance formation. Academy of Management Journal, 49, 6, 797-818.

Lerner, J., and Tirole, J. (2005) The Scope of Open Source Licensing. Journal of Law, Economics, & Organization, 21, 1, 20-56.

Levinthal, D. A., and March, J. G. (1993) The myopia of Learning. Strategic Management Journal, 14, 95-112.

Levin, D. Z., and Cross, R. (2004) The Strength of Weak Ties You Can Trust: The Mediating Role of Trust in Effective Knowledge Transfer. Management Science, 50, 11, 1477-1490.

Lewin, A. Y., Long, C. P., and Caroll, T. N. (1999) The Coevolution of New Organizational Forms. Organization Science, 10, 5, 535-550.

Li, S., Shang, J., and Slaughter, S. A. (2010) Why Do Software Firms Fail? Capabilities, Competitive Actions, and Firm Survival in the Software Industry from 1995 to 2007. Information Systems Research, 21, 3, pp. 631–654.

Lin, N. (2005) A Network Theory of Social Capital. Handbook on Social Capital, Oxford University Press.

Lin, N. (1999) Building a Network Theory of Social Capital. Connections, 22, 1, 28-51.

Lin, Z., Yang, H., and Demirkan, I. (2007) The Performance Consequences of Ambidexterity in Strategic Alliance Formations: Empirical Investigation and Computational Theorizing. Management Science, 53, 10, 1645-1658.

Lin, D. Y. and Wei, L. J. (1989) The robust inference for the Cox proportional hazards model. Journal of the American Statistical Association, 84, 408, 1074-1078.

Liu, X., and Iyer, B. (2007) Design Architecture, Developer Networks, and Performance of Open Source Software Projects. International Conference on Information Systems, Montreal: Association for Information Systems.

Malone, T. W., and Crowston, K. (1994) The interdisciplinary study of coordination. ACM Computing Surveys, 26, 1, 87-119.

March, J. G. (1991) Exploration and Exploitation in Organizational Learning. Organization Science, 2, 1, 71-87.

Marsden, P. V. (2005) Recent Developments in Network Measurement, in Models and Methods in Social Network Analysis, P. Carrington and J. Scott and S. Wasserman, Eds. New York: Cambridge University Press.

Mazen, A., Magid, M., Hemmasi, M., and Lewis, M. F. (1985) In Search of Power: A Statistical Power Analysis of Contemporary Research in Strategic Management. Academy of Management Proceedings, 30-34.

McCain, B. E., O'Reilly, C., and Pfeffer, J. (1983) The Effects of Departmental Demography on Turnover: The Case of a University. The Academy of Management Journal, 26, 4, 626-641.

Mell, P., Scarfone, K., and Romanosky, S. A. (2007) Complete Guide to the Common Vulnerability Scoring System Version 2.0. Forum of Incident Response and Security Teams, available at http://www.first.org/cvss/cvss-guide.pdf

Mell, P., and Scarfone, K. (2007) Improving the Common Vulnerability Scoring System. IET Information Security, 1, 3, 119-127.

Mockus A., Fielding, R., and Herbsleb, J. (2002) Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11, 3, 309-346.

Myers, R.H. 1990. Classical and Modern Regression Application. 2nd edition, Duxbury Press, Belmont, California.

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

Narayanan, S., Balasubramanian, S., and Swaminathan, J. M. (2009) A Matter of Balance: Specialization, Task Variety, and Individual Learning in a Software Maintenance Environment. Management Science, 55, 11, 1861-1876.

National Vulnerability Database (NVD), available at http://nvd.nist.gov/

Nelson, R., and Winter, S. (1982) An Evolutionary Theory of Economic Change. Harvard University Press, Cambridge: M A.

Nelson, M. L., Sen, R., and Subramaniam, C. (2006) Understanding open source software: A research classification framework. Communications of AIS, 17, 12, 266-287.

Newman, M. E. J., Strogatz, S. H., and Watts, D. J. (2001) Random graphs with arbitrary degree distributions and their applications. Physical Review E, 64, 1-17.

Nord, W. R., and Tucker, S. (1987) Implementing Routine and Radical Innovations. Lexington Books, Lexington, MA.

O'Brien, J. A. and Marakas, G. M. (2008) Management Information Systems. New York: McGraw-Hill/Irwin.

O'Reilly, C. A., and Tushman, M. L. (2004) The Ambidextrous Organization. Harvard Business Review, 82, 4, 74-81.

O'Reilly, C. A., Caldwell, D. F., and Barnett, W. P. (1989) Work Group Demography, Social Integration, and Turnover. Administrative Science Quarterly, 34, 1, 21-37.

Open Source Vulnerability Database (OSVDB), available at http://www.osvdb.org/

Park, S. H., Chen, R., and Gallagher, S. (2002) Firm resources as moderators of the relationship between market growth and strategic alliances in semiconductor start-ups. Academy of Management Journal, 45, 527-545.

Pedhazur, E.J. (1997). Multiple Regression in Behavioral Research. 3rd edition. New York: Harcourt Brace College Publishers.

Peduzzi, P., Concato, J., Feinstein, A. R., and Holford, T. R. (1995) Importance of events per independent variable in proportional hazards regression analysis II. Accuracy and precision of regression estimates. Journal of Clinical Epidemiology, 48, 12, 1503-1510.

Pelled, L. H., Eisenhardt, K. M., and Xin, K. R. (1999) Exploring the black box: An analysis of work group diversity, conflict, and productivity. Administrative Science Quarterly, 44, pp 1-28.

Pfeffer, J. (1983) Organizational demography: Implications for management. California Management Review, 28, pp. 67-81.

Pfeffer, J., and Salancik, G. R. (1978) The External Control of Organizations: A Resource Dependence Perspective, Harper, New York.

Pfleeger, C. P. and Pfleeger, S. L. (2003) Security in Computing. Prentice Hall, NJ, USA.

Postrel, S. (2002) Islands of Shared Knowledge: Specialization and Mutual Understanding in Problem-Solving Teams. Organization Science, 13, 3 303-320.

Powell, W. W., and Smith-Doerr, L. (1994) Networks and Economic Life. The Handbook of Economic Sociology, Princeton, NJ: Princeton University Press, 368-402.

Png, I. P. L., Wang, C. Y., and Wang, Q. H. (2008) The deterrent and displacement effects of information security enforcement: International evidence. Journal of Management Information Systems, 25, 2, 125-144.

Poel, D. V. D., and Lariviere, B. (2004) Customer attrition analysis for financial services using proportional hazard models. European Journal of Operational Research, 157, 1, 196-217.

Portes, A., and Sensenbrenner, J. (1993) Embeddedness and immigration: Notes on the social determinants of economic action. American Journal of Sociology, 98, pp. 1320-1350.

Powell, W. W., Koput, K. W., and Smith-Doerr, L. (1996) Interorganizational Collaboration and the Locus of Innovation: Networks of Learning in Biotechnology. Administrative Science Quarterly, 41, 116-145.

Rai, A., Lang, S. S., and Welker, R. B. (2002) Assessing the validity of IS success models: An empirical test and theoretical analysis. Information Systems Research, 13, 1, 50-69.

Raisch, S., Birkinshaw, J., Probst, G., and Tushman, M. L. (2009) Organizational Ambidexterity: Balancing Exploitation and Exploration for Sustained Performance. Organization Science, 20, 4, 685–695.

Ransbotham, S. and Mitra, S. (2009) Choice and Chance: A Conceptual Model of Paths to Information Security Compromise, Information Systems Research, 20, 1, 121-139.

Raymond, E. S. (1999) The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. (1st Edition). Cambridge: O'Reilly Media Inc.

Reagans, R., and Zuckerman, E. W. (2001) Networks, Diversity, and Productivity: The Social Capital of Corporate R&D Teams. Organization Science, 12, 4, 502-517.

Roberts, T. L., Cheney, P. H., Sweeney, P. D., and Hightower, R. T. (2005) The effects of information technology project complexity on group interaction, Journal of Management Information Systems, 21, 3, 223-247.

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

Rosenkopf, L., and Almeida, P. (2003) Overcoming Local Search through Alliances and Mobility. Management Science, 49, 6, 751-766.

Rosenkopf, L., and Nerkar, A. (2001) Beyond local search: Boundary spanning, exploration, and impact in the optical disk industry. Strategic Management Journal, 22, 4, 287-306.

Rothaermel, F. T., and Deeds, D. (2004) Exploration and exploitation alliances in biotechnology: A system of new product development. Strategic Management Journal, 25, 201-221.

Rowley, T., Behrens, D., and Krackhardt D. (2000) Redundant Governance Structures: An Analysis of Structural and Relational Embeddedness in the Steel and Semiconductor Industries. Strategic Management Journal, 21, 369-386.

Schilling, M. A., and Phelps, C .C. (2007) Interfirm Collaboration Networks: The impact of Large Scale Network Structure on Firm Innovation. Management Science, 53, 7, 1113-1126.

Sen, R. (2007) A strategic analysis of competition between open source and proprietary software. Journal of Management Information Systems, 24, 1, 233-257.

Shaft, T. M., and Vessey, I. (2006) The Role of Cognitive Fit in The Relationship Between Software Comprehension and Modification. MIS Quarterly, 30, 1, 29-55.

Shane, S., and Cable, D. (2002) Network Ties, Reputation, and the Financing of New Ventures. Management Science, 48, 3, pp. 364-381.

Siggelkow, N., and Levinthal, D. A. (2003) Temporarily Divide to Conquer: Centralized, Decentralized, and Reintegrated Organizational Approaches to Exploration and Adaptation. Organization Science, 14, 6, 650-669.

Simon, H. (1991) Bounded Rationality and Organizational Learning. Organization Science, 2, 1, 125-134.

Singh, P. V., Tan, Y., and Mookerjee, V. (2011) Network Effects: The Influence of Structural Social Capital on Open Source Project Success. Management Information Systems Quarterly, 35, 4, 813-829.

Singh, P. V. (2010) The Small World Effect: The Influence of Macro Level Properties of Developer Collaboration Networks on Open Source Project Success. ACM Transactions of Software Engineering and Methodology, 20, 2, 1-37.

Singh, P.V., Tan, Y., and Mookerjee, V. (2007) Social Capital, Structural Holes, and Team Composition: Collaborative Networks of the Open Source Software Community. Proceedings of the 28th ICIS.

Sirmon, D. G., Hitt, M. A., and Ireland, R. D. (2007) Managing Firm Resources in Dynamic Environments to Create Value: Looking Inside the Black Box. Academy of Management Review, 32, 1, 273-292.

Slaughter, S. A., Levine, L., Ramesh, B., Pries-Heje, J., and Baskerville, R (2006) Aligning Software Processes with Strategy. MIS Quarterly, 30, 4, 891-918.

Slaughter, S. A., Harter, D. E., and Krishnan, M. S. (1998) Evaluating the Cost of Software Quality. Communications of the ACM, 41, 8, 67-73.

SourceForge.net. Available at http://SourceForge.net.

Smith, W. K., and Tushman, M. L. (2005) Managing Strategic Contradictions: A Top Management Model for Managing Innovation Streams. Organization Science, 16, 5, 522-536.

Snow, C. C., and Hrebiniak, L. G., (1980) Strategy, Distinctive Competence, and Organizational Performance. Administrative Science Quarterly, 25, 2, 317-336.

Subramaniam, C.; Sen, R.; and Nelson, M. L. (2009) Determinants of open source software project success: A longitudinal study. Decision Support Systems, 46, 576–585.

Stephenson, K., and Zelen, M. (1989) Rethinking Centrality: Methods and Applications. Social Networks, 13.

Stevens, J. P. (2002). Applied Multivariate Statistics for the Social Sciences, 4th edition, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Subramaniam, M., and Youndt, M. A. (2005) The influence of intellectual capital on the types of innovative capabilities. Academy of Management Journal, 48, 450-463.

Szulanski, G. (1996) Exploring Internal Stickiness: Impediments to the Transfer of Best Practice within the Firm. Strategic Management Journal, 17, 27-43.

Tabachnick, B. G., and Fidell, L. S. (2007) Using Multivariate Statistics. 5th Edition, Pearson Education, Boston.

Tan, P. N., Steinbach, M., and Kumar, V. Introduction to Data Mining. Pearson Addison Wesley, 2006.

Tushman, M. L., and O'Reilly, C. A. (2006) Ambidextrous Organizations: Managing Evolutionary and Revolutionary Change. California Management Review, 38, 4, 8-30.

Katz, R. and Tushman, R. (1979) Communication Patterns, Project Performance, and Task Characteristics: An Empirical Evaluation and Integration in an R&D Setting, Organizational Behavior and Human Performance, 23, 139-162.

United States Computer Emergency Readiness Team (US-CERT), available at http://www.us-cert.gov/

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

Uzzi, B. (1999) Embeddedness in the making of financial capital: How social relations and networks benefit firms seeking financing. American Sociological Review, 64, 481-505.

Uzzi, B. (1997) Social Structure and Competition in Interfirm Networks: The Paradox of Embeddedness. Administrative Science Quarterly, 42, 35-67.

Uzzi, B. (1996) The sources and consequences of embeddedness for the economic performance of organizations: The network effect. American Sociological Review, 61, 674-698.

Van Antwerp, M., and Madey, G. (2008) Advances in the SourceForge Research Data Archive (SRDA). Proceedings of the 4th International Conference on Open Source Systems, Milan, Italy, September 7-10. available at http://www.nd.edu/~oss/Papers/srda_final.pdf

Van den Bosch, F. A. J., Volberda, H. W., and Boer, M. D. (1999) Coevolution of Firm Absorptive Capacity and Knowledge Environment: Organizational Forms and Combinative Capabilities. Organizational Science, 10, 5, 551-568.

Van Vliet, H. (2000). Software Engineering: Principles and Practices, 2nd Edition. JohnWiley & Sons, West Sussex, England.

Vlas, R. E., and Robinson, W. N. (2012) Two Rule-Based Natural Language Strategies for Requirements Discovery and Classification in Open Source Software Development Projects. Journal of Management Information Systems, 28, 4, 11-38.

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

Wasserman, S., and Frost, K. (1994) Social Network Analysis: Methods and Applications. Cambridge Univeersity Press, Cambridge, UK.

Walker, G., Kogut, B., and Shan, W. (1997) Social capital, structural holes and the formation of an industry network. Organization Science, 8, 109-125.

Watts, D.J. (1999) Networks, Dynamics, and the Small World Phenomenon. American Journal of Sociology, 105, 2, 493-527.

Watts, D.J., and Strogatz, S.H. (1998) Collective Dynamics of Small World Networks. Nature, 393, 440-442.

Weitzner, D. J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., and Sussman, G. J. (2008) Information Accountability. Communications of the ACM, 51, 6, 82-87.

Wejnert, B. (2002) Integrating models of diffusion of innovation: A conceptual framework. Annual Review of Sociology, 28 pp. 297-326.

Wood, D. J. (1991) Corporate Social Performance Revisited. Academy of Management Review, 16, 4, 691-718.

Xia, W., and Lee, G. (2005) Complexity of information systems development projects: Conceptualization and measurement development. Journal of Management Information Systems, 22, 1, 45-83.

Xu, J.; Gao, Y.; Christley, S.; and Madey, G. (2005) A Topological Analysis of the Open Source Software Development Community, The 38th Hawaii International Conference on Systems Science (HICSS-38), Hawaii, available at http://www.cse.nd.edu/~oss/Papers/papers.html

Zaheer, A., and Bell, G. G. (2005) Benefiting From Network Position: Firm Capabilities, Structural Holes, and Performance. Strategic Management Journal, 26, 809-825.

Zenger, T. R., and Lawrence, B. S. (1989) Organizational Demography: The Differential Effects of Age and Tenure Distributions on Technical Communication. The Academy of Management Journal, 32, 2, 353-376.

APPENDIX A: HAZARD RATIO CALCULATION FOR VARIABLES

Hazard ratios of model variables on patch release time have been calculated based on Cox's proportional hazard model expressed as follows:

$$h(t, X(t)) = h_0(t) \exp[\sum_{i=1}^{k} \beta_i X_i + \sum_{j=1}^{n} \beta_j X_j(t)]$$

where:
$h_0(t)$ = Baseline hazard function at time t
β = Model coefficient of independent variables or interactions
X = Independent variables

TABLE A1: Hazard Ratio Calculation for Disclosure

	Confidentiality Model	Integrity Model
Disclosure	exp[Disclosure*$\beta_{Disclosure}$]	exp[Disclosure*$\beta_{Disclosure}$]
0 [Not disclosed]	exp[0*(1.07655)] = 1	exp[0*(1.10792)] = 1
1 [Disclosed]	exp[1*(1.07655)] = 2.93	exp[1*(1.10792)] = 3.03

TABLE A2: Hazard Ratio Calculation for Multiple Vendors

	Confidentiality Model	Integrity Model
MVendor	exp[MVendor*$\beta_{MVendor}$]	exp[MVendor*$\beta_{MVendor}$]
0 [Single]	exp[0*(0.49679)] = 1	exp[0*(0.45523)] = 1
1 [Multiple]	exp[1*(0.49679)] = 1.64	exp[1*(0.45523)] = 1.58

TABLE A3: Hazard Ratio Calculation for Confidentiality

	Confidentiality Model
Confidentiality	exp[C*$\beta_{Confidentiality}$]
0 [None]	exp[0*(0.26318)] = 1
1 [Partial]	exp[1*(0.26318)] = 1.30
2 [Complete]	exp[2*(0.26318)] = 1.69

TABLE A4: Hazard Ratio Calculation for Integrity

	Integrity Model
Integrity	$\exp[I*\beta_{Integrity}]$
0 [None]	exp[0*(0.24536)] = 1
1 [Partial]	exp[1*(0.24536)] = 1.28
2 [Complete]	exp[2*(0.24536)] = 1.63

TABLE A5: Hazard Ratio Calculation for Availability

		Confidentiality Model	Integrity Model
Availability	VType *	$\exp[A*\beta_{Availability} + A*VType*\beta_{VType\_A}]$	$\exp[A*\beta_{Availability} + A*VType*\beta_{VType\_A}]$
0 [None]	0 [PS]	exp[0*(-0.44305) + 0*0*(0.53838)] = 1	exp[0*(-0.37897) + 0*0*(0.53785)] = 1
0 [None]	1 [OSS]	exp[0*(-0.44305) + 0*1*(0.53838)] = 1	exp[0*(-0.37897) + 0*1*(0.53785)] = 1
1 [Partial]	0 [PS]	exp[1*(-0.44305) + 1*0*(0.53838)] = 0.64	exp[1*(-0.37897) + 1*0*(0.53785)] = 0.68
1 [Partial]	1 [OSS]	exp[1*(-0.44305) + 1*1*(0.53838)] = 1.10	exp[1*(-0.37897) + 1*1*(0.53785)] = 1.17
2 [Complete]	0 [PS]	exp[2*(-0.44305) + 2*0*(0.53838)] = 0.41	exp[2*(-0.37897) + 2*0*(0.53785)] = 0.47
2 [Complete]	1 [OSS]	exp[2*(-0.44305) + 2*1*(0.53838)] = 1.21	exp[2*(-0.37897) + 2*1*(0.53785)] = 1.37
* Availability has an interaction with Vendor Type.			

TABLE A6: Hazard Ratio Calculation for Patch Type

	Confidentiality Model	Integrity Model
PType	$\exp[PType*\beta_{PType}]$	$\exp[PType*\beta_{PType}]$
0 [Update]	exp[0*(-0.39724)] = 1	exp[0*(-0.39043)] = 1
1 [New Release]	exp[1*(-0.39724)] = 0.67	exp[1*(-0.39043)] = 0.68

TABLE A7: Hazard Ratio Calculation for Software Type

	Confidentiality Model	Integrity Model
SWType	$\exp[SWType*\beta_{SWType}]$	$\exp[SWType*\beta_{SWType}]$
0 [Application SW]	exp[0*(0.39553)] = 1	exp[0*(0.38421)] = 1
1 [System SW]	exp[1*(0.39553)] = 1.49	exp[1*(0.38421)] = 1.47

TABLE A8: Hazard Ratio Calculation for Patch Quality (Multiple Patches)

	Confidentiality Model	Integrity Model
Multiple Patches	exp[MPatches*$\beta_{MPatches}$]	exp[MPatches*$\beta_{MPatches}$]
0 [Single]	exp[0*(0.31560)] = 1	exp[0*(0.31731)] = 1
1 [Multiple]	exp[1*(0.31560)] = 1.37	exp[1*(0.31731)] = 1.37

TABLE A9: Hazard Ratio Calculation for Vendor Type

		Confidentiality Model	Integrity Model
VType *	Availability **	exp[VType*β_{VType} + VType*β_{VTypeT} + VType*A*β_{VType_A}]	exp[VType*β_{VType} + VType*β_{VTypeT} + VType*A*β_{VType_A}]
0 [PS]	0 [None]	exp[0*(1.62207) + 0*(-0.38656) + 0*0*(0.53838)] = 1	exp[0*(1.67718) + 0*(-0.39976) + 0*0*(0.53785)] = 1
0 [PS]	1 [Partial]	exp[0*(1.62207) + 0*(-0.38656) + 0*1*(0.53838)] = 1	exp[0*(1.67718) + 0*(-0.39976) + 0*1*(0.53785)] = 1
0 [PS]	2 [Complete]	exp[0*(1.62207) + 0*(-0.38656) + 0*2*(0.53838)] = 1	exp[0*(1.67718) + 0*(-0.39976) + 0*2*(0.53785)] = 1
1 [OSS]	0 [None]	exp[1*(1.62207) + 1*(-0.38656) + 1*0*(0.53838)] = 3.44	exp[1*(1.67718) + 1*(-0.39976) + 1*0*(0.53785)] = 3.59
1 [OSS]	1 [Partial]	exp[1*(1.62207) + 1*(-0.38656) + 1*1*(0.53838)] = 5.89	exp[1*(1.67718) + 1*(-0.39976) + 1*1*(0.53785)] = 6.14
1 [OSS]	2 [Complete]	exp[1*(1.62207) + 1*(-0.38656) + 1*2*(0.53838)] = 10.10	exp[1*(1.67718) + 1*(-0.39976) + 1*2*(0.53785)] = 10.52
* Vendor Type has been constructed as a time-dependent covariate due to its interaction with time. ** Vendor Type has an interaction with Availability.			

APPENDIX B: VARIABLE CALCULATIONS

Calculation of Clustering Coefficient:

Following Watts and Strogatz (1998), we measured the clustering coefficient as follows:

$$Clustering\ Coefficient = \frac{3 \times (number\ of\ triangles\ in\ the\ network)}{(number\ of\ connected\ triples\ of\ vertices)}$$

The "triangles" are trios of vertices where each one is connected to the other two. The "connected triples" are trios where at least one is connected to the other two (Watts and Strogatz 1998). A triplet consists of three nodes that are connected by either two or three undirected ties. A triangle consists of the three different configurations of closed trios of three vertices. In order to account for the three different configurations, a factor of three is added in the numerator (Watts and Strogatz 1998). The factor of three ensures that the clustering coefficient lies strictly in the range from 0 to 1.

Calculation of External Cohesion:

We measured the external cohesion with Burt's (1992) network constraint. Network constraint measures the extent to which a project member *i*'s external network is invested in his relationship with an external alter *j*. The network constraint posed by external alter[12] *j* on ego *i* is measured as:

$$Network\ Constraint_p = \frac{\sum_{i=1}^{N_p} \sum_{q=1}^{N_e} (p_{ij} + \sum_{q=1}^{N_e} p_{iq} p_{qj})^2}{N_p}, q \neq i, j$$

where N_p is the number of project members and N_e is the number of developers external to the project. There are two components to this constraint measure. The first component is the proportion of her total network time and energy that a project member *i* directly allocates to external alter *j*:

$$p_{ij} = \frac{(z_{ij} + z_{ji})}{\sum_{q=1}^{N} (z_{iq} + z_{qi})}$$

where z_{ij} is the tie strength between *i* and *j*. The second component is the strength of the indirect connections between *i* and *j* through mutual contacts *q*:

[12] In social network analysis, the focal actor is termed as ego and the actors who have ties to the ego are termed as alters.

$$\sum_{q=1}^{N_e} p_{iq} p_{qi}$$

Here p_{iq} is the proportion of her total network time and energy that i devotes to q and p_{qj} is the proportion of her total network time and energy that contact q devotes to contact j. Note that contact q belongs to a group a developers that are external to the focal project. This formulation allows us to measure the extent to which a project member's external contacts share relationships with each other.

Calculation of Technological Diversity:

In order to calculate Technological diversity, we first defined the technological position of each project. The technological position of a project can be defined in terms of different dimensions such as the type of the project, programming language, user interface, and operating system (Singh et al. 2011). Each of these dimensions represents different type of technical expertise. A project type represents the application domain knowledge whereas other three dimensions represent the tool knowledge and expertise that comprise the knowledge of process, data and functional architecture (Kim and Stohr 1998, Singh et al. 2011). The similarity of domain and tools affect the amount of knowledge that can be reused from one project to another (Singh et al. 2011).

Following Jaffe (1986), we characterized a project's technological position by a vector $F_p = (F_1 \ldots F_k)$, where k is the total number of categories under the four dimensions, and F_k is an indicator variable that equals to 1 if the project p falls under the category k. A project can fall under several categories within a single dimension. For example, a project can fall under education, internet, communication, and office/business categories in the project type dimension. A project can fall under developers, industrial users, system administrators, and end users in the user interface (target users) dimension. Technological diversity between the two projects p and q is then calculated by the angular separation or uncentered correlation of the vectors Fp and Fq as follows (Jaffe 1986):

$$Technological\ Diversity_{pq} = \frac{1 - F_p F_q'}{\sqrt{\left(F_p F_p'\right)\left(F_q F_q'\right)}}$$

Calculation of Indirect Ties with Frequency Decay Function:

Burt (1992) provided a frequency decay measure for indirect ties that accounts for this decline in tie strength across distant ties. The argument for the frequency decay function is that the rate at which the strength of a relation decreases with the increasing length of its corresponding path distance should vary with the social structure in which it

occurs (Burt 1992). Following Burt (1992), this decay function for the developer *i* is given as:

$$d_{ij} = 1 - \frac{f_{ij}}{(N_i + 1)}$$

where *fij* is the number of developers that the developer *i* can reach within and including path length *j*, and *Ni* is the total number of developers that the developer *i* can reach in the network. Then *dij* is the decay associated with the information that is received from developers at path length *j*. The measure of indirect ties with a frequency decay function for the developer *i* is then calculated as:

$$Indirect\ Ties\ with\ FD_i = \sum_{u=2}^{N} d_{ij} w_{ij}$$

where *N* is the total number of developers in the network and *wij* is the number of developers that lie at a path length of *j* from *i*.

Calculation of Degree Centrality:

We measured the degree centrality with Freeman's (1979) degree centrality. Degree centrality is the measure of how many an actor is connected to other actors in the network, i.e. the number of direct connections of an actor (Freeman 1979, Wasserman and Frost 1994). Degree centrality of a developer reflects the activeness of a developer in the network. Following Wasserman and Frost (1994), the degree centrality of an actor *i* is defined as:

$$Degree\ Centrality_i = \frac{k_i}{N-1} = \frac{\sum_{j=1}^{M} X_{ij}}{N-1}$$

where k_i is the degree of an actor *i* calculated as the sum of X_{ij} which gets the value of 1 if an actor *i* is connected to *j*, otherwise gets the value of 0. *N* is the total number of actors in the network. The degree centrality is normalized by dividing by the maximum possible degree in the network (*N-1*) which is that one actor is connected to all other actors in the network. This calculation results in that the degree centrality lies in the range from 0 to 1. However, UCINET reports the normalized degree centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of degree centrality for a project ranges from 0 to 100.

Calculation of Betweenness Centrality:

We measured the betweenness centrality with Freeman's (1979) betweenness centrality. Betweenness centrality is the measure of how often a developer falls on the shortest path between pairs of other developers (Freeman 1979, Wasserman and Faust

1994). Developers with a high betweenness centrality lie in the shortest path of information flow between other developers. These developers can exert control over information flow among other developers, and potentially may have some control over the interactions between other developers (Wasserman and Faust 1994). Thus, betweenness centrality signifies a developer's ability to be central to the flow of information and resources in the network. These developers can be important to the network-wide information diffusion process by occupying a central position on the shortest path between other developers in a network. Following Wasserman and Frost (1994), the degree centrality of an actor *i* is defined as:

$$Betweenness\ Centrality_i = \frac{\sum_{j<k} \frac{a^i_{jk}}{n_{jk}}}{[(N-1)(N-2)/2]}$$

where n_{jk} is the number of shortest paths between actors *j* and *k*, a^i_{jk} is the number of shortest paths between actors *j* and *k* passing through an actor *i*. *N* is the total number of actors in the network. The betweenness centrality is normalized by dividing by the maximum possible betweenness in the network [(*N-1*)(*N-2*)/2] which is the number of pairs of actors not including an actor *i* (the maximum possible paths passing through an actor). This calculation results in that the betweenness centrality lies in the range from 0 to 1. However, UCINET reports the normalized betweenness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of betweenness centrality for a project ranges from 0 to 100.

Calculation of Closeness Centrality:

We measured the closeness centrality with Freeman's (1979) closeness centrality. Closeness centrality is the measure of how close an actor is to all other actors in the network by considering direct and indirect connections to all other actors (Freeman 1979, Wasserman and Frost 1994). It basically measures the inverse of the sum of geodesic distances between actors in the network, thereby an actor with high closeness centrality has minimum geodesic distances to other actors. Closeness centrality signifies a developer's ability to reach resources in the network (Gulati and Gargiulo1999). Information would have to travel over shorter distances to reach a developer who is more central in the network (Wasserman and Faust 1994). A developer who is close to many developers can quickly interact and communicate with them without passing through many intermediaries (Wasserman and Faust 1994). Following Wasserman and Frost (1994), the closeness centrality of an actor *i* is defined as:

$$Closeness\ Centrality_i = \frac{N-1}{\sum_{j=1}^{N} d(n_i, n_j)}$$

where $d(n_i, n_j)$ is the shortest path distance between actors *j* and *k*. *N* is the total number of actors in the network. The closeness centrality is normalized by multiplying by

the maximum possible path distance in the network (*N-1*) which is that one actor is connected to another one actor passing through all other actors in the network, i.e., there are (*N-1*) path distances between those two actors. This calculation results in that the closeness centrality lies in the range from 0 to 1. However, UCINET reports the normalized closeness centrality as a percentage for each node by multiplying with 100 (Wasserman and Frost 1994). Therefore, the measure of closeness centrality for a project ranges from 0 to 100.

APPENDIX C: CORRELATION BETWEEN PAIRED VARIABLES

TABLE C1: Correlations between Paired Variables (N=690)

Variable Type	Paired Variable Names	Correlation	Sig.
Internal Connectivity	Clustering Coefficient (Patch) Clustering Coefficient (FR)	0.804	.000***
	Repeat Ties (Patch) Repeat Ties (FR)	0.837	.000***
	Third Party Ties (Patch) Third Party Ties (FR)	0.872	.000***
	Jaccard Similarity (Patch) Jaccard Similarity (FR)	0.835	.000***
	Correlation Similarity (Patch) Correlation Similarity (FR)	0.816	.000***
External Connectivity	External Cohesion (Patch) External Cohesion (FR)	0.836	.000***
	Direct Ties (Patch) Direct Ties (FR)	0.849	.000***
	Indirect Ties (Patch) Indirect Ties (FR)	0.554	.000***
	Indirect Ties FD (Patch) Indirect Ties FD (FR)	0.405	.000***
	Technological Diversity (Patch) Technological Diversity (FR)	0.638	.000***
Network Location	Degree Centrality (Patch) Degree Centrality (FR)	0.835	.000***
	Betweenness Centrality (Patch) Betweenness Centrality (FR)	0.584	.000***
	Closeness Centrality (Patch) Closeness Centrality (FR)	0.234	.000***
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level			

APPENDIX D: ADDITIONAL COMBINED MODELS

TABLE D1: Results of Additional Combined Models for Technical Performance
(Internal Cohesion Measure: Clustering Coefficient, Dependent Variable: CVS Commits, N=2360)

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Independent Variables							
Ambidexterity							
Ambidexterity	5.143 ***	3.317 ***	3.394 ***	3.430 ***	2.997 ***	3.694 ***	-1.970 **
Ambidexterity Squared	-4.850 ***	-3.102 ***	-3.287 ***	-3.320 ***	-3.053 ***	-3.549 ***	.738
Internal Cohesion							
Clustering Coefficient		3.781 ***	4.084 ***	4.312 ***	5.174 ***	4.371 ***	4.192 ***
Repeat Ties							
Third Party Ties							
Jaccard Similarity							
Correlation Similarity							
External Connectivity							
External Cohesion		2.264 **					
External Cohesion Squared		-2.468 **					
Direct Ties			1.830 *	1.562			
Indirect Ties			1.535				
Direct x Indirect Term			-1.958 *				
Indirect Ties FD				1.194			
Direct x Indirect Ties FD				-1.604			
Tech. Diversity					2.477 **		
Tech. Diversity Squared					-2.194 **		
Network Location							
Degree Centrality						1.249	
Betweenness Centrality						-2.222 **	
Closeness Centrality						1.366	
DC x BC						1.908 *	
DC x CC						-1.914 *	

TABLE D1: Cont'd

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Network Location							
Ambi Degree Centrality							-1.919 *
Ambi Betweenness Centrality							2.082 **
Ambi Closeness Centrality							3.270 ***
Number of Projects							1.497
Ambi DC x NP							2.168 **
Ambi BC x NP							-2.149 **
Ambi CC x NP							-2.527 **
NP x Project Team Size							-2.027 **
Control Variables							
Team Human Capital and Ability							
Project Team Size	7.981 ***	6.106 ***	3.843 ***	3.671 ***	5.826 ***	3.654 ***	4.670 ***
User Input and Market Potential							
Bugs Closed	10.277 ***	9.461 ***	9.364 ***	9.393 ***	9.647 ***	9.338 ***	9.238 ***
Support Requests Answered	-.675	-.630	-.590	-.583	-.551	-.645	-.477
Page Views	4.827 ***	4.586 ***	4.605 ***	4.608 ***	4.640 ***	4.554 ***	4.488 ***
Project Life-Cycle Effects							
Project Age (in months)	27.532 ***	27.000 ***	26.892 ***	26.934 ***	26.853 ***	26.999 ***	26.776 ***
Project Age Squared	6.598 ***	6.436 ***	6.435 ***	6.445 ***	6.367 ***	6.443 ***	6.348 ***
Development Status	-1.685 *	-1.647	-1.650 *	-1.653 *	-1.647 *	-1.659 *	-1.595
English	2.668 ***	2.710 ***	2.714 ***	2.700 ***	2.733 ***	2.582 ***	2.898 ***
Results							
F statistics	46.217 ***	44.661 ***	43.598 ***	43.543 ***	44.663 ***	41.900 ***	39.969 ***
Degree of Freedom	42	45	46	46	45	48	51
R	0.675	0.682	0.681	0.681	0.682	0.682	0.685
R Square	0.456	0.465	0.464	0.464	0.465	0.465	0.469
Adj. R Square	0.446	0.454	0.454	0.453	0.454	0.454	0.457
Std. Error of Estimate	2.306	2.289	2.290	2.291	2.289	2.289	2.283
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level							

TABLE D2: Results of Additional Combined Models for Technical Performance
(Internal Cohesion Measure: Repeat Ties, Dependent Variable: CVS Commits, N=2360)

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Independent Variables							
Ambidexterity							
Ambidexterity	5.143 ***	2.492 **	2.831 ***	2.856 ***	2.337 **	3.140 ***	-2.821 ***
Ambidexterity Squared	-4.850 ***	-2.113 **	-2.581 ***	-2.601 ***	-2.251 **	-2.841 ***	1.653 *
Internal Cohesion							
Clustering Coefficient							
Repeat Ties		4.098 ***	4.154 ***	4.299 ***	4.996 ***	4.404 ***	4.541 ***
Third Party Ties							
Jaccard Similarity							
Correlation Similarity							
External Connectivity							
External Cohesion		1.853 *					
External Cohesion Squared		-1.874 *					
Direct Ties			2.661 ***	2.413 **			
Indirect Ties			2.088 **				
Direct x Indirect Term			-2.931 ***				
Indirect Ties FD				1.712 *			
Direct x Indirect Ties FD				-2.584 **			
Tech. Diversity					2.995 ***		
Tech. Diversity Squared					-2.592 ***		
Network Location							
Degree Centrality						1.839 *	
Betweenness Centrality						-2.408 **	
Closeness Centrality						1.520	
DC x BC						1.980 **	
DC x CC						-2.796 ***	

TABLE D2: Cont'd

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Network Location							
Ambi Degree Centrality							-1.567
Ambi Betweenness Centrality							2.030 **
Ambi Closeness Centrality							3.768 ***
Number of Projects							1.888 *
Ambi DC x NP							1.848 *
Ambi BC x NP							-2.080 **
Ambi CC x NP							-3.092 ***
NP x Project Team Size							-2.537 **
Control Variables							
Team Human Capital and Ability							
Project Team Size	7.981 ***	5.490 ***	3.985 ***	3.748 ***	6.379 ***	3.788 ***	5.186 ***
User Input and Market Potential							
Bugs Closed	10.277 ***	9.062 ***	9.199 ***	9.228 ***	9.599 ***	9.173 ***	9.113 ***
Support Requests Answered	-.675	-.374	-.380	-.358	-.309	-.430	-.286
Page Views	4.827 ***	4.876 ***	4.921 ***	4.941 ***	5.026 ***	4.884 ***	4.818 ***
Project Life-Cycle Effects							
Project Age (in months)	27.532 ***	26.992 ***	26.851 ***	26.900 ***	26.791 ***	26.971 ***	26.776 ***
Project Age Squared	6.598 ***	6.603 ***	6.589 ***	6.608 ***	6.538 ***	6.605 ***	6.508 ***
Development Status	-1.685 *	-1.861 *	-1.750 *	-1.760 *	-1.772 *	-1.763 *	-1.643
English	2.668 ***	2.918 ***	2.830 ***	2.813 ***	2.872 ***	2.691 ***	2.975 ***
Results							
F statistics	46.217 ***	44.764 ***	43.621 ***	43.539 ***	44.588 ***	41.911 ***	40.081 ***
Degree of Freedom	42	45	46	46	45	48	51
R	0.675	0.682	0.682	0.681	0.681	0.682	0.685
R Square	0.456	0.465	0.465	0.464	0.464	0.465	0.470
Adj. R Square	0.446	0.455	0.454	0.453	0.454	0.454	0.458
Std. Error of Estimate	2.306	2.287	2.290	2.291	2.289	2.289	2.281
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level							

TABLE D3: Results of Additional Combined Models for Technical Performance
(Internal Cohesion Measure: Third Party Ties, Dependent Variable: CVS Commits, N=2360)

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Independent Variables							
Ambidexterity							
Ambidexterity	5.143 ***	3.186 ***	3.688 ***	3.750 ***	3.395 ***	4.077 ***	-2.585 **
Ambidexterity Squared	-4.850 ***	-3.122 ***	-3.727 ***	-3.795 ***	-3.677 ***	-4.071 ***	.995
Internal Cohesion							
Clustering Coefficient							
Repeat Ties							
Third Party Ties		1.309	0.447	0.359	0.542	0.812	1.324
Jaccard Similarity							
Correlation Similarity							
External Connectivity							
External Cohesion		5.001 ***					
External Cohesion Squared		-4.763 ***					
Direct Ties			3.456 ***	3.227 ***			
Indirect Ties			2.755 ***				
Direct x Indirect Term			-4.051 ***				
Indirect Ties FD				2.291 **			
Direct x Indirect Ties FD				-3.636 ***			
Tech. Diversity					3.332 ***		
Tech. Diversity Squared					-2.626 ***		
Network Location							
Degree Centrality						2.066 **	
Betweenness Centrality						-2.466 **	
Closeness Centrality						1.793 *	
DC x BC						2.034 **	
DC x CC						-3.862 ***	

TABLE D3: Cont'd

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Network Location							
Ambi Degree Centrality							-2.228 **
Ambi Betweenness Centrality							2.210 **
Ambi Closeness Centrality							4.158 ***
Number of Projects							1.946 *
Ambi DC x NP							2.292 **
Ambi BC x NP							-2.217 **
Ambi CC x NP							-3.243 ***
NP x Project Team Size							-3.271 ***
Control Variables							
Team Human Capital and Ability							
Project Team Size	7.981 ***	7.833 ***	3.608 ***	3.312 ***	7.083 ***	3.625 ***	6.702 ***
User Input and Market Potential							
Bugs Closed	10.277 ***	9.830 ***	9.621 ***	9.677 ***	10.293 ***	9.620 ***	9.557 ***
Support Requests Answered	-.675	-.507	-.512	-.492	-.489	-.568	-.427
Page Views	4.827 ***	4.881 ***	4.803 ***	4.820 ***	4.897 ***	4.758 ***	4.701 ***
Project Life-Cycle Effects							
Project Age (in months)	27.532 ***	27.013 ***	27.004 ***	27.065 ***	27.002 ***	27.154 ***	26.969 ***
Project Age Squared	6.598 ***	6.545 ***	6.525 ***	6.548 ***	6.467 ***	6.530 ***	6.445 ***
Development Status	-1.685 *	-1.736 *	-1.744 *	-1.752 *	-1.769 *	-1.765 *	-1.660 *
English	2.668 ***	2.731 ***	2.746 ***	2.720 ***	2.758 ***	2.599 ***	2.892 ***
Results							
F statistics	46.217 ***	44.142 ***	42.934 ***	42.800 ***	43.576 ***	41.187 ***	39.390 ***
Degree of Freedom	42	45	46	46	45	48	51
R	0.675	0.680	0.679	0.678	0.677	0.679	0.682
R Square	0.456	0.462	0.461	0.460	0.459	0.461	0.465
Adj. R Square	0.446	0.451	0.450	0.449	0.448	0.450	0.454
Std. Error of Estimate	2.306	2.295	2.298	2.300	2.302	2.298	2.290
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level							

TABLE D4: Results of Additional Combined Models for Technical Performance
(Internal Cohesion Measure: Jaccard Similarity, Dependent Variable: CVS Commits, N=2360)

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Independent Variables							
Ambidexterity							
Ambidexterity	5.143 ***	3.715 ***	3.446 ***	3.490 ***	2.963 ***	3.757 ***	-1.912 *
Ambidexterity Squared	-4.850 ***	-3.126 ***	-3.049 ***	-3.084 ***	-2.718 ***	-3.325 ***	.879
Internal Cohesion							
Clustering Coefficient							
Repeat Ties							
Third Party Ties							
Jaccard Similarity		4.563 ***	4.964 ***	5.146 ***	6.179 ***	5.091 ***	4.909 ***
Correlation Similarity							
External Connectivity							
External Cohesion		2.797 ***					
External Cohesion Squared		-3.153 ***					
Direct Ties			2.444 **	2.238 **			
Indirect Ties			2.201 **				
Direct x Indirect Term			-2.175 **				
Indirect Ties FD				1.904 *			
Direct x Indirect Ties FD				-1.904 *			
Tech. Diversity					3.964 ***		
Tech. Diversity Squared					-2.876 ***		
Network Location							
Degree Centrality						2.029 **	
Betweenness Centrality						-2.141 **	
Closeness Centrality						2.038 **	
DC x BC						1.747 *	
DC x CC						-2.302 **	

TABLE D4: Cont'd

	Combined Model 1	Combined Model 2.1	Combined Model 2.2	Combined Model 2.3	Combined Model 2.4	Combined Model 3	Combined Model 4
Network Location							
Ambi Degree Centrality							-1.364
Ambi Betweenness Centrality							2.045 **
Ambi Closeness Centrality							2.883 ***
Number of Projects							1.610
Ambi DC x NP							1.739 *
Ambi BC x NP							-2.137 **
Ambi CC x NP							-2.008 **
NP x Project Team Size							-2.292 **
Control Variables							
Team Human Capital and Ability							
Project Team Size	7.981 ***	5.527 ***	2.894 ***	2.726 ***	4.889 ***	2.625 ***	4.327 ***
User Input and Market Potential							
Bugs Closed	10.277 ***	9.013 ***	8.940 ***	8.960 ***	9.191 ***	8.936 ***	8.936 ***
Support Requests Answered	-.675	-.740	-.658	-.650	-.615	-.700	-.481
Page Views	4.827 ***	4.476 ***	4.540 ***	4.543 ***	4.556 ***	4.519 ***	4.490 ***
Project Life-Cycle Effects							
Project Age (in months)	27.532 ***	27.207 ***	27.001 ***	27.056 ***	26.985 ***	27.115 ***	26.984 ***
Project Age Squared	6.598 ***	6.425 ***	6.416 ***	6.431 ***	6.305 ***	6.440 ***	6.377 ***
Development Status	-1.685 *	-1.631	-1.678 *	-1.684 *	-1.655 *	-1.692 *	-1.612
English	2.668 ***	2.696 ***	2.758 ***	2.747 ***	2.726 ***	2.624 ***	2.915 ***
Results							
F statistics	46.217 ***	44.930 ***	43.919 ***	43.860 ***	45.131 ***	42.163 ***	40.209 ***
Degree of Freedom	42	45	46	46	45	48	51
R	0.675	0.683	0.683	0.683	0.684	0.683	0.686
R Square	0.456	0.466	0.466	0.466	0.467	0.467	0.470
Adj. R Square	0.446	0.456	0.456	0.455	0.457	0.456	0.459
Std. Error of Estimate	2.306	2.285	2.286	2.287	2.283	2.286	2.279
*Significant at 10% level, **Significant at 5% level, ***Significant at 1% level							

APPENDIX E: VULNERABILITY DATA

TABLE E1: Vulnerability Data

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#701121	CVE-2006-3134	5/24/2006	3/20/2006	6/27/2006	65	99	0	0	0	0	1	2	2	2
VU#597721	CVE-2006-1176	6/12/2006	6/8/2006	6/21/2006	4	13	0	0	0	0	0	1	1	1
VU#701121	CVE-2006-3134	6/27/2006	5/11/2006	6/27/2006	47	47	0	0	0	0	1	2	2	2
VU#907836	CVE-2006-1467	6/29/2006	4/7/2006	6/29/2006	83	83	0	0	0	0	0	1	1	1
VU#243681	CVE-2006-2199	6/30/2006	6/29/2006	6/29/2006	1	0	1	1	1	1	1	2	2	2
VU#243681	CVE-2006-2199	7/3/2006	6/29/2006	6/29/2006	4	0	1	0	1	1	1	2	2	2
VU#243681	CVE-2006-2199	7/3/2006	6/29/2006	6/29/2006	4	0	0	0	1	1	1	2	2	2
VU#971705	CVE-2006-3687	7/5/2006	2/27/2006	7/17/2006	128	140	0	1	1	0	0	1	1	1
VU#170113	CVE-2006-2198	7/6/2006	6/29/2006	6/29/2006	7	0	1	1	1	1	1	2	2	2
VU#243681	CVE-2006-2199	7/10/2006	6/29/2006	6/29/2006	11	0	1	0	1	0	1	2	2	2
VU#668564	CVE-2006-0007	7/11/2006	5/27/2005	7/11/2006	410	410	0	0	0	0	0	2	2	2
VU#257164	CVE-2006-2372	7/11/2006	12/26/2005	7/11/2006	197	197	0	0	1	0	0	2	2	2
VU#189140	CVE-2006-1314	7/11/2006	3/1/2006	7/11/2006	132	132	0	0	1	0	0	1	1	1
VU#527676	CVE-2006-3811	7/25/2006	5/1/2006	7/25/2006	85	85	1	1	1	1	1	1	1	1
VU#513068	CVE-2006-3838	7/25/2006	5/10/2006	7/26/2006	76	77	0	1	0	0	1	2	2	2
VU#398492	CVE-2006-3812	7/25/2006	5/15/2006	7/25/2006	71	71	0	1	1	1	1	0	1	0
VU#476724	CVE-2006-3801	7/25/2006	5/17/2006	7/25/2006	69	69	0	1	1	1	1	1	1	1
VU#911004	CVE-2006-3810	7/25/2006	5/19/2006	7/25/2006	67	67	0	1	1	1	1	1	1	1
VU#897540	CVE-2006-3804	7/25/2006	5/30/2006	7/25/2006	56	56	0	1	0	1	1	0	0	1
VU#687396	CVE-2006-3807	7/25/2006	6/7/2006	7/25/2006	48	48	1	1	1	1	1	1	1	1
VU#670060	CVE-2006-3677	7/25/2006	6/16/2006	7/25/2006	39	39	0	1	1	1	1	1	1	1
VU#876420	CVE-2006-3805	7/25/2006	6/18/2006	7/25/2006	37	37	0	1	1	1	1	1	1	1
VU#239124	CVE-2006-3113	7/25/2006	6/23/2006	7/25/2006	32	32	0	1	1	1	1	1	1	1
VU#265964	CVE-2006-3803	7/25/2006	7/14/2006	7/25/2006	11	11	0	1	1	1	1	1	1	1
VU#395412	CVE-2006-3747	7/27/2006	7/25/2006	7/27/2006	2	2	1	1	1	1	1	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#372878	CVE-2006-1178	7/28/2006	3/28/2006	7/27/2006	122	121	0	1	1	1	0	0	0	1
VU#395412	CVE-2006-3747	7/28/2006	7/25/2006	7/27/2006	3	2	1	0	1	1	1	2	2	2
VU#395412	CVE-2006-3747	7/28/2006	7/25/2006	7/27/2006	3	2	1	0	1	1	1	2	2	2
VU#395412	CVE-2006-3747	7/28/2006	7/25/2006	7/27/2006	3	2	0	0	1	1	1	2	2	2
VU#395412	CVE-2006-3747	8/1/2006	7/25/2006	7/27/2006	7	2	0	1	1	1	1	2	2	2
VU#199348	CVE-2006-4082	8/3/2006	8/1/2006	8/1/2006	2	0	0	1	1	0	0	2	2	2
VU#230208	CVE-2006-3992	8/7/2006	7/28/2006	7/28/2006	10	0	1	1	1	0	1	1	1	1
VU#159220	CVE-2006-3357	8/8/2006	4/27/2006	7/2/2006	103	66	0	0	1	0	0	1	1	1
VU#119180	CVE-2006-3450	8/8/2006	6/14/2006	8/8/2006	55	55	1	0	1	0	0	1	1	1
VU#262004	CVE-2006-3451	8/8/2006	6/14/2006	8/8/2006	55	55	1	0	1	0	0	1	1	1
VU#655100	CVE-2006-3281	8/8/2006	6/27/2006	6/27/2006	42	0	0	0	1	0	0	1	1	1
VU#883108	CVE-2006-3280	8/8/2006	6/27/2006	6/27/2006	42	0	1	0	1	0	0	1	1	1
VU#936945	CVE-2006-3590	8/8/2006	7/13/2006	7/13/2006	26	0	0	0	0	0	0	1	1	1
VU#401660	CVE-2006-3084	8/8/2006	7/26/2006	7/26/2006	13	0	1	0	1	1	1	2	2	2
VU#580124	CVE-2006-3083	8/8/2006	7/26/2006	7/26/2006	13	0	1	0	1	1	1	2	2	2
VU#650769	CVE-2006-3439	8/8/2006	8/3/2006	8/8/2006	5	5	1	0	1	0	0	2	2	2
VU#401660	CVE-2006-3084	8/9/2006	7/26/2006	7/26/2006	14	0	1	0	1	1	1	2	2	2
VU#401660	CVE-2006-3084	8/10/2006	7/26/2006	7/26/2006	15	0	1	1	1	1	1	2	2	2
VU#580124	CVE-2006-3083	8/10/2006	7/26/2006	7/26/2006	15	0	1	1	1	1	1	2	2	2
VU#395412	CVE-2006-3747	8/25/2006	7/25/2006	7/27/2006	31	2	1	0	1	0	1	2	2	2
VU#300368	CVE-2006-4447	8/31/2006	8/29/2006	8/29/2006	2	0	0	0	1	1	1	2	2	2
VU#697164	CVE-2006-4096	9/5/2006	8/23/2006	9/5/2006	13	13	0	0	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/5/2006	8/23/2006	9/5/2006	13	13	0	0	1	1	1	0	0	1
VU#542197	CVE-2006-4379	9/6/2006	6/22/2006	9/7/2006	76	77	1	1	1	0	0	1	1	1
VU#697164	CVE-2006-4096	9/6/2006	7/3/2006	9/5/2006	65	64	0	1	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/6/2006	8/18/2006	9/5/2006	19	18	0	1	1	1	1	0	0	1
VU#697164	CVE-2006-4096	9/6/2006	8/23/2006	9/5/2006	14	13	1	1	1	1	1	0	0	1
VU#697164	CVE-2006-4096	9/6/2006	8/23/2006	9/5/2006	14	13	1	0	1	1	1	0	0	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#697164	CVE-2006-4096	9/6/2006	8/23/2006	9/5/2006	14	13	0	0	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/6/2006	8/23/2006	9/5/2006	14	13	1	1	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/6/2006	8/23/2006	9/5/2006	14	13	1	0	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/6/2006	8/23/2006	9/5/2006	14	13	0	0	1	1	1	0	0	1
VU#845620	CVE-2006-4339	9/6/2006	9/5/2006	9/5/2006	1	0	1	0	1	1	1	1	1	1
VU#845620	CVE-2006-4339	9/6/2006	9/5/2006	9/5/2006	1	0	1	0	1	1	1	1	1	1
VU#845620	CVE-2006-4339	9/6/2006	9/5/2006	9/5/2006	1	0	0	0	1	1	1	1	1	1
VU#845620	CVE-2006-4339	9/7/2006	9/5/2006	9/5/2006	2	0	1	1	1	1	1	1	1	1
VU#697164	CVE-2006-4096	9/8/2006	8/23/2006	9/5/2006	16	13	0	0	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/8/2006	8/23/2006	9/5/2006	16	13	0	0	1	1	1	0	0	1
VU#697164	CVE-2006-4096	9/8/2006	9/5/2006	9/5/2006	3	0	0	0	1	1	1	0	0	1
VU#845620	CVE-2006-4339	9/8/2006	9/5/2006	9/5/2006	3	0	1	0	1	1	1	1	1	1
VU#915404	CVE-2006-4095	9/8/2006	9/5/2006	9/5/2006	3	0	0	0	1	1	1	0	0	1
VU#697164	CVE-2006-4096	9/9/2006	8/23/2006	9/5/2006	17	13	0	1	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/9/2006	8/23/2006	9/5/2006	17	13	0	1	1	1	1	0	0	1
VU#845620	CVE-2006-4339	9/10/2006	9/5/2006	9/5/2006	5	0	1	1	1	1	1	1	1	1
VU#406236	CVE-2006-0001	9/12/2006	3/8/2005	9/12/2006	553	553	0	0	0	0	0	2	2	2
VU#474593	CVE-2006-3587	9/12/2006	7/10/2006	7/10/2006	64	0	0	1	0	0	1	1	1	1
VU#845620	CVE-2006-4339	9/12/2006	9/5/2006	9/5/2006	7	0	1	1	1	1	1	1	1	1
VU#845620	CVE-2006-4339	9/12/2006	9/5/2006	9/5/2006	7	0	1	0	1	0	1	1	1	1
VU#697164	CVE-2006-4096	9/14/2006	8/23/2006	9/5/2006	22	13	0	0	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/14/2006	8/23/2006	9/5/2006	22	13	0	0	1	1	1	0	0	1
VU#845620	CVE-2006-4339	9/14/2006	9/1/2006	9/5/2006	13	4	0	1	1	1	1	1	1	1
VU#697164	CVE-2006-4096	9/15/2006	8/23/2006	9/5/2006	23	13	0	1	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/15/2006	8/23/2006	9/5/2006	23	13	0	1	1	1	1	0	0	1
VU#845620	CVE-2006-4339	9/19/2006	9/5/2006	9/5/2006	14	0	1	0	1	1	1	1	1	1
VU#697164	CVE-2006-4096	9/21/2006	8/23/2006	9/5/2006	29	13	1	1	1	1	1	0	0	1
VU#915404	CVE-2006-4095	9/21/2006	8/23/2006	9/5/2006	29	13	1	1	1	1	1	0	0	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#484380	CVE-2006-4819	9/21/2006	9/15/2006	10/17/2006	6	32	0	1	1	1	1	1	1	1
VU#845620	CVE-2006-4339	9/22/2006	9/5/2006	9/5/2006	17	0	1	0	1	1	1	1	1	1
VU#845620	CVE-2006-4339	9/22/2006	9/5/2006	9/5/2006	17	0	1	1	1	0	1	1	1	1
VU#416092	CVE-2006-4868	9/26/2006	9/18/2006	9/18/2006	8	0	1	0	1	0	1	2	2	2
VU#386964	CVE-2006-4343	9/28/2006	9/6/2006	9/28/2006	22	22	1	1	1	1	1	0	0	1
VU#547300	CVE-2006-3738	9/28/2006	9/6/2006	9/28/2006	22	22	1	1	1	1	1	2	2	2
VU#386964	CVE-2006-4343	9/28/2006	9/15/2006	9/28/2006	13	13	1	1	1	1	1	0	0	1
VU#386964	CVE-2006-4343	9/28/2006	9/15/2006	9/28/2006	13	13	1	1	1	1	1	0	0	1
VU#386964	CVE-2006-4343	9/28/2006	9/15/2006	9/28/2006	13	13	1	0	1	1	1	0	0	1
VU#386964	CVE-2006-4343	9/28/2006	9/15/2006	9/28/2006	13	13	1	0	1	1	1	0	0	1
VU#547300	CVE-2006-3738	9/28/2006	9/15/2006	9/28/2006	13	13	1	1	1	1	1	2	2	2
VU#547300	CVE-2006-3738	9/28/2006	9/15/2006	9/28/2006	13	13	1	1	1	1	1	2	2	2
VU#547300	CVE-2006-3738	9/28/2006	9/15/2006	9/28/2006	13	13	1	0	1	1	1	2	2	2
VU#547300	CVE-2006-3738	9/28/2006	9/15/2006	9/28/2006	13	13	1	0	1	1	1	2	2	2
VU#247744	CVE-2006-2937	9/28/2006	9/20/2006	9/28/2006	8	8	1	0	1	1	1	0	0	2
VU#787448	CVE-2006-4924	9/28/2006	9/25/2006	9/27/2006	3	2	0	0	1	1	1	0	0	2
VU#386964	CVE-2006-4343	9/29/2006	9/15/2006	9/28/2006	14	13	1	0	1	1	1	0	0	1
VU#386964	CVE-2006-4343	9/29/2006	9/15/2006	9/28/2006	14	13	0	0	1	1	1	0	0	1
VU#547300	CVE-2006-3738	9/29/2006	9/15/2006	9/28/2006	14	13	1	0	1	1	1	2	2	2
VU#547300	CVE-2006-3738	9/29/2006	9/15/2006	9/28/2006	14	13	0	0	1	1	1	2	2	2
VU#262352	CVE-2006-4958	9/29/2006	9/21/2006	9/21/2006	8	0	0	1	0	0	0	1	1	1
VU#787448	CVE-2006-4924	9/29/2006	9/27/2006	9/27/2006	2	0	1	1	1	1	1	0	0	2
VU#787448	CVE-2006-4924	9/29/2006	9/27/2006	9/27/2006	2	0	1	0	1	1	1	0	0	2
VU#247744	CVE-2006-2937	9/29/2006	9/28/2006	9/28/2006	1	0	0	0	1	1	1	0	0	2
VU#787448	CVE-2006-4924	9/30/2006	9/27/2006	9/27/2006	3	0	0	0	1	1	1	0	0	2
VU#851340	CVE-2006-5051	9/30/2006	9/29/2006	9/29/2006	1	0	0	0	1	1	1	2	2	2
VU#845620	CVE-2006-4339	10/2/2006	9/5/2006	9/5/2006	27	0	1	0	1	0	1	1	1	1
VU#851340	CVE-2006-5051	10/2/2006	9/29/2006	9/29/2006	3	0	0	1	1	1	1	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#202604	CVE-2006-5084	10/3/2006	9/23/2006	10/3/2006	10	10	0	1	0	1	0	1	1	1
VU#787448	CVE-2006-4924	10/3/2006	9/27/2006	9/27/2006	6	0	0	0	1	1	1	0	0	2
VU#787448	CVE-2006-4924	10/4/2006	9/27/2006	9/27/2006	7	0	1	1	1	1	1	0	0	2
VU#361792	CVE-2006-5143	10/5/2006	4/7/2006	10/5/2006	181	181	0	1	1	1	0	1	1	1
VU#946820	CVE-2006-4927	10/5/2006	9/19/2006	10/6/2006	16	17	1	0	0	0	0	1	1	1
VU#300368	CVE-2006-4447	10/9/2006	8/29/2006	8/29/2006	41	0	0	1	1	1	1	2	2	2
VU#176556	CVE-2006-3864	10/10/2006	6/14/2006	10/10/2006	118	118	0	0	0	0	0	2	2	2
VU#187028	CVE-2006-3435	10/10/2006	6/14/2006	10/10/2006	118	118	0	0	0	0	0	2	2	2
VU#534276	CVE-2006-3650	10/10/2006	6/14/2006	10/10/2006	118	118	0	0	0	0	0	2	2	2
VU#706668	CVE-2006-2387	10/10/2006	6/15/2006	10/10/2006	117	117	1	0	0	0	0	1	1	1
VU#143292	CVE-2006-3431	10/10/2006	7/3/2006	7/3/2006	99	0	1	0	0	0	0	1	1	1
VU#753044	CVE-2006-3730	10/10/2006	7/18/2006	7/18/2006	84	0	0	0	1	0	0	2	2	2
VU#806548	CVE-2006-4534	10/10/2006	9/5/2006	9/5/2006	35	0	0	0	0	0	0	2	2	2
VU#231204	CVE-2006-4694	10/10/2006	9/27/2006	9/27/2006	13	0	0	0	0	0	0	2	2	2
VU#180864	CVE-2006-4182	10/15/2006	8/16/2006	10/16/2006	60	61	0	1	0	1	1	1	1	1
VU#318764	CVE-2006-5341	10/17/2006	11/1/2005	10/17/2006	350	350	0	0	0	0	0	2	2	2
VU#717140	CVE-2006-5332	10/17/2006	11/1/2005	10/17/2006	350	350	0	0	0	0	0	2	2	2
VU#736324	CVE-2006-5335	10/17/2006	11/1/2005	10/17/2006	350	350	0	0	0	0	0	2	2	2
VU#869292	CVE-2006-5340	10/17/2006	4/19/2006	10/17/2006	181	181	0	0	0	0	0	2	2	2
VU#395412	CVE-2006-3747	10/17/2006	7/25/2006	7/27/2006	84	2	0	0	1	0	1	2	2	2
VU#366020	CVE-2006-4154	10/17/2006	8/16/2006	10/14/2006	62	59	0	1	1	1	1	1	1	1
VU#180864	CVE-2006-4182	10/17/2006	10/16/2006	10/16/2006	1	0	0	0	1	1	1	1	1	1
VU#180864	CVE-2006-4182	10/18/2006	10/16/2006	10/16/2006	2	0	0	0	1	1	1	1	1	1
VU#245984	CVE-2006-4342	10/19/2006	9/7/2006	10/19/2006	42	42	0	0	1	1	0	0	0	2
VU#180864	CVE-2006-4182	10/19/2006	10/16/2006	10/16/2006	3	0	0	1	1	1	1	1	1	1
VU#521252	CVE-2006-5444	10/19/2006	10/17/2006	10/18/2006	2	1	0	0	1	1	1	1	1	1
VU#787448	CVE-2006-4924	10/20/2006	9/27/2006	9/27/2006	23	0	0	0	1	1	1	0	0	2
VU#788860	CVE-2006-5157	10/21/2006	6/27/2006	10/2/2006	116	97	0	0	0	0	0	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#180864	CVE-2006-4182	10/24/2006	10/16/2006	10/16/2006	8	0	0	1	1	1	1	1	1	1
VU#845620	CVE-2006-4339	10/25/2006	9/5/2006	9/5/2006	50	0	0	0	1	0	1	1	1	1
VU#147252	CVE-2006-5379	10/25/2006	10/16/2006	10/16/2006	9	0	1	1	1	1	1	1	1	1
VU#383092	CVE-2005-2454	10/26/2006	7/22/2005	10/18/2006	461	453	0	1	0	0	0	1	1	1
VU#845620	CVE-2006-4339	10/31/2006	9/5/2006	9/5/2006	56	0	0	1	1	1	1	1	1	1
VU#845620	CVE-2006-4339	10/31/2006	9/5/2006	9/5/2006	56	0	1	0	0	0	1	1	1	1
VU#363992	CVE-2006-5468	10/31/2006	10/23/2006	10/27/2006	8	4	0	1	0	1	1	0	0	1
VU#723736	CVE-2006-4805	10/31/2006	10/23/2006	10/27/2006	8	4	0	1	0	1	1	0	0	1
VU#147252	CVE-2006-5379	11/2/2006	10/16/2006	10/16/2006	17	0	0	1	1	0	1	1	1	1
VU#102465	CVE-2007-0603	11/2/2006	10/20/2006	1/25/2007	13	97	0	1	0	0	0	2	2	2
VU#901852	CVE-2006-6603	11/2/2006	10/26/2006	12/8/2006	7	43	0	1	0	0	0	2	2	2
VU#495288	CVE-2006-5464	11/7/2006	8/27/2006	11/8/2006	72	73	0	1	1	1	1	0	0	1
VU#390480	CVE-2006-5748	11/7/2006	10/5/2006	11/8/2006	33	34	0	1	1	1	1	0	0	1
VU#714496	CVE-2006-5463	11/7/2006	10/5/2006	11/8/2006	33	34	0	1	1	1	1	1	1	1
VU#815432	CVE-2006-5747	11/7/2006	10/5/2006	11/8/2006	33	34	0	1	1	1	1	1	1	1
VU#335392	CVE-2006-5462	11/7/2006	10/10/2006	11/8/2006	28	29	0	1	1	1	1	1	1	0
VU#845620	CVE-2006-4339	11/8/2006	9/5/2006	9/5/2006	64	0	1	1	1	1	1	1	1	1
VU#474593	CVE-2006-3587	11/14/2006	7/10/2006	7/10/2006	127	0	1	0	0	0	1	1	1	1
VU#197852	CVE-2006-4687	11/14/2006	7/18/2006	11/14/2006	119	119	0	0	1	0	0	1	1	1
VU#778036	CVE-2006-4691	11/14/2006	7/25/2006	11/14/2006	112	112	0	0	1	0	0	2	2	2
VU#796956	CVE-2006-4511	11/14/2006	8/17/2006	10/3/2006	89	47	1	0	0	0	0	0	0	1
VU#225217	CVE-2006-3890	11/14/2006	8/21/2006	11/14/2006	85	85	0	1	0	0	0	2	2	2
VU#813588	CVE-2006-4446	11/14/2006	8/28/2006	8/28/2006	78	0	0	0	0	0	0	0	0	1
VU#512804	CVE-2006-5198	11/14/2006	8/28/2006	11/14/2006	78	78	0	1	0	0	0	1	1	0
VU#168372	CVE-2006-4640	11/14/2006	9/12/2006	9/12/2006	63	0	1	0	0	0	1	1	1	1
VU#451380	CVE-2006-3311	11/14/2006	9/12/2006	9/12/2006	63	0	1	0	0	0	1	1	1	1
VU#377369	CVE-2006-4777	11/14/2006	9/13/2006	9/13/2006	62	0	0	0	0	0	0	2	2	2
VU#352825	CVE-2006-5864	11/20/2006	11/9/2006	11/9/2006	11	0	1	1	1	1	1	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#300636	CVE-2006-5854	11/21/2006	10/2/2006	11/21/2006	50	50	0	1	1	0	0	1	1	1
VU#653076	CVE-2006-5854	11/21/2006	10/2/2006	11/21/2006	50	50	0	1	1	0	0	1	1	1
VU#352825	CVE-2006-5864	11/24/2006	11/9/2006	11/9/2006	15	0	1	1	1	1	1	1	1	1
VU#845620	CVE-2006-4339	11/28/2006	9/5/2006	9/5/2006	84	0	1	0	0	0	1	1	1	1
VU#848960	CVE-2006-4412	11/28/2006	9/5/2006	11/28/2006	84	84	0	0	1	0	0	1	1	1
VU#870960	CVE-2006-4406	11/28/2006	9/14/2006	11/28/2006	75	75	0	0	1	0	0	1	1	1
VU#191336	CVE-2006-5710	11/28/2006	11/1/2006	11/1/2006	27	0	0	0	1	0	0	1	1	1
VU#335392	CVE-2006-5462	11/28/2006	11/8/2006	11/8/2006	20	0	0	0	1	0	1	1	1	0
VU#335392	CVE-2006-5462	11/28/2006	11/8/2006	11/8/2006	20	0	0	1	1	0	1	1	1	0
VU#427009	CVE-2006-6235	11/28/2006	11/24/2006	12/6/2006	4	12	1	1	1	1	1	2	2	2
VU#221700	CVE-2006-6121	11/29/2006	11/19/2006	11/19/2006	10	0	0	0	1	0	1	2	2	2
VU#210969	CVE-2006-6334	12/4/2006	9/19/2006	11/29/2006	76	71	0	1	0	0	0	1	1	1
VU#448569	CVE-2006-5856	12/5/2006	4/7/2006	12/6/2006	242	243	0	1	0	0	0	1	1	1
VU#350625	CVE-2006-5855	12/5/2006	5/9/2006	12/4/2006	210	209	0	0	1	0	0	2	2	2
VU#198908	CVE-2006-6027	12/5/2006	11/17/2006	11/17/2006	18	0	1	1	0	0	0	2	2	2
VU#989144	CVE-2006-6223	12/6/2006	11/17/2006	11/17/2006	19	0	0	0	0	0	0	0	1	0
VU#427009	CVE-2006-6235	12/6/2006	12/4/2006	12/6/2006	2	2	1	1	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/6/2006	12/4/2006	12/6/2006	2	2	1	0	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/6/2006	12/5/2006	12/6/2006	1	1	0	0	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/7/2006	12/6/2006	12/6/2006	1	0	0	1	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/7/2006	12/6/2006	12/6/2006	1	0	1	0	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/8/2006	12/6/2006	12/6/2006	2	0	0	0	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/8/2006	12/6/2006	12/6/2006	2	0	1	0	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/9/2006	12/6/2006	12/6/2006	3	0	0	1	1	1	1	2	2	2
VU#427009	CVE-2006-6235	12/10/2006	12/6/2006	12/6/2006	4	0	0	1	1	1	1	2	2	2
VU#925529	CVE-2006-6332	12/10/2006	12/7/2006	12/7/2006	3	0	0	1	1	1	1	1	1	1
VU#427009	CVE-2006-6235	12/11/2006	12/6/2006	12/6/2006	5	0	0	0	1	1	1	2	2	2
VU#925529	CVE-2006-6332	12/11/2006	12/7/2006	12/7/2006	4	0	1	0	1	0	1	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#854856	CVE-2006-4704	12/12/2006	6/15/2006	11/1/2006	180	139	1	0	0	0	0	1	1	1
VU#347448	CVE-2006-5581	12/12/2006	8/31/2006	12/12/2006	103	103	1	0	1	0	0	2	2	2
VU#208769	CVE-2006-6134	12/12/2006	11/22/2006	11/22/2006	20	0	1	0	0	0	1	1	1	1
VU#607312	CVE-2006-6222	12/13/2006	8/14/2006	12/14/2006	121	122	0	1	1	0	0	2	2	2
VU#650432	CVE-2006-5822	12/13/2006	8/14/2006	12/14/2006	121	122	0	1	1	0	0	2	2	2
VU#339004	CVE-2006-3896	12/18/2006	8/10/2006	12/18/2006	130	130	0	1	1	0	0	1	1	1
VU#405092	CVE-2006-6503	12/19/2006	9/4/2006	12/19/2006	106	106	0	1	1	1	1	1	1	1
VU#428500	CVE-2006-6502	12/19/2006	9/10/2006	12/19/2006	100	100	0	1	1	1	1	0	0	2
VU#447772	CVE-2006-6498	12/19/2006	9/15/2006	12/19/2006	95	95	0	1	1	1	1	1	1	1
VU#722244	CVE-2006-6500	12/19/2006	9/20/2006	12/19/2006	90	90	0	1	1	1	1	1	1	1
VU#263412	CVE-2006-6501	12/19/2006	9/30/2006	12/19/2006	80	80	0	1	1	1	1	1	1	1
VU#427972	CVE-2006-6499	12/19/2006	10/28/2006	12/19/2006	52	52	0	1	1	1	1	0	0	1
VU#928956	CVE-2006-6504	12/19/2006	11/8/2006	12/19/2006	41	41	0	1	1	1	1	2	2	2
VU#606260	CVE-2006-6497	12/19/2006	11/13/2006	12/19/2006	36	36	0	1	1	1	1	1	1	1
VU#887332	CVE-2006-6505	12/19/2006	11/29/2006	12/19/2006	20	20	0	1	0	1	1	1	1	1
VU#263412	CVE-2006-6501	12/19/2006	12/14/2006	12/19/2006	5	5	1	0	1	1	1	1	1	1
VU#405092	CVE-2006-6503	12/19/2006	12/14/2006	12/19/2006	5	5	1	0	1	1	1	1	1	1
VU#428500	CVE-2006-6502	12/19/2006	12/14/2006	12/19/2006	5	5	1	0	1	1	1	0	0	2
VU#447772	CVE-2006-6498	12/19/2006	12/14/2006	12/19/2006	5	5	1	0	1	1	1	1	1	1
VU#606260	CVE-2006-6497	12/19/2006	12/14/2006	12/19/2006	5	5	1	0	1	1	1	1	1	1
VU#263412	CVE-2006-6501	12/22/2006	12/19/2006	12/19/2006	3	0	1	0	1	1	1	1	1	1
VU#405092	CVE-2006-6503	12/22/2006	12/19/2006	12/19/2006	3	0	1	0	1	1	1	1	1	1
VU#447772	CVE-2006-6498	12/22/2006	12/19/2006	12/19/2006	3	0	1	0	1	1	1	1	1	1
VU#606260	CVE-2006-6497	12/22/2006	12/19/2006	12/19/2006	3	0	1	0	1	1	1	1	1	1
VU#863313	CVE-2006-6761	12/23/2006	10/10/2006	12/23/2006	74	74	0	0	0	0	0	1	1	1
VU#944273	CVE-2006-6762	12/23/2006	10/16/2006	12/23/2006	68	68	0	0	0	0	0	0	0	1
VU#263412	CVE-2006-6501	12/23/2006	12/19/2006	12/19/2006	4	0	0	0	1	1	1	1	1	1
VU#405092	CVE-2006-6503	12/23/2006	12/19/2006	12/19/2006	4	0	0	0	1	1	1	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#428500	CVE-2006-6502	12/23/2006	12/19/2006	12/19/2006	4	0	0	0	1	1	1	0	0	2
VU#447772	CVE-2006-6498	12/23/2006	12/19/2006	12/19/2006	4	0	0	0	1	1	1	1	1	1
VU#606260	CVE-2006-6497	12/23/2006	12/19/2006	12/19/2006	4	0	0	0	1	1	1	1	1	1
VU#258753	CVE-2006-6425	12/25/2006	8/14/2006	12/23/2006	133	131	0	0	0	0	0	2	2	2
VU#381161	CVE-2006-6424	12/25/2006	8/14/2006	12/23/2006	133	131	0	0	0	0	0	2	2	2
VU#912505	CVE-2006-6424	12/25/2006	9/8/2006	12/23/2006	108	106	0	0	0	0	0	2	2	2
VU#845620	CVE-2006-4339	12/28/2006	9/5/2006	9/5/2006	114	0	0	0	1	1	1	1	1	1
VU#481564	CVE-2006-6143	12/28/2006	12/5/2006	1/9/2007	23	35	0	1	1	1	1	2	2	2
VU#263412	CVE-2006-6501	12/29/2006	12/19/2006	12/19/2006	10	0	1	0	1	1	1	1	1	1
VU#405092	CVE-2006-6503	12/29/2006	12/19/2006	12/19/2006	10	0	1	0	1	1	1	1	1	1
VU#447772	CVE-2006-6498	12/29/2006	12/19/2006	12/19/2006	10	0	1	0	1	1	1	1	1	1
VU#606260	CVE-2006-6497	12/29/2006	12/19/2006	12/19/2006	10	0	1	0	1	1	1	1	1	1
VU#722244	CVE-2006-6500	12/29/2006	12/19/2006	12/19/2006	10	0	1	0	1	1	1	1	1	1
VU#263412	CVE-2006-6501	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#263412	CVE-2006-6501	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#405092	CVE-2006-6503	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#405092	CVE-2006-6503	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#428500	CVE-2006-6502	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	0	0	2
VU#428500	CVE-2006-6502	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	0	0	2
VU#428500	CVE-2006-6502	1/2/2007	12/19/2006	12/19/2006	14	0	0	0	1	1	1	0	0	2
VU#447772	CVE-2006-6498	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#447772	CVE-2006-6498	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#606260	CVE-2006-6497	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#606260	CVE-2006-6497	1/2/2007	12/19/2006	12/19/2006	14	0	1	1	1	1	1	1	1	1
VU#263412	CVE-2006-6501	1/4/2007	12/19/2006	12/19/2006	16	0	1	1	1	1	1	1	1	1
VU#405092	CVE-2006-6503	1/4/2007	12/19/2006	12/19/2006	16	0	1	1	1	1	1	1	1	1
VU#428500	CVE-2006-6502	1/4/2007	12/19/2006	12/19/2006	16	0	1	1	1	1	1	0	0	2
VU#447772	CVE-2006-6498	1/4/2007	12/19/2006	12/19/2006	16	0	1	1	1	1	1	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#606260	CVE-2006-6497	1/4/2007	12/19/2006	12/19/2006	16	0	1	1	1	1	1	1	1	1
VU#722244	CVE-2006-6500	1/4/2007	12/19/2006	12/19/2006	16	0	1	1	1	1	1	1	1	1
VU#437300	CVE-2006-6076	1/5/2007	11/21/2006	11/21/2006	45	0	1	0	1	1	0	2	2	2
VU#698924	CVE-2006-5857	1/9/2007	3/9/2006	1/10/2007	306	307	1	1	0	0	0	2	2	2
VU#749964	CVE-2007-0027	1/9/2007	7/11/2006	1/9/2007	182	182	1	0	0	0	0	2	2	2
VU#302836	CVE-2007-0030	1/9/2007	9/14/2006	1/9/2007	117	117	1	0	0	0	0	2	2	2
VU#625532	CVE-2007-0031	1/9/2007	9/22/2006	1/9/2007	109	109	1	0	0	0	0	2	2	2
VU#122084	CVE-2007-0024	1/9/2007	10/3/2006	1/9/2007	98	98	1	0	1	0	0	2	2	2
VU#251969	CVE-2006-6488	1/9/2007	12/4/2006	1/2/2007	36	29	0	0	0	0	0	1	1	1
VU#815960	CVE-2007-0045	1/9/2007	12/29/2006	12/29/2006	11	0	1	1	0	0	1	0	1	0
VU#220288	CVE-2006-5870	1/9/2007	1/4/2007	1/4/2007	5	0	0	0	0	0	1	2	2	2
VU#481564	CVE-2006-6143	1/9/2007	1/4/2007	1/9/2007	5	5	0	0	1	1	1	2	2	2
VU#481564	CVE-2006-6143	1/9/2007	1/4/2007	1/9/2007	5	5	0	0	1	1	1	2	2	2
VU#831452	CVE-2006-6144	1/9/2007	1/4/2007	1/9/2007	5	5	0	0	1	1	1	0	0	1
VU#831452	CVE-2006-6144	1/9/2007	1/4/2007	1/9/2007	5	5	0	0	1	1	1	0	0	1
VU#481564	CVE-2006-6143	1/10/2007	1/4/2007	1/9/2007	6	5	0	0	1	1	1	2	2	2
VU#481564	CVE-2006-6143	1/10/2007	1/4/2007	1/9/2007	6	5	0	0	1	1	1	2	2	2
VU#831452	CVE-2006-6144	1/10/2007	1/4/2007	1/9/2007	6	5	0	0	1	1	1	0	0	1
VU#481564	CVE-2006-6143	1/10/2007	1/9/2007	1/9/2007	1	0	0	1	1	1	1	2	2	2
VU#831452	CVE-2006-6144	1/10/2007	1/9/2007	1/9/2007	1	0	0	1	1	1	1	0	0	1
VU#662400	CVE-2007-0168	1/11/2007	11/1/2006	1/11/2007	71	71	0	0	1	1	0	1	1	1
VU#151032	CVE-2007-0169	1/11/2007	11/8/2006	1/11/2007	64	64	0	0	1	1	0	1	1	1
VU#263412	CVE-2006-6501	1/11/2007	12/19/2006	12/19/2006	23	0	0	0	1	1	1	1	1	1
VU#405092	CVE-2006-6503	1/11/2007	12/19/2006	12/19/2006	23	0	0	0	1	1	1	1	1	1
VU#428500	CVE-2006-6502	1/11/2007	12/19/2006	12/19/2006	23	0	0	0	1	1	1	0	0	2
VU#447772	CVE-2006-6498	1/11/2007	12/19/2006	12/19/2006	23	0	0	0	1	1	1	1	1	1
VU#606260	CVE-2006-6497	1/11/2007	12/19/2006	12/19/2006	23	0	0	0	1	1	1	1	1	1
VU#722244	CVE-2006-6500	1/11/2007	12/19/2006	12/19/2006	23	0	0	0	1	1	1	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#428500	CVE-2006-6502	1/12/2007	12/19/2006	12/19/2006	24	0	0	0	1	1	1	0	0	2
VU#443108	CVE-2006-4097	1/12/2007	1/5/2007	1/5/2007	7	0	1	1	1	1	0	0	0	2
VU#477164	CVE-2006-4098	1/12/2007	1/5/2007	1/5/2007	7	0	1	1	1	1	0	2	2	2
VU#744249	CVE-2007-0105	1/12/2007	1/5/2007	1/8/2007	7	3	1	1	1	1	0	1	1	1
VU#481564	CVE-2006-6143	1/15/2007	1/4/2007	1/9/2007	11	5	0	1	1	1	1	2	2	2
VU#481564	CVE-2006-6143	1/15/2007	1/9/2007	1/9/2007	6	0	1	1	1	1	1	2	2	2
VU#388289	CVE-2007-0243	1/16/2007	6/16/2006	1/16/2007	214	214	0	1	1	0	1	1	1	1
VU#845620	CVE-2006-4339	1/16/2007	9/5/2006	9/5/2006	133	0	0	0	0	0	1	1	1	1
VU#386964	CVE-2006-4343	1/16/2007	9/28/2006	9/28/2006	110	0	0	0	0	0	1	0	0	1
VU#547300	CVE-2006-3738	1/16/2007	9/28/2006	9/28/2006	110	0	0	0	0	0	1	2	2	2
VU#963889	CVE-2007-4474	1/17/2007	10/19/2006	12/20/2007	90	427	1	1	0	0	1	2	2	2
VU#963889	CVE-2007-4474	1/17/2007	10/19/2006	12/20/2007	90	427	1	1	0	0	1	2	2	2
VU#845620	CVE-2006-4339	1/23/2007	9/5/2006	9/5/2006	140	0	0	0	1	0	1	1	1	1
VU#335392	CVE-2006-5462	1/23/2007	11/8/2006	11/8/2006	76	0	0	0	1	0	1	1	1	0
VU#442497	CVE-2007-0015	1/23/2007	1/2/2007	1/2/2007	21	0	0	0	0	0	0	1	1	1
VU#481564	CVE-2006-6143	1/24/2007	1/4/2007	1/9/2007	20	5	0	1	1	1	1	2	2	2
VU#831452	CVE-2006-6144	1/24/2007	1/4/2007	1/9/2007	20	5	0	1	1	1	1	0	0	1
VU#583552	CVE-2006-6292	1/25/2007	11/25/2006	11/30/2006	61	5	1	0	1	0	0	0	0	2
VU#405092	CVE-2006-6503	1/27/2007	12/19/2006	12/19/2006	39	0	1	1	1	1	1	1	1	1
VU#428500	CVE-2006-6502	1/27/2007	12/19/2006	12/19/2006	39	0	1	1	1	1	1	0	0	2
VU#447772	CVE-2006-6498	1/27/2007	12/19/2006	12/19/2006	39	0	1	1	1	1	1	1	1	1
VU#584436	CVE-2007-0669	1/31/2007	1/28/2007	2/8/2007	3	11	1	0	1	1	1	1	1	1
VU#919369	CVE-2007-1350	2/1/2007	12/12/2006	3/7/2007	51	85	1	0	0	0	0	1	1	1
VU#649732	CVE-2007-0454	2/5/2007	1/8/2007	2/5/2007	28	28	0	0	1	1	1	1	1	1
VU#303012	CVE-2007-0446	2/7/2007	10/27/2006	2/8/2007	103	104	0	0	0	0	0	2	2	2
VU#831452	CVE-2006-6144	2/7/2007	1/9/2007	1/9/2007	29	0	1	1	1	1	1	0	0	1
VU#282240	CVE-2007-0856	2/7/2007	1/17/2007	2/7/2007	21	21	0	1	0	0	0	2	2	2
VU#784369	CVE-2007-0325	2/12/2007	10/6/2006	2/15/2007	129	132	0	1	0	0	0	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#613564	CVE-2007-0217	2/13/2007	8/16/2006	2/13/2007	181	181	1	0	1	0	0	2	2	2
VU#205948	CVE-2006-3877	2/13/2007	10/10/2006	10/10/2006	126	0	1	0	0	0	0	2	2	2
VU#589272	CVE-2006-5559	2/13/2007	10/24/2006	10/24/2006	112	0	0	0	1	0	0	2	2	2
VU#167928	CVE-2006-5994	2/13/2007	12/5/2006	12/5/2006	70	0	0	0	0	0	0	2	2	2
VU#166700	CVE-2006-6456	2/13/2007	12/10/2006	12/10/2006	65	0	0	0	0	0	0	2	2	2
VU#996892	CVE-2006-6561	2/13/2007	12/12/2006	12/12/2006	63	0	0	0	0	0	0	2	2	2
VU#412225	CVE-2007-0515	2/13/2007	1/25/2007	1/25/2007	19	0	0	0	0	0	0	2	2	2
VU#613740	CVE-2007-0671	2/13/2007	2/2/2007	2/2/2007	11	0	1	0	0	0	0	2	2	2
VU#881872	CVE-2007-0882	2/13/2007	2/10/2007	2/10/2007	3	0	1	0	1	0	0	2	2	2
VU#522393	CVE-2007-0324	2/15/2007	11/21/2006	2/15/2007	86	86	0	1	0	0	0	1	1	1
VU#240880	CVE-2007-0197	2/15/2007	1/9/2007	1/9/2007	37	0	0	0	1	0	0	1	1	1
VU#349393	CVE-2007-1070	2/15/2007	1/19/2007	2/20/2007	27	32	1	0	0	0	0	2	2	2
VU#794752	CVE-2007-0021	2/15/2007	1/20/2007	1/20/2007	26	0	0	0	0	0	0	1	1	1
VU#315856	CVE-2007-0023	2/15/2007	1/23/2007	1/23/2007	23	0	0	0	0	0	0	2	2	2
VU#836024	CVE-2007-0710	2/15/2007	1/29/2007	1/30/2007	17	1	0	0	0	0	0	0	0	1
VU#308087	CVE-2007-1083	2/16/2007	12/22/2006	2/16/2007	56	56	0	0	0	1	0	2	2	2
VU#196240	CVE-2006-5276	2/19/2007	2/17/2007	2/19/2007	2	2	0	1	0	1	1	2	2	2
VU#441785	CVE-2006-6490	2/22/2007	8/21/2006	2/22/2007	185	185	0	0	0	0	1	2	2	2
VU#551436	CVE-2007-0776	2/23/2007	11/14/2006	2/23/2007	101	101	0	1	1	1	1	2	2	2
VU#269484	CVE-2007-0777	2/23/2007	12/5/2006	2/23/2007	80	80	0	1	1	1	1	2	2	2
VU#761756	CVE-2007-0775	2/23/2007	12/14/2006	2/23/2007	71	71	0	1	1	1	1	1	1	1
VU#377812	CVE-2007-0008	2/23/2007	12/18/2006	2/23/2007	67	67	0	1	1	1	1	1	1	1
VU#592796	CVE-2007-0009	2/23/2007	12/18/2006	2/23/2007	67	67	0	1	1	1	1	1	1	1
VU#885753	CVE-2007-0981	2/23/2007	2/14/2007	2/14/2007	9	0	0	1	1	1	1	1	1	1
VU#196240	CVE-2006-5276	2/23/2007	2/19/2007	2/19/2007	4	0	1	1	1	1	1	2	2	2
VU#393921	CVE-2007-1092	2/25/2007	2/22/2007	2/22/2007	3	0	0	1	1	1	1	2	2	2
VU#377812	CVE-2007-0008	2/26/2007	2/23/2007	2/23/2007	3	0	1	1	1	1	1	1	1	1
VU#377812	CVE-2007-0008	2/26/2007	2/23/2007	2/23/2007	3	0	1	1	1	1	1	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#592796	CVE-2007-0009	2/26/2007	2/23/2007	2/23/2007	3	0	1	1	1	1	1	1	1	1
VU#592796	CVE-2007-0009	2/26/2007	2/23/2007	2/23/2007	3	0	1	1	1	1	1	1	1	1
VU#377812	CVE-2007-0008	2/28/2007	2/23/2007	2/23/2007	5	0	1	0	1	1	1	1	1	1
VU#592796	CVE-2007-0009	2/28/2007	2/23/2007	2/23/2007	5	0	1	0	1	1	1	1	1	1
VU#861817	CVE-2007-0714	3/5/2007	8/14/2006	3/6/2007	203	204	0	1	0	0	0	2	2	2
VU#498553	CVE-2006-3892	3/5/2007	11/17/2006	2/26/2007	108	101	0	0	0	1	0	2	2	2
VU#313225	CVE-2007-0718	3/5/2007	12/6/2006	3/6/2007	89	90	0	1	0	0	0	0	1	1
VU#304064	CVE-2007-0059	3/5/2007	1/3/2007	1/3/2007	61	0	0	0	0	0	0	1	1	1
VU#377812	CVE-2007-0008	3/6/2007	2/23/2007	2/23/2007	11	0	1	0	1	1	1	1	1	1
VU#592796	CVE-2007-0009	3/6/2007	2/23/2007	2/23/2007	11	0	1	0	1	1	1	1	1	1
VU#986425	CVE-2007-1365	3/7/2007	2/20/2007	3/12/2007	15	20	1	0	1	1	0	2	2	2
VU#377812	CVE-2007-0008	3/7/2007	2/23/2007	2/23/2007	12	0	0	0	1	1	1	1	1	1
VU#592796	CVE-2007-0009	3/7/2007	2/23/2007	2/23/2007	12	0	0	0	1	1	1	1	1	1
VU#920689	CVE-2007-1000	3/7/2007	3/6/2007	3/12/2007	1	6	0	1	1	1	1	2	2	2
VU#765096	CVE-2006-5836	3/13/2007	11/9/2006	11/9/2006	124	0	0	0	1	0	0	2	2	2
VU#367424	CVE-2006-6061	3/13/2007	11/20/2006	11/20/2006	113	0	0	0	1	0	0	2	2	2
VU#214040	CVE-2006-6062	3/13/2007	11/21/2006	11/21/2006	112	0	0	0	1	0	0	1	1	1
VU#346656	CVE-2006-6129	3/13/2007	11/26/2006	11/26/2006	107	0	0	0	1	0	0	1	1	1
VU#552136	CVE-2006-5679	3/13/2007	1/10/2007	1/10/2007	62	0	0	0	1	0	0	1	1	1
VU#515792	CVE-2007-0299	3/13/2007	1/11/2007	1/11/2007	61	0	0	0	1	0	0	0	0	2
VU#363112	CVE-2007-0467	3/13/2007	1/28/2007	1/28/2007	44	0	0	0	1	0	0	2	2	2
VU#589097	CVE-2007-1819	3/13/2007	1/30/2007	3/27/2007	42	56	0	0	0	0	0	2	2	2
VU#926551	CVE-2007-1319	3/16/2007	1/12/2006	3/16/2007	428	428	0	1	0	0	0	2	2	2
VU#377812	CVE-2007-0008	3/18/2007	2/23/2007	2/23/2007	23	0	1	1	1	1	1	1	1	1
VU#592796	CVE-2007-0009	3/18/2007	2/23/2007	2/23/2007	23	0	1	1	1	1	1	1	1	1
VU#375353	CVE-2007-1447	3/20/2007	3/16/2007	3/16/2007	4	0	0	0	1	1	0	2	2	2
VU#647273	CVE-2007-1448	3/20/2007	3/16/2007	3/16/2007	4	0	0	0	1	1	0	0	0	1
VU#606700	CVE-2007-1536	3/22/2007	3/19/2007	3/19/2007	3	0	0	0	1	1	1	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#714593	CVE-2007-1498	3/23/2007	3/13/2007	3/13/2007	10	0	0	0	0	0	0	2	2	2
VU#606700	CVE-2007-1536	3/25/2007	3/19/2007	3/19/2007	6	0	1	0	1	1	1	2	2	2
VU#377812	CVE-2007-0008	3/29/2007	2/23/2007	2/23/2007	34	0	1	0	1	0	1	1	1	1
VU#592796	CVE-2007-0009	3/29/2007	2/23/2007	2/23/2007	34	0	1	0	1	0	1	1	1	1
VU#606700	CVE-2007-1536	3/30/2007	3/19/2007	3/19/2007	11	0	1	0	1	1	1	2	2	2
VU#606700	CVE-2007-1536	3/30/2007	3/19/2007	3/19/2007	11	0	1	1	1	1	1	2	2	2
VU#495288	CVE-2006-5464	4/2/2007	11/8/2006	11/8/2006	145	0	0	1	1	0	1	0	0	1
VU#606700	CVE-2007-1536	4/2/2007	3/19/2007	3/19/2007	14	0	0	1	1	1	1	2	2	2
VU#606700	CVE-2007-1536	4/3/2007	3/19/2007	3/19/2007	15	0	1	0	1	1	1	2	2	2
VU#220816	CVE-2007-0956	4/3/2007	3/21/2007	4/3/2007	13	13	0	0	1	1	1	2	2	2
VU#220816	CVE-2007-0956	4/3/2007	3/21/2007	4/3/2007	13	13	1	0	1	1	1	2	2	2
VU#220816	CVE-2007-0956	4/3/2007	3/21/2007	4/3/2007	13	13	0	1	1	1	1	2	2	2
VU#220816	CVE-2007-0956	4/3/2007	3/21/2007	4/3/2007	13	13	1	0	1	1	1	2	2	2
VU#419344	CVE-2007-1216	4/3/2007	3/21/2007	4/3/2007	13	13	0	1	1	1	1	2	2	2
VU#704024	CVE-2007-0957	4/3/2007	3/21/2007	4/3/2007	13	13	0	1	1	1	1	2	2	2
VU#191609	CVE-2007-0038	4/3/2007	3/29/2007	3/29/2007	5	0	1	0	1	0	1	2	2	2
VU#419344	CVE-2007-1216	4/3/2007	4/2/2007	4/3/2007	1	1	0	0	1	1	1	2	2	2
VU#704024	CVE-2007-0957	4/3/2007	4/2/2007	4/3/2007	1	1	0	0	1	1	1	2	2	2
VU#220816	CVE-2007-0956	4/4/2007	3/21/2007	4/3/2007	14	13	0	1	1	1	1	2	2	2
VU#220816	CVE-2007-0956	4/4/2007	3/21/2007	4/3/2007	14	13	1	0	1	0	1	2	2	2
VU#220816	CVE-2007-0956	4/4/2007	3/21/2007	4/3/2007	14	13	1	0	1	1	1	2	2	2
VU#419344	CVE-2007-1216	4/4/2007	3/21/2007	4/3/2007	14	13	0	1	1	1	1	2	2	2
VU#704024	CVE-2007-0957	4/4/2007	3/21/2007	4/3/2007	14	13	0	1	1	1	1	2	2	2
VU#220816	CVE-2007-0956	4/4/2007	4/3/2007	4/3/2007	1	0	1	1	1	1	1	2	2	2
VU#419344	CVE-2007-1216	4/4/2007	4/3/2007	4/3/2007	1	0	1	1	1	1	1	2	2	2
VU#419344	CVE-2007-1216	4/4/2007	4/3/2007	4/3/2007	1	0	1	0	1	1	1	2	2	2
VU#704024	CVE-2007-0957	4/4/2007	4/3/2007	4/3/2007	1	0	1	1	1	1	1	2	2	2
VU#704024	CVE-2007-0957	4/4/2007	4/3/2007	4/3/2007	1	0	1	0	1	1	1	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#754281	CVE-2006-3894	4/5/2007	12/1/2006	5/22/2007	125	172	0	1	1	0	1	0	0	1
VU#730169	CVE-2007-2927	4/5/2007	2/12/2007	8/1/2007	52	170	1	1	1	0	1	0	0	1
VU#220816	CVE-2007-0956	4/5/2007	3/21/2007	4/3/2007	15	13	1	0	1	1	1	2	2	2
VU#419344	CVE-2007-1216	4/5/2007	4/3/2007	4/3/2007	2	0	1	0	1	1	1	2	2	2
VU#704024	CVE-2007-0957	4/5/2007	4/3/2007	4/3/2007	2	0	1	0	1	1	1	2	2	2
VU#704024	CVE-2007-0957	4/5/2007	4/3/2007	4/3/2007	2	0	0	0	1	0	1	2	2	2
VU#704024	CVE-2007-0957	4/6/2007	4/3/2007	4/3/2007	3	0	1	0	1	1	1	2	2	2
VU#728057	CVE-2007-1205	4/10/2007	11/12/2006	4/10/2007	149	149	0	0	1	0	0	2	2	2
VU#337953	CVE-2007-1206	4/10/2007	12/12/2006	4/10/2007	119	119	1	0	1	0	0	2	2	2
VU#740636	CVE-2006-6797	4/10/2007	12/15/2006	12/15/2006	116	0	1	0	1	0	0	2	0	2
VU#219848	CVE-2007-1209	4/10/2007	1/19/2007	4/10/2007	81	81	1	0	1	0	0	2	2	2
VU#120241	CVE-2007-1891	4/11/2007	4/3/2007	4/16/2007	8	13	0	1	0	0	0	2	2	2
VU#488424	CVE-2007-2508	4/13/2007	2/1/2007	5/7/2007	71	95	0	0	0	0	0	2	2	2
VU#419344	CVE-2007-1216	4/19/2007	4/3/2007	4/3/2007	16	0	0	0	1	0	1	2	2	2
VU#704024	CVE-2007-0957	4/19/2007	4/3/2007	4/3/2007	16	0	0	0	1	0	1	2	2	2
VU#821865	CVE-2007-2883	4/19/2007	4/4/2007	5/24/2007	15	50	1	0	1	0	0	1	1	1
VU#979825	CVE-2007-2139	4/24/2007	3/8/2007	4/24/2007	47	47	0	0	1	1	0	2	2	2
VU#151305	CVE-2007-1785	4/24/2007	3/29/2007	3/29/2007	26	0	0	0	1	1	0	2	2	2
VU#426737	CVE-2007-2929	4/25/2007	2/1/2007	8/14/2007	83	194	0	0	0	0	1	0	1	1
VU#570705	CVE-2007-2240	4/25/2007	2/1/2007	8/14/2007	83	194	0	0	0	0	1	0	1	1
VU#599657	CVE-2007-2928	4/25/2007	2/1/2007	8/14/2007	83	194	0	0	0	0	1	0	1	1
VU#426737	CVE-2007-2929	4/25/2007	3/13/2007	8/14/2007	43	154	0	0	0	0	1	0	1	1
VU#570705	CVE-2007-2240	4/25/2007	3/13/2007	8/14/2007	43	154	0	0	0	0	1	0	1	1
VU#599657	CVE-2007-2928	4/25/2007	3/13/2007	8/14/2007	43	154	0	0	0	0	1	0	1	1
VU#267289	CVE-2007-2242	4/26/2007	4/24/2007	4/24/2007	2	0	0	0	1	1	1	0	0	2
VU#773720	CVE-2007-2446	4/28/2007	4/25/2007	5/14/2007	3	19	1	0	1	1	1	2	2	2
VU#718460	CVE-2007-2241	4/30/2007	4/29/2007	5/1/2007	1	2	0	1	1	1	1	0	0	2
VU#420668	CVE-2007-2175	5/1/2007	4/20/2007	4/20/2007	11	0	0	0	0	0	0	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#305657	CVE-2007-3624	5/2/2007	1/11/2007	7/5/2007	111	175	0	1	0	0	0	2	2	2
VU#679041	CVE-2007-3614	5/2/2007	1/11/2007	7/5/2007	111	175	0	1	0	0	0	1	1	1
VU#355809	CVE-2007-2239	5/3/2007	3/27/2007	5/3/2007	37	37	0	0	0	0	0	2	2	2
VU#253825	CVE-2007-1214	5/8/2007	2/8/2007	5/8/2007	89	89	1	0	0	0	0	1	1	1
VU#332404	CVE-2007-0870	5/8/2007	2/9/2007	2/9/2007	88	0	0	0	0	0	0	2	2	2
VU#555489	CVE-2007-1202	5/8/2007	2/27/2007	5/8/2007	70	70	0	0	0	0	0	1	1	1
VU#555920	CVE-2007-1748	5/8/2007	4/13/2007	4/13/2007	25	0	1	0	1	0	0	2	2	2
VU#718460	CVE-2007-2241	5/9/2007	5/1/2007	5/1/2007	8	0	0	0	1	1	1	0	0	2
VU#268336	CVE-2007-2447	5/9/2007	5/7/2007	5/14/2007	2	7	1	0	1	1	1	1	1	1
VU#589188	CVE-2007-4470	5/10/2007	10/6/2006	9/6/2007	216	335	0	1	0	1	0	2	2	2
VU#680616	CVE-2007-2522	5/12/2007	11/6/2006	5/11/2007	187	186	0	0	0	1	0	2	2	2
VU#788416	CVE-2007-2523	5/12/2007	2/7/2007	5/11/2007	94	93	0	0	0	1	0	2	2	2
VU#773720	CVE-2007-2446	5/14/2007	5/8/2007	5/14/2007	6	6	0	0	1	1	1	2	2	2
VU#268336	CVE-2007-2447	5/14/2007	5/11/2007	5/14/2007	3	3	0	0	1	1	1	1	1	1
VU#268336	CVE-2007-2447	5/15/2007	5/14/2007	5/14/2007	1	0	1	1	1	1	1	1	1	1
VU#773720	CVE-2007-2446	5/15/2007	5/14/2007	5/14/2007	1	0	1	1	1	1	1	2	2	2
VU#983953	CVE-2007-1689	5/16/2007	2/12/2007	5/16/2007	93	93	0	0	0	0	0	2	2	2
VU#267289	CVE-2007-2242	5/16/2007	4/24/2007	4/24/2007	22	0	0	0	1	0	1	0	0	2
VU#267289	CVE-2007-2242	5/16/2007	4/24/2007	4/24/2007	22	0	0	0	1	1	1	0	0	2
VU#268336	CVE-2007-2447	5/16/2007	5/14/2007	5/14/2007	2	0	1	1	1	1	1	1	1	1
VU#684664	CVE-2007-2445	5/17/2007	5/8/2007	5/16/2007	9	8	1	1	1	1	1	0	0	1
VU#684664	CVE-2007-2445	5/17/2007	5/8/2007	5/16/2007	9	8	0	0	1	1	1	0	0	1
VU#789121	CVE-2007-2238	5/21/2007	12/14/2006	4/15/2009	158	853	0	0	0	0	0	2	2	2
VU#754281	CVE-2006-3894	5/22/2007	11/30/2006	5/22/2007	173	173	1	1	1	1	1	0	0	1
VU#524681	CVE-2007-0328	5/22/2007	3/6/2007	5/31/2007	77	86	0	0	0	0	1	2	2	2
VU#746889	CVE-2007-2881	5/25/2007	3/20/2007	5/25/2007	66	66	1	1	1	0	0	2	2	2
VU#609956	CVE-2007-2868	5/30/2007	4/11/2007	5/31/2007	49	50	0	1	1	1	1	2	2	2
VU#751636	CVE-2007-2867	5/30/2007	4/11/2007	5/31/2007	49	50	0	1	1	1	1	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#684664	CVE-2007-2445	5/31/2007	5/8/2007	5/16/2007	23	8	0	1	1	1	1	0	0	1
VU#105105	CVE-2007-2864	6/5/2007	2/16/2007	6/5/2007	109	109	0	0	0	1	0	2	2	2
VU#739409	CVE-2007-2863	6/5/2007	2/16/2007	6/5/2007	109	109	0	0	0	1	0	2	2	2
VU#200928	CVE-2007-3316	6/7/2007	6/6/2007	6/20/2007	1	14	1	0	0	1	1	2	2	2
VU#983249	CVE-2007-2921	6/11/2007	8/17/2006	6/13/2007	298	300	0	1	0	0	0	2	2	2
VU#507433	CVE-2007-2222	6/12/2007	10/16/2006	6/12/2007	239	239	0	0	1	0	0	2	2	2
VU#810073	CVE-2007-2218	6/12/2007	3/19/2007	6/12/2007	85	85	0	0	1	0	0	2	2	2
VU#267289	CVE-2007-2242	6/14/2007	4/24/2007	4/24/2007	51	0	0	1	1	1	1	0	0	2
VU#187033	CVE-2007-2478	6/18/2007	5/4/2007	6/18/2007	45	45	0	0	0	0	0	2	2	2
VU#399896	CVE-2007-2949	6/18/2007	6/14/2007	7/3/2007	4	19	0	0	0	1	1	1	1	1
VU#267289	CVE-2007-2242	6/20/2007	4/24/2007	4/24/2007	57	0	1	0	1	0	1	0	0	2
VU#684664	CVE-2007-2445	6/22/2007	5/8/2007	5/16/2007	45	8	0	0	1	1	1	0	0	1
VU#845620	CVE-2006-4339	6/25/2007	9/5/2006	9/5/2006	293	0	0	1	0	0	1	1	1	1
VU#356961	CVE-2007-2442	6/26/2007	6/18/2007	6/26/2007	8	8	1	0	1	1	1	2	2	2
VU#365313	CVE-2007-2443	6/26/2007	6/18/2007	6/26/2007	8	8	1	1	1	1	1	2	2	2
VU#365313	CVE-2007-2443	6/26/2007	6/18/2007	6/26/2007	8	8	1	0	1	1	1	2	2	2
VU#365313	CVE-2007-2443	6/26/2007	6/18/2007	6/26/2007	8	8	0	0	1	1	1	2	2	2
VU#554257	CVE-2007-2798	6/26/2007	6/18/2007	6/26/2007	8	8	1	1	1	1	1	2	2	2
VU#554257	CVE-2007-2798	6/26/2007	6/18/2007	6/26/2007	8	8	1	0	1	1	1	2	2	2
VU#770904	CVE-2007-3410	6/27/2007	6/26/2007	6/26/2007	1	0	1	0	1	1	1	2	2	2
VU#684664	CVE-2007-2445	6/28/2007	5/8/2007	5/16/2007	51	8	1	0	1	0	1	0	0	1
VU#356961	CVE-2007-2442	6/28/2007	6/18/2007	6/26/2007	10	8	0	1	1	1	1	2	2	2
VU#365313	CVE-2007-2443	6/28/2007	6/18/2007	6/26/2007	10	8	0	1	1	1	1	2	2	2
VU#554257	CVE-2007-2798	6/28/2007	6/18/2007	6/26/2007	10	8	0	1	1	1	1	2	2	2
VU#138545	CVE-2007-2788	6/29/2007	6/4/2007	6/4/2007	25	0	1	0	1	0	1	1	1	1
VU#718460	CVE-2007-2241	7/1/2007	5/1/2007	5/1/2007	61	0	0	1	1	1	1	0	0	2
VU#356961	CVE-2007-2442	7/2/2007	6/18/2007	6/26/2007	14	8	0	0	1	0	1	2	2	2
VU#138457	CVE-2007-3457	7/10/2007	3/12/2007	7/10/2007	120	120	0	1	0	0	1	0	1	0

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#730785	CVE-2007-3456	7/10/2007	3/12/2007	7/10/2007	120	120	0	1	0	0	1	2	2	2
VU#110297	CVE-2007-2022	7/10/2007	4/12/2007	4/12/2007	89	0	0	1	0	0	1	1	1	1
VU#213697	CVE-2007-3509	7/11/2007	5/1/2007	7/11/2007	71	71	0	0	1	0	0	1	1	1
VU#730785	CVE-2007-3456	7/12/2007	7/10/2007	7/10/2007	2	0	1	0	1	1	1	2	2	2
VU#143297	CVE-2007-3089	7/17/2007	5/19/2007	6/4/2007	59	16	0	1	1	1	1	0	1	0
VU#358017	CVE-2007-3670	7/17/2007	6/13/2007	7/10/2007	34	27	0	1	1	1	1	0	1	0
VU#871497	CVE-2007-3375	7/18/2007	6/25/2007	6/25/2007	23	0	0	1	1	0	1	1	1	1
VU#730785	CVE-2007-3456	7/19/2007	7/10/2007	7/10/2007	9	0	0	0	1	1	1	2	2	2
VU#252735	CVE-2007-2926	7/24/2007	5/29/2007	7/24/2007	56	56	0	1	1	1	1	0	1	0
VU#205073	CVE-2007-4473	7/25/2007	1/13/2007	12/14/2007	193	335	0	1	0	0	0	2	2	2
VU#554257	CVE-2007-2798	7/25/2007	6/18/2007	6/26/2007	37	8	0	1	1	1	1	2	2	2
VU#187297	CVE-2007-2925	7/26/2007	7/24/2007	7/24/2007	2	0	0	1	1	1	1	1	1	0
VU#109056	CVE-2007-4218	7/27/2007	6/14/2007	8/21/2007	43	68	0	0	0	0	0	2	2	2
VU#204448	CVE-2007-4218	7/27/2007	6/14/2007	8/21/2007	43	68	0	0	0	0	0	2	2	2
VU#959400	CVE-2007-4219	7/27/2007	6/14/2007	8/21/2007	43	68	0	0	0	0	0	2	2	2
VU#252735	CVE-2007-2926	7/30/2007	7/24/2007	7/24/2007	6	0	0	0	1	1	1	0	1	0
VU#783400	CVE-2007-3845	7/30/2007	7/25/2007	7/25/2007	5	0	0	1	1	1	1	2	2	2
VU#845708	CVE-2007-2401	7/31/2007	6/14/2007	6/21/2007	47	7	0	0	1	0	0	0	1	0
VU#252735	CVE-2007-2926	8/1/2007	7/24/2007	7/24/2007	8	0	1	0	1	1	1	0	1	0
VU#970849	CVE-2007-3644	8/3/2007	7/12/2007	7/12/2007	22	0	0	0	1	1	1	0	0	1
VU#730785	CVE-2007-3456	8/8/2007	7/10/2007	7/10/2007	29	0	0	1	1	1	1	2	2	2
VU#970849	CVE-2007-3644	8/8/2007	7/12/2007	7/12/2007	27	0	0	1	1	1	1	0	0	1
VU#747233	CVE-2007-0319	8/12/2007	12/12/2006	8/14/2007	243	245	0	0	0	0	0	1	1	1
VU#993544	CVE-2007-3382	8/13/2007	7/2/2007	8/13/2007	42	42	1	1	1	1	1	1	0	0
VU#361968	CVE-2007-2223	8/14/2007	5/17/2006	8/14/2007	454	454	1	0	1	0	0	2	2	2
VU#468800	CVE-2007-1749	8/14/2007	10/24/2006	8/14/2007	294	294	1	0	1	0	0	2	2	2
VU#121024	CVE-2007-3032	8/14/2007	3/21/2007	8/14/2007	146	146	0	0	1	0	0	1	1	1
VU#558648	CVE-2007-3033	8/14/2007	3/21/2007	8/14/2007	146	146	0	0	0	0	0	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#640136	CVE-2007-3034	8/14/2007	3/27/2007	8/14/2007	140	140	1	0	1	0	0	2	2	2
VU#554257	CVE-2007-2798	8/15/2007	6/18/2007	6/26/2007	58	8	0	0	1	0	1	2	2	2
VU#267289	CVE-2007-2242	8/16/2007	4/24/2007	4/24/2007	114	0	0	0	1	0	1	0	0	2
VU#515968	CVE-2007-4391	8/21/2007	8/15/2007	8/15/2007	6	0	0	0	0	0	0	2	2	2
VU#252735	CVE-2007-2926	8/22/2007	7/24/2007	7/24/2007	29	0	1	0	1	0	1	0	1	0
VU#927905	CVE-2007-2930	8/29/2007	7/26/2007	8/27/2007	34	32	0	0	1	1	1	0	1	0
VU#981849	CVE-2007-4827	8/30/2007	8/20/2007	9/20/2007	10	31	1	0	1	0	0	1	1	1
VU#907481	CVE-2007-0322	9/4/2007	8/21/2006	9/4/2007	379	379	0	1	0	0	0	2	2	2
VU#979638	CVE-2007-4471	9/4/2007	8/21/2006	9/4/2007	379	379	0	1	0	0	0	2	2	2
VU#377544	CVE-2007-4000	9/4/2007	8/23/2007	9/4/2007	12	12	1	0	1	1	1	2	2	2
VU#883632	CVE-2007-3999	9/4/2007	8/23/2007	9/4/2007	12	12	1	0	1	1	1	2	2	2
VU#377544	CVE-2007-4000	9/4/2007	8/24/2007	9/4/2007	11	11	0	0	1	1	1	2	2	2
VU#883632	CVE-2007-3999	9/4/2007	8/24/2007	9/4/2007	11	11	1	1	1	1	1	2	2	2
VU#883632	CVE-2007-3999	9/4/2007	8/24/2007	9/4/2007	11	11	1	0	1	1	1	2	2	2
VU#787448	CVE-2006-4924	9/6/2007	9/27/2006	9/27/2006	344	0	0	1	1	0	1	0	0	2
VU#166521	CVE-2007-2931	9/11/2007	1/31/2007	1/31/2007	223	0	1	1	0	0	0	2	2	2
VU#196240	CVE-2006-5276	9/11/2007	2/19/2007	2/19/2007	204	0	0	0	0	0	1	2	2	2
VU#716872	CVE-2007-3040	9/11/2007	7/9/2007	9/11/2007	64	64	0	0	1	0	0	2	2	2
VU#377544	CVE-2007-4000	9/11/2007	8/24/2007	9/4/2007	18	11	0	1	1	1	1	2	2	2
VU#883632	CVE-2007-3999	9/11/2007	8/24/2007	9/4/2007	18	11	1	1	1	1	1	2	2	2
VU#854769	CVE-2007-0326	9/17/2007	8/16/2006	9/14/2007	397	394	0	1	0	1	0	2	2	2
VU#544656	CVE-2007-4619	9/17/2007	8/29/2007	10/11/2007	19	43	0	1	1	1	1	2	2	2
VU#751808	CVE-2007-4673	9/18/2007	9/12/2007	9/12/2007	6	0	0	1	1	1	1	2	2	2
VU#377544	CVE-2007-4000	9/28/2007	8/24/2007	9/4/2007	35	11	1	0	1	1	1	2	2	2
VU#883632	CVE-2007-3999	9/28/2007	8/24/2007	9/4/2007	35	11	1	0	1	1	1	2	2	2
VU#639169	CVE-2007-5602	10/2/2007	8/21/2007	1/30/2008	42	162	0	1	0	0	0	2	2	2
VU#751808	CVE-2007-4673	10/3/2007	9/12/2007	9/12/2007	21	0	0	0	0	0	1	2	2	2
VU#138633	CVE-2007-6033	10/10/2007	10/2/2007	11/19/2007	8	48	0	1	1	0	1	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#349217	CVE-2007-5334	10/18/2007	8/5/2007	10/19/2007	74	75	0	1	1	1	1	0	1	0
VU#298521	CVE-2007-5603	10/19/2007	9/20/2007	11/1/2007	29	42	0	0	1	1	0	2	2	2
VU#403150	CVE-2007-3896	10/22/2007	7/25/2007	7/25/2007	89	0	0	1	0	0	1	2	2	2
VU#883632	CVE-2007-3999	10/22/2007	8/24/2007	9/4/2007	59	11	0	0	1	0	1	2	2	2
VU#759385	CVE-2007-5080	10/25/2007	10/2/2006	10/29/2007	388	392	0	1	0	1	0	2	2	2
VU#449089	CVE-2007-2919	10/25/2007	2/21/2007	6/5/2007	246	104	0	1	0	0	0	2	2	2
VU#871673	CVE-2007-5601	10/25/2007	10/16/2007	10/18/2007	9	2	0	0	0	1	1	2	2	2
VU#446897	CVE-2007-4351	10/31/2007	10/16/2007	10/31/2007	15	15	0	1	1	1	1	2	2	2
VU#446897	CVE-2007-4351	10/31/2007	10/19/2007	10/31/2007	12	12	1	0	1	1	1	2	2	2
VU#446897	CVE-2007-4351	11/1/2007	10/20/2007	10/31/2007	12	11	1	0	1	1	1	2	2	2
VU#690515	CVE-2007-4676	11/5/2007	9/14/2007	11/5/2007	52	52	0	1	0	0	0	2	2	2
VU#446897	CVE-2007-4351	11/6/2007	10/17/2007	10/31/2007	20	14	0	1	1	1	1	2	2	2
VU#484649	CVE-2007-3898	11/13/2007	4/30/2007	11/13/2007	197	197	0	0	1	0	0	0	1	1
VU#403150	CVE-2007-3896	11/13/2007	7/25/2007	7/25/2007	111	0	0	0	1	0	1	2	2	2
VU#498105	CVE-2007-4682	11/14/2007	10/24/2006	10/14/2007	386	355	0	1	1	0	0	1	1	1
VU#212984	CVE-2007-5615	11/18/2007	10/25/2007	11/3/2007	24	9	0	1	1	1	1	0	1	0
VU#237888	CVE-2007-5613	11/18/2007	10/25/2007	11/5/2007	24	11	0	1	1	1	1	0	1	0
VU#438616	CVE-2007-5614	11/18/2007	10/25/2007	11/5/2007	24	11	0	1	1	1	1	1	1	1
VU#715737	CVE-2007-5947	11/26/2007	11/7/2007	11/7/2007	19	0	1	1	1	1	1	0	1	0
VU#203220	CVE-2008-0006	12/2/2007	9/26/2007	1/17/2008	67	113	1	0	1	1	1	1	1	1
VU#305208	CVE-2008-2462	12/5/2007	11/28/2007	12/5/2007	7	7	0	1	1	1	0	0	1	0
VU#438395	CVE-2007-6015	12/6/2007	11/22/2007	12/10/2007	14	18	1	0	1	1	1	2	2	2
VU#110947	CVE-2008-0177	12/8/2007	11/30/2007	2/6/2008	8	68	0	1	1	0	1	0	0	2
VU#570089	CVE-2007-6255	12/11/2007	5/29/2007	12/11/2007	196	196	0	0	0	0	0	2	2	2
VU#804089	CVE-2007-3901	12/11/2007	9/28/2007	12/11/2007	74	74	1	0	1	0	1	2	2	2
VU#659761	CVE-2007-6166	12/13/2007	11/23/2007	11/23/2007	20	0	0	0	0	0	1	2	2	2
VU#120593	CVE-2007-6330	12/14/2007	5/29/2007	12/11/2007	199	196	0	0	0	0	0	2	2	2
VU#232881	CVE-2007-6239	12/18/2007	11/27/2007	11/27/2007	21	0	0	0	1	1	1	0	0	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#929656	CVE-2007-6372	12/18/2007	12/12/2007	12/12/2007	6	0	0	1	1	0	1	0	0	2
VU#970849	CVE-2007-3644	1/8/2008	7/12/2007	7/12/2007	180	0	0	1	1	1	1	0	0	1
VU#232881	CVE-2007-6239	1/9/2008	11/27/2007	11/27/2007	43	0	0	0	1	1	1	0	0	1
VU#232881	CVE-2007-6239	1/9/2008	11/27/2007	11/27/2007	43	0	1	0	1	1	1	0	0	1
VU#308556	CVE-2008-0176	1/11/2008	12/20/2007	1/24/2008	22	35	0	0	1	0	0	2	2	2
VU#203611	CVE-2008-0122	1/14/2008	12/10/2007	12/10/2007	35	0	0	1	1	1	1	2	2	2
VU#474433	CVE-2007-4467	1/15/2008	8/15/2006	8/28/2007	518	378	0	0	0	0	0	2	2	2
VU#180876	CVE-2008-0174	1/17/2008	12/20/2007	1/24/2008	28	35	0	1	1	0	0	1	0	0
VU#203220	CVE-2008-0006	1/18/2008	1/8/2008	1/17/2008	10	9	1	0	1	1	1	1	1	1
VU#203220	CVE-2008-0006	1/18/2008	1/17/2008	1/17/2008	1	0	1	1	1	1	1	1	1	1
VU#203220	CVE-2008-0006	1/20/2008	1/17/2008	1/17/2008	3	0	1	1	1	1	1	1	1	1
VU#158609	CVE-2008-0401	1/22/2008	10/24/2007	1/24/2008	90	92	0	1	1	0	0	2	2	2
VU#203220	CVE-2008-0006	1/22/2008	1/17/2008	1/17/2008	5	0	0	0	1	1	1	1	1	1
VU#203220	CVE-2008-0006	1/23/2008	1/17/2008	1/17/2008	6	0	0	0	1	1	1	1	1	1
VU#339345	CVE-2008-0175	1/31/2008	12/20/2007	1/24/2008	42	35	0	1	1	0	0	1	1	1
VU#110947	CVE-2008-0177	2/1/2008	11/30/2007	2/6/2008	63	68	1	0	1	1	1	0	0	2
VU#110947	CVE-2008-0177	2/5/2008	11/30/2007	2/6/2008	67	68	0	1	1	0	1	0	0	2
VU#112179	CVE-2008-0234	2/6/2008	1/10/2008	1/10/2008	27	0	1	0	0	0	0	2	2	2
VU#879056	CVE-2008-0419	2/7/2008	10/20/2007	2/7/2008	110	110	0	1	1	1	1	2	2	2
VU#309608	CVE-2008-0418	2/7/2008	1/22/2008	2/7/2008	16	16	0	1	1	1	1	1	0	0
VU#203220	CVE-2008-0006	2/8/2008	1/17/2008	1/17/2008	22	0	1	0	1	1	1	1	1	1
VU#862600	CVE-2007-3383	2/9/2008	7/2/2007	7/21/2007	222	19	0	1	1	1	1	0	1	0
VU#228569	CVE-2008-0077	2/12/2008	10/24/2007	2/12/2008	111	111	1	0	1	0	0	2	2	2
VU#264385	CVE-2008-0556	2/13/2008	12/12/2007	2/13/2008	63	63	1	0	0	1	0	1	1	1
VU#110947	CVE-2008-0177	2/14/2008	11/30/2007	2/6/2008	76	68	0	1	1	1	1	0	0	2
VU#929656	CVE-2007-6372	2/24/2008	12/12/2007	12/12/2007	74	0	0	1	1	0	1	0	0	2
VU#661651	CVE-2008-0304	2/26/2008	1/14/2008	2/26/2008	43	43	1	1	0	1	1	1	1	1
VU#404515	CVE-2008-1145	3/1/2008	2/20/2008	3/6/2008	10	15	1	1	1	1	1	1	0	0

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#223028	CVE-2008-1196	3/4/2008	2/1/2008	3/6/2008	32	34	0	0	1	0	1	1	1	1
VU#512491	CVE-2008-0072	3/5/2008	3/3/2008	3/5/2008	2	2	1	0	1	1	1	1	1	1
VU#512491	CVE-2008-0072	3/5/2008	3/4/2008	3/5/2008	1	1	1	1	1	1	1	1	1	1
VU#512491	CVE-2008-0072	3/6/2008	3/5/2008	3/5/2008	1	0	0	0	0	1	1	1	1	1
VU#393305	CVE-2008-0110	3/11/2008	7/3/2007	3/11/2008	252	252	1	0	0	0	0	2	2	2
VU#362849	CVE-2007-6253	3/11/2008	11/19/2007	3/11/2008	113	113	0	0	0	0	0	2	2	2
VU#248372	CVE-2008-1262	3/13/2008	12/27/2007	3/6/2008	77	70	0	1	1	0	0	2	2	2
VU#329673	CVE-2007-6254	3/18/2008	5/9/2007	3/18/2008	314	314	0	0	0	0	0	2	2	2
VU#374121	CVE-2008-0947	3/18/2008	3/6/2008	3/18/2008	12	12	0	0	1	1	1	2	2	2
VU#895609	CVE-2008-0062	3/18/2008	3/6/2008	3/18/2008	12	12	0	0	1	0	1	2	2	2
VU#895609	CVE-2008-0062	3/18/2008	3/6/2008	3/18/2008	12	12	0	0	1	1	1	2	2	2
VU#914785	CVE-2007-1682	3/19/2008	7/7/2006	8/25/2007	621	414	0	1	0	1	0	2	2	2
VU#374121	CVE-2008-0947	3/19/2008	3/6/2008	3/18/2008	13	12	0	1	1	1	1	2	2	2
VU#895609	CVE-2008-0062	3/19/2008	3/6/2008	3/18/2008	13	12	0	1	1	1	1	2	2	2
VU#776931	CVE-2008-0660	3/21/2008	11/22/2007	11/22/2007	120	0	0	1	0	0	0	2	2	2
VU#466521	CVE-2008-1233	3/25/2008	1/6/2008	3/25/2008	79	79	0	1	1	1	1	1	1	1
VU#858595	CVE-2008-1100	4/14/2008	3/10/2008	4/14/2008	35	35	0	1	0	1	1	2	2	2
VU#441529	CVE-2008-1380	4/16/2008	3/27/2008	4/16/2008	20	20	0	1	1	1	1	2	2	2
VU#218395	CVE-2008-1722	4/18/2008	4/15/2008	4/15/2008	3	0	0	1	1	1	1	0	0	1
VU#596268	CVE-2008-2005	4/28/2008	1/30/2008	5/5/2008	89	96	0	0	1	0	0	0	0	1
VU#140129	CVE-2007-5663	5/6/2008	10/3/2007	2/9/2008	216	129	0	1	0	0	1	2	2	2
VU#666281	CVE-2007-5659	5/6/2008	2/9/2008	2/9/2008	87	0	0	1	0	0	1	2	2	2
VU#936529	CVE-2007-6026	5/13/2008	11/16/2007	11/16/2007	179	0	0	0	1	0	0	2	2	2
VU#315107	CVE-2007-6078	5/19/2008	11/21/2007	11/21/2007	180	0	0	1	0	1	0	1	1	1
VU#929656	CVE-2007-6372	5/19/2008	12/12/2007	12/12/2007	159	0	1	1	1	0	1	0	0	2
VU#119747	CVE-2008-1104	5/20/2008	4/23/2008	5/20/2008	27	27	1	1	0	0	0	2	2	2
VU#110947	CVE-2008-0177	5/28/2008	11/30/2007	2/6/2008	180	68	1	0	1	0	1	0	0	2
VU#190939	CVE-2007-5604	6/3/2008	4/11/2008	6/4/2008	53	54	0	0	0	0	0	1	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#878044	CVE-2008-0960	6/4/2008	5/20/2008	5/31/2008	15	11	0	1	1	0	1	2	2	2
VU#132419	CVE-2008-1585	6/9/2008	5/8/2008	6/9/2008	32	32	1	1	0	0	0	1	1	1
VU#878044	CVE-2008-0960	6/9/2008	5/16/2008	5/31/2008	24	15	0	1	1	1	1	2	2	2
VU#878044	CVE-2008-0960	6/9/2008	5/20/2008	5/31/2008	20	11	1	0	1	0	1	2	2	2
VU#878044	CVE-2008-0960	6/9/2008	5/31/2008	5/31/2008	9	0	1	1	1	1	1	2	2	2
VU#878044	CVE-2008-0960	6/10/2008	5/20/2008	5/31/2008	21	11	1	0	1	1	1	2	2	2
VU#878044	CVE-2008-0960	6/11/2008	5/31/2008	5/31/2008	11	0	0	1	1	1	1	2	2	2
VU#476345	CVE-2008-2639	6/12/2008	4/14/2008	6/11/2008	59	58	1	0	0	0	0	2	2	2
VU#127185	CVE-2008-2306	6/19/2008	6/9/2008	6/19/2008	10	10	0	0	1	0	0	2	2	2
VU#800113	CVE-2008-1447	6/27/2008	4/21/2008	7/8/2008	67	78	0	0	1	0	1	0	1	1
VU#166651	CVE-2008-4385	7/3/2008	4/21/2008	10/14/2008	73	176	0	1	0	0	0	2	2	2
VU#800113	CVE-2008-1447	7/3/2008	6/3/2008	7/8/2008	30	35	0	0	1	0	1	0	1	1
VU#889747	CVE-2008-0951	7/8/2008	2/19/2008	3/20/2008	140	30	0	0	1	0	0	2	2	2
VU#800113	CVE-2008-1447	7/8/2008	4/14/2008	7/8/2008	85	85	1	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/8/2008	4/21/2008	7/8/2008	78	78	0	0	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/8/2008	4/29/2008	7/8/2008	70	70	0	1	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/8/2008	5/1/2008	7/8/2008	68	68	1	1	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/8/2008	5/5/2008	7/8/2008	64	64	1	1	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/8/2008	5/5/2008	7/8/2008	64	64	1	0	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/8/2008	5/5/2008	7/8/2008	64	64	1	0	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/8/2008	5/5/2008	7/8/2008	64	64	1	1	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/9/2008	4/21/2008	7/8/2008	79	78	0	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/9/2008	5/5/2008	7/8/2008	65	64	1	0	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/9/2008	5/5/2008	7/8/2008	65	64	1	0	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/9/2008	5/5/2008	7/8/2008	65	64	0	0	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/9/2008	7/3/2008	7/8/2008	6	5	0	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/10/2008	4/21/2008	7/8/2008	80	78	0	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/10/2008	4/21/2008	7/8/2008	80	78	0	1	1	0	1	0	1	1

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#800113	CVE-2008-1447	7/11/2008	4/21/2008	7/8/2008	81	78	1	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/11/2008	5/5/2008	7/8/2008	67	64	1	0	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/11/2008	5/5/2008	7/8/2008	67	64	1	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/11/2008	6/6/2008	7/8/2008	35	32	1	1	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/11/2008	7/8/2008	7/8/2008	3	0	0	1	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/13/2008	5/5/2008	7/8/2008	69	64	0	1	1	1	1	0	1	1
VU#800113	CVE-2008-1447	7/14/2008	7/8/2008	7/8/2008	6	0	1	1	1	0	1	0	1	1
VU#130923	CVE-2008-2933	7/15/2008	6/22/2008	7/16/2008	23	24	1	1	1	1	1	1	0	0
VU#277313	CVE-2008-4387	7/16/2008	7/7/2007	11/7/2008	375	489	0	0	0	0	0	2	2	2
VU#800113	CVE-2008-1447	7/16/2008	4/21/2008	7/8/2008	86	78	1	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/17/2008	7/8/2008	7/8/2008	9	0	0	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/21/2008	5/5/2008	7/8/2008	77	64	0	1	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/21/2008	5/5/2008	7/8/2008	77	64	0	1	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/24/2008	7/8/2008	7/8/2008	16	0	1	1	1	0	1	0	1	1
VU#831457	CVE-2008-1309	7/25/2008	3/10/2008	3/10/2008	137	0	0	1	0	1	0	2	2	2
VU#800113	CVE-2008-1447	7/28/2008	7/8/2008	7/8/2008	20	0	0	0	1	0	1	0	1	1
VU#800113	CVE-2008-1447	7/30/2008	4/21/2008	7/8/2008	100	78	0	1	1	0	1	0	1	1
VU#878044	CVE-2008-0960	7/30/2008	5/20/2008	5/31/2008	71	11	0	0	1	0	1	2	2	2
VU#800113	CVE-2008-1447	7/31/2008	5/5/2008	7/8/2008	87	64	1	0	1	0	1	0	1	1
VU#716387	CVE-2008-3257	8/4/2008	7/21/2008	7/21/2008	14	0	0	0	1	0	0	2	2	2
VU#800113	CVE-2008-1447	8/5/2008	4/21/2008	7/8/2008	106	78	0	0	0	0	1	0	1	1
VU#800113	CVE-2008-1447	8/8/2008	5/5/2008	7/8/2008	95	64	0	0	1	0	1	0	1	1
VU#309739	CVE-2008-2245	8/12/2008	4/10/2008	8/12/2008	124	124	0	0	1	0	0	2	2	2
VU#837785	CVE-2008-2463	8/12/2008	7/1/2008	7/7/2008	42	6	1	0	0	0	0	1	1	1
VU#661827	CVE-2008-3558	8/14/2008	6/20/2008	8/7/2008	55	48	1	1	0	1	0	2	2	2
VU#298651	CVE-2007-5400	8/14/2008	7/25/2008	7/25/2008	20	0	0	1	0	1	1	2	2	2
VU#938323	CVE-2008-2936	8/14/2008	8/1/2008	8/18/2008	13	17	0	1	1	1	1	2	2	2
VU#938323	CVE-2008-2936	8/14/2008	8/1/2008	8/18/2008	13	17	0	0	1	1	1	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#938323	CVE-2008-2936	8/15/2008	8/1/2008	8/18/2008	14	17	0	0	1	1	1	2	2	2
VU#985449	CVE-2007-4475	8/19/2008	7/21/2008	3/31/2009	29	253	0	0	0	0	1	2	2	2
VU#817940	CVE-2008-2464	9/4/2008	8/22/2008	8/22/2008	13	0	0	0	1	1	1	0	0	2
VU#146896	CVE-2008-3636	9/9/2008	3/13/2008	10/7/2008	180	208	0	1	0	0	1	2	2	2
VU#538011	CVE-2008-2468	9/12/2008	9/3/2008	9/12/2008	9	9	0	0	1	0	0	2	2	2
VU#126787	CVE-2008-3618	9/15/2008	12/13/2007	9/15/2008	277	277	0	0	1	0	0	2	2	2
VU#889484	CVE-2008-3964	9/18/2008	9/5/2008	9/5/2008	13	0	0	1	1	1	1	0	0	1
VU#800113	CVE-2008-1447	9/26/2008	4/21/2008	7/8/2008	158	78	0	1	1	0	1	0	1	1
VU#837092	CVE-2008-1093	9/26/2008	9/16/2008	9/16/2008	10	0	0	1	0	0	1	2	2	2
VU#472363	CVE-2008-4404	10/1/2008	7/30/2008	10/2/2008	63	64	0	1	1	1	1	2	2	2
VU#472363	CVE-2008-4404	10/2/2008	7/30/2008	10/2/2008	64	64	0	0	1	1	1	2	2	2
VU#472363	CVE-2008-4404	10/2/2008	7/30/2008	10/2/2008	64	64	0	0	1	0	1	2	2	2
VU#472363	CVE-2008-4404	10/2/2008	7/30/2008	10/2/2008	64	64	0	1	1	0	1	2	2	2
VU#146896	CVE-2008-3636	10/7/2008	3/13/2008	10/7/2008	208	208	0	0	0	0	1	2	2	2
VU#343971	CVE-2008-2474	10/8/2008	9/25/2008	9/25/2008	13	0	0	0	1	0	0	2	2	2
VU#406937	CVE-2008-0957	10/14/2008	8/16/2006	5/19/2008	790	642	0	0	0	0	0	1	1	1
VU#793233	CVE-2008-1446	10/14/2008	5/2/2008	10/14/2008	165	165	1	0	1	0	0	2	2	2
VU#146896	CVE-2008-3636	10/15/2008	3/13/2008	10/7/2008	216	208	0	1	0	0	1	2	2	2
VU#183657	CVE-2008-2469	10/23/2008	9/18/2008	10/21/2008	35	33	0	1	1	0	1	2	2	2
VU#472363	CVE-2008-4404	10/27/2008	7/30/2008	10/2/2008	89	64	0	0	1	1	1	2	2	2
VU#837092	CVE-2008-1093	10/29/2008	9/16/2008	9/16/2008	43	0	0	0	0	0	1	2	2	2
VU#472363	CVE-2008-4404	10/31/2008	7/30/2008	10/2/2008	93	64	0	1	1	1	1	2	2	2
VU#593409	CVE-2008-2992	11/4/2008	3/21/2008	11/4/2008	228	228	0	1	0	0	0	2	2	2
VU#524681	CVE-2007-0328	11/27/2008	5/31/2007	5/31/2007	546	0	0	1	0	0	0	2	2	2
VU#493881	CVE-2008-4844	12/17/2008	12/9/2008	12/9/2008	8	0	0	0	1	0	1	2	2	2
VU#541025	CVE-2008-2434	12/18/2008	8/18/2008	12/21/2008	122	125	0	1	0	0	0	2	2	2
VU#702628	CVE-2008-2435	12/18/2008	8/25/2008	12/21/2008	115	118	0	1	0	0	0	2	2	2
VU#768681	CVE-2006-5268	1/12/2009	8/8/2008	11/11/2008	157	95	1	0	0	0	0	2	2	2

TABLE E1: Cont'd

CERT Name	NVD Name	Patch Date	Notification Date	Public Date	Patch Time	Disclosure Time	Multiple Patches	Patch Type	SW Type	Vendor Type	Multiple Vendor	C	I	A
VU#194505	CVE-2008-4388	1/15/2009	10/17/2008	1/15/2009	90	90	0	1	0	0	0	2	2	2
VU#131100	CVE-2009-0305	2/10/2009	8/20/2008	2/10/2009	174	174	0	1	0	0	0	2	2	2
VU#696644	CVE-2008-5416	2/10/2009	12/9/2008	12/9/2008	63	0	1	0	1	0	0	2	2	2
VU#461321	CVE-2009-0208	2/21/2009	10/1/2008	2/24/2009	143	146	0	0	0	0	0	2	2	2
VU#472363	CVE-2008-4404	3/5/2009	7/30/2008	10/2/2008	218	64	0	1	1	0	1	2	2	2
VU#905281	CVE-2009-0658	3/10/2009	2/19/2009	2/19/2009	19	0	1	0	0	0	1	2	2	2
VU#276563	CVE-2008-4564	3/17/2009	1/14/2009	3/17/2009	62	62	0	0	0	0	1	2	2	2
VU#196617	CVE-2009-0799	3/19/2009	2/23/2009	4/16/2009	24	52	0	0	0	1	1	0	0	1
VU#878044	CVE-2008-0960	3/23/2009	5/20/2008	5/31/2008	307	11	0	1	0	0	1	2	2	2
VU#926676	CVE-2008-4841	4/14/2009	12/9/2008	12/9/2008	126	0	1	0	0	0	0	2	2	2
VU#196617	CVE-2009-0799	4/16/2009	3/12/2009	4/16/2009	35	35	1	1	1	1	1	0	0	1
VU#196617	CVE-2009-0799	4/16/2009	3/12/2009	4/16/2009	35	35	1	0	1	1	1	0	0	1
VU#196617	CVE-2009-0799	4/16/2009	3/31/2009	4/16/2009	16	16	0	0	1	0	1	0	0	1
VU#196617	CVE-2009-0799	4/28/2009	4/6/2009	4/16/2009	22	10	1	0	1	1	1	0	0	1
VU#196617	CVE-2009-0799	5/5/2009	4/6/2009	4/16/2009	29	10	1	1	1	1	1	0	0	1
VU#238019	CVE-2009-0688	5/12/2009	4/8/2009	4/8/2009	34	0	0	0	1	1	1	1	1	1
VU#970180	CVE-2009-1492	5/12/2009	4/28/2009	4/28/2009	14	0	1	0	0	0	1	2	2	2
VU#853097	CVE-2009-1252	5/18/2009	5/6/2009	5/18/2009	12	12	1	0	1	1	1	1	1	1
VU#853097	CVE-2009-1252	5/19/2009	5/6/2009	5/18/2009	13	12	1	1	1	1	1	1	1	1
VU#853097	CVE-2009-1252	5/26/2009	5/7/2009	5/18/2009	19	11	0	1	1	1	1	1	1	1
VU#196617	CVE-2009-0799	6/8/2009	2/23/2009	4/16/2009	105	52	1	1	1	0	1	0	0	1
VU#787932	CVE-2009-1535	6/9/2009	3/12/2009	3/12/2009	89	0	0	0	0	0	0	2	2	2
VU#853097	CVE-2009-1252	6/9/2009	5/6/2009	5/18/2009	34	12	0	0	1	1	1	1	1	1
VU#568153	CVE-2009-1861	6/9/2009	5/8/2009	6/9/2009	32	32	1	0	0	0	1	2	2	2
VU#251793	CVE-2009-0690	6/19/2009	6/2/2009	6/19/2009	17	17	0	1	0	0	0	2	2	2