# Exploitation and Exploration Networks in Open Source Software Development: An Artifact-Level Analysis

ORCUN TEMIZKAN AND RAM L. KUMAR

ORCUN TEMIZKAN received his Ph.D. in computing and information systems with a concentration in business information systems and operations management at the Belk College of Business, University of North Carolina at Charlotte. His research interests include open innovation, social networks, information security, organization and team structures, and business analytics. His research has appeared in leading journals and conference proceedings, such as *Journal of Management Information Systems, Institute for Operations Research and the Management Sciences (INFORMS),* and *Workshop on e-Business (WeB)*.

RAM L. KUMAR is a professor in the Belk College of Business, UNC-Charlotte. He received his Ph.D. from the University of Maryland, where he was the recipient of the Frank T. Paine Award for Academic merit. He worked for major multinational corporations, such as Fujitsu, before entering academia. His research has been funded by organizations such as the U.S. Department of Commerce and firms in the financial services and energy industries. His current research interests include open innovation, techniques for evaluating and managing portfolios of information technology investments, service science, knowledge management, and business analytics. His research has been published in *Communications of the ACM, Computers and Operations Research, Decision Sciences, Decision Support Systems, European Journal of Information Systems, IEEE Transactions on Engineering Management, International Journal of Electronic Commerce, Journal of Management Information Systems,* and other journals. He has advised organizations such as the U.S. Department of Energy and private sector firms on the evaluation of research and development projects, risk management, and project portfolio management.

ABSTRACT: Open source software (OSS) development is an increasingly important paradigm of software development. However, key aspects of OSS such as the determinants of project success and motivations of developers in joining these projects are not well understood. Based on organizational theory, we propose that OSS activities of patch development and feature request can be classified as exploitation (implementation-oriented) and exploration (innovation-

We are grateful to the anonymous reviewers and the editor in chief for their insightful comments. This research was funded in part by a grant from the Belk College of Business, University of North Carolina-Charlotte.

*Journal of Management Information Systems* / 2015, Vol. 32, No. 1, pp. 116–150.

ISSN 0742–1222 (print) / ISSN 1557–928X (online)
DOI: 10.1080/07421222.2015.1029382

oriented) activities, respectively. We empirically examine how the structure of social network affects the success of patch-development and feature-request networks in OSS projects, using a data set collected from the SourceForge database. Our results provide empirical support for the view that patch development and feature request are exploitation and exploration activities, respectively. Network structures differ due to team formation differences and have a differential impact on development success based on the type of activity. The concepts of ambidextrous developers and ambidexterity are explored in the context of OSS projects. Collectively, our results indicate that studying OSS projects at the artifact level could improve our understanding of OSS project success and team formation. This, in turn, could lead to better management of OSS projects.

KEY WORDS AND PHRASES: exploitation and exploration, open source software development, project success, social networks, software development, team formation.

---

Open source software (OSS) development is a community-based model of software development that depends on the volunteer contribution of software developers. Network social capital is defined as the benefits OSS developers secure from their membership in social networks of OSS projects [66]. The social network structure of software developers reflects network social capital for OSS projects. Prior research has studied the social network structure of software developers as a determinant of project success [30, 65, 66]. However, prior research has focused on projects as the unit of analysis. Projects indeed consist of different teams that produce different types of artifacts such as patches and feature requests. OSS development at the artifact level is underresearched. Understanding factors such as network structures of artifact teams in the context of OSS development is an interesting area of research. This research aims to produce a deeper and more nuanced understanding of OSS projects by answering the following research questions: Does the network structure of artifact teams have a differential impact on OSS artifact development performance depending on the type of artifact? Does heterogeneity exist in the network structure of developers depending on the type of artifact?

Software products are improved by correcting faults, or enhanced by adding new features based on user requirements [5, 6]. The total cost of software maintenance is estimated to make up at least 50 percent of total software life-cycle costs [73]. Thus, the modification of software after delivery is one of the major phases of software development. Software maintenance includes two important types of OSS project activities: patch development and feature request. Patch-development activities are defined as activities to correct faults in software (SourceForge.net). Feature-request activities are defined as software-enhancement activities to add new features based on new user requirements (SourceForge.net). While prior OSS research has focused on project-level analysis [30, 33, 65, 66], recent OSS research has studied feature-request activities [74].

Prior research indicates that communication patterns differ based on the characteristics of a task [41]. A task (activity) can differ along several dimensions, including the generation of new knowledge vs. using existing knowledge, specific vs. general problem orientation, and time span [41]. Recent research has studied requirements determination tasks such as those represented by feature requests in an OSS development context [74]. Such tasks represent requests for new software features, can be complex to trace in terms of time span, and are stated in general terms. They are focused toward radical improvement by enhancing software. Sophisticated natural language processing techniques have to be used to delimit tasks and understand their scopes [74]. Patch requests, on the other hand, represent tasks to improve software by correcting faults, tend to be less complex, and are focused toward specific problems. They are focused toward incremental improvement by correcting faults. Thus, feature request and patch development are different along multiple task dimensions.

In an organizational context, March [50] has contrasted exploitation and exploration as two types of activities for the development and use of knowledge in organizations. Exploration represents activities that change organizational competencies and build on a different technological trajectory. These activities require new knowledge, offer new designs, and create new products and services. They tend to be somewhat complex, broadly focused, and can involve significant amounts of time. Thus, exploration activities tend to have task dimensions that resemble feature requests. Hence, we suggest that feature-request activities in OSS development can be considered exploration activities.

In contrast, 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. Such activities are likely to be relatively simple, narrowly focused, and somewhat short in duration, particularly when compared to exploration activities. Thus, exploitation activities tend to have task dimensions similar to patch-development activities. Hence, we suggest that patch-development activities in OSS development can be considered exploitation activities. In addition, exploration is associated with flexibility, decentralization, and loose cultures, whereas exploitation is related to efficiency, centralization, and tight cultures [7].

Prior research on OSS development 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. Different network structures may also make related activities more successful, and thus network structures have differential effects on the success of different types of subnetworks.

In this study, we test our hypotheses using data sets collected from the SourceForge database. Our results illustrate empirical support for the view that patch development and feature request are exploitation and exploration activities,

respectively. Network structures influence the success of teams at the activity level but have a differential impact based on the type of activity. Our results also illustrate that significant heterogeneity exists between the network structures of patch-development and feature-request networks. Collectively, these results provide a deeper and more nuanced understanding of OSS development compared to the results of the extant research. These results represent important contributions to multiple streams of OSS research including OSS development success and OSS team formation, and open up new avenues for future research.

## Literature Review

### Open Source Software Development Activities

Software development involves knowledge work and relies on specialized skills and expertise that a developer brings to the project [21, 23, 59]. 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 [57]. In contrast, OSS vendors depend mainly on the voluntary contributions of software developers and, hence, OSS products are developed in a collective manner beyond the boundaries of a single organization [57].

Given the importance of voluntary contributions of software developers for OSS development, the impact of the network structure of OSS developer networks [30, 65, 66] and the formation of OSS developer teams [33] have been intensively studied. Recent studies have shown that the network structure of OSS developers significantly affects OSS project success [30, 65, 66]. However, these studies have not examined different types of activities within projects. 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 [5, 6]. Our focus is on different types of software maintenance activities.

OSS projects may vary in terms of the degree of control and coordination based on a combination of governance mechanisms that are effectively used together [20]. The combination of control modes found in OSS projects is different from those found in other types of software development (such as propriety software development). It is important to realize that this can be due to unique features of OSS projects such as volunteer participation and a controller who is part of the development team [20].

### Open Source Software Collaboration Network

In the social network literature, an affiliation network is a special kind of network that depends on the affiliation between two groups [75]. 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 in events. Therefore, actors are related to each other through their joint affiliation with or their comembership in events. Events are also related to each other through common actor(s).

OSS software development is a community-based model that involves collaboration among software developers. OSS developers may work on multiple activities concurrently. An activity starts when a developer starts a new activity under a project. Other developers may join and start participating in an activity. An activity is performed by developers who joined that activity. OSS developers can belong to multiple activities. A comembership 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). These kinds of relationships between developers and activities can be represented by an affiliation network. In OSS networks, actors are developers, and events are activities such as patch-development and feature-request activities. This research seeks to better understand the social structure of these relationships.

## Exploitation and Exploration Activities

The exploitation and exploration framework distinguishes two general approaches to organizational learning involving the development and use of knowledge in organizations: the exploitation of old certainties and the exploration of new possibilities [50]. Exploitation is the refinement and extension of existing competencies, technologies, and paradigms, and includes things such as refinement, choice, production, efficiency, selection, implementation, and execution [50]. It refers to incremental innovations that involve improvements in existing components and build on the existing technological trajectory [7]. 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 [46]. In contrast, exploration is experimentation with new alternatives, and captured by terms such as search, variation, risk taking, experimentation, play, flexibility, discovery, and innovation [50]. It refers to radical innovations that involve a shift to a different technological trajectory and changes organizational competencies [7]. It offers new designs and creates new markets. Thus, exploration requires new knowledge or departures from existing knowledge [46].

For March [50], exploitation and exploration represent 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 because exploitation results in the convergence of ideas by eliminating the differences [50]. In contrast, 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 [50]. Moreover, exploitation is related to efficiency, centralization, and tight cultures whereas exploration is associated with

flexibility, decentralization, and loose cultures [7]. Therefore, exploitation and exploration require different organizational structures [7, 46]. In the context of information technology (IT) outsourcing, organizations may need different governance structures to address different goals in IT service management [15]. For example, contractual governance helps to improve efficiency in an outsourcing relationship, whereas relational governance facilitates the satisfaction of changing business needs. Different organizational structures for exploitation and exploration enable exploitative teams to develop the best viable solutions and enable exploratory teams to explore new ideas [22]. Recent studies have found that organizational units pursuing exploration are smaller, more decentralized, and more flexible than those responsible for exploitation [7].

## 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 [37]. Therefore, interactions among team members are necessary activities to transform team members' knowledge into project success [37]. However, the nature of OSS development characterized by volunteer contributions of software developers poses challenges in coordination among developers [21, 59].

The formation and evolution of team cognition have been analyzed in software project teams [37]. 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 [37]. Team cognition helps software project teams effectively manage their members' knowledge, expertise, and skills as integrated assets [21, 37]. The development of shared understanding among team members also enables them to reach an agreement on decisions in the face of conflicting ideas [40]. The development of shared understanding promotes commitment to project goals, which, in turn, leads to the better completion of project tasks. The development of shared understanding is critical for effective collaboration among team members [8]. Hahn, Moon, and Zhang [33] studied the impact of prior collaboration ties on OSS 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.

In the 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 [18]. Through social ties, an actor can 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 [32]. Recent studies have also indicated that the position of a team in a network affects team outcomes [4, 30, 35, 39, 48, 58, 60, 61, 63, 67].

The social network literature contains two contradictory perspectives about the form of network structures: the internal focus or social closure perspective [18] and

the external focus or structural holes perspective [14]. From Coleman's [18] 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 fosters group identification [58] and a level of mutual trust, which facilitates knowledge exchange and collective action [18]. Social closure enables the convergence of individual interests to pursue common initiatives and to facilitate mutual coordination [58]. From Burt's [14] structural holes perspective, constructing networks that consist of disconnected alters[1] is the optimal strategy. The structural holes perspective focuses on value derived from bridging gaps (i.e., structural holes) between nodes in a social network [14]. This boundary-spanning structure generates information benefits because information tends to be relatively redundant within a given group [14]. 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 a single group [29]. Although prior research on social network analysis has indicated the trade-off between two contradictory perspectives, these two perspectives do not conflict with one another [58]. 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 [1] studied the impact of social network structures on innovation in terms of structural holes. Consistent with Burt's [14] structural holes perspective, Ahuja [1] indicates that external connections can provide access to knowledge spillovers and serve as information conduits through which news of technical breakthroughs, new insights into problems, or failed approaches travels from one firm to another. Ahuja [1] found that structural holes provide access to novel information.

Zaheer and Bell [77] 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. They found that firms bridging structural holes are more innovative and perform better than other firms that are not so positioned. The underlying mechanism posited by Burt [14] is that actors in a network rich in structural holes will be able to access novel information from remote parts of the network, and exploit that information to their advantage. Zaheer and Bell [77] also indicated that internal connectiveness enables firms to further exploit the ideas obtained from external resources.

Jansen, Van Den Bosch, and Volberda [39] focused on the differences between exploitation and exploration, and examined the impact of internal cohesion and centralization on exploitation and exploration in organizational contexts. They found that internal connectedness within teams positively affects the performance of exploitation and exploration teams, whereas centralization negatively affects exploration teams. However, Balkundi and Harrison [4] indicated that teams that are central in their intergroup network tend to perform better.

In the social network literature, divisions of actors into subgroups can be a very important aspect of social structure to understand how the network as a whole is likely to behave [34]. Nooy, Mrvar, and Batagelj [55] indicated that people are joined by more than interaction. Social interaction is the basis for shared norms, identity, and collective behavior [55]. Therefore, people who interact intensively are likely to consider themselves as a social group. Social relations between people tie them into cohesive subgroups that have their own norms, values, orientations, and subcultures [64]. More interesting subgroups are components that divide the network into separate parts based on connectedness [34]. Component analysis is used to identify the extent and nature of the pattern of connectedness in the network [55]. Breschia and Catalini [12] used a component analysis to compare the macro-level properties of scientific and technological research networks. Connectedness can be an indicator for the behavior of the network as a whole [34]. Therefore, connectedness may be consequential for predicting both the opportunities and constraints facing groups and actors as well as for predicting the evolution of the network itself [34].

Social network analysis [75] 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. Consistent with previous studies on social network research [1, 14, 18, 25, 29, 71, 75, 76], organizational research [35, 36, 58, 63], and OSS development research [30, 33, 65, 66], we categorized our social network variables into four categories: internal cohesion, external connectivity, network location, and network decomposition. Social network analysis also provides multiple levels of analyses. Micro-level analysis is concerned with how software developers are connected, whereas macro-level analysis is concerned with how individuals are embedded within a structure and how the structure emerges from the micro-level interactions between individuals [34]. Thus, we further categorized our social network variables into two levels. Micro-level analyses are performed using internal cohesion, external connectivity, and network location variables. Macro-level analyses are performed using a network decomposition variable.

## Theoretical Background and Hypotheses

Based on concepts underlying multiple-level analyses of social networks, we analyze how the network structure of artifact teams affects OSS artifact development performance depending on the type of artifact. We develop hypotheses for internal cohesion, external connectivity, and network location at a micro level, and for network decomposition at a macro level. We also develop hypotheses to illustrate the existence of heterogeneity in network structure depending on the type of artifact.

### Internal Cohesion

The OSS development process is characterized by the lack of a relatively strong control of design and implementation [57] and the lack of face-to-face communication

[66]. Therefore, OSS teams require a constructive environment to foster trust, reciprocity norms, and shared identity, which, in turn, are preconditions for successful resource sharing and collaboration among developers [18, 45].

Internal cohesion increases the information transmission capacity of a team [63]. First, internal cohesion improves access to information because the same information is available via multiple paths [63]. 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 [63]. Second, internal cohesion makes information exchange meaningful and useful [63]. 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 from team members [21, 37]. Shared knowledge improves coordination among team members because it enables the team to develop more accurate explanations and expectations about tasks and other team members [21] because prior interactions enable developers to acquire information about the skills and capabilities of other developers [28] and about who knows what [23]. In addition, shared knowledge of problems and solutions enhances further learning [63]. Third, internal cohesion can make developers more willing and able to improve information exchange and collaboration among team members by fostering trust, reciprocity norms, and shared identity [1, 18, 36, 45]. Enhanced trust, reciprocity norms, and shared identity results in a high level of collaboration by providing self-enforcing informal governance mechanisms [63]. Fourth, internal cohesion fosters group identification, which enables the convergence of individual interests to pursuit common initiatives and to facilitate mutual coordination [58]. Fifth, internal cohesion also helps developers to develop team cognition, which promotes team coordination [21, 37]. Team cognition helps developer teams to effectively manage team members' knowledge, expertise, and skills as integrated assets [21, 37]. As a result, internal cohesion results in a high level of collaboration among team members. By improving the information transmission capacity of a team, it also enables the more rapid exchange and integration of greater amounts of information and knowledge. Internal cohesion allows individuals to develop a deep understanding to further refine and improve existing products and processes [62]. Thus, we argue that the success of artifact teams will be positively related to internal cohesion.

Internal cohesion diffuses strong norms and establishes shared expectations [62, 64]. Therefore, it reduces deviant behavior, limits search scope, and increases the selective perception of alternatives. Internal cohesion may result in the homogenization of information within a team [14, 29] and the convergence of knowledge and ideas [46]. Therefore, internal cohesion may limit access to alternative ways of thinking and novel information [54]. Cohesive team members start thinking similarly and have a tendency to overlook alternate ways of solving problems [13]. Patch-

development teams build on existing knowledge and reinforce existing skills, processes, and structures [7, 46, 47]. In contrast, feature-request teams build on diverse knowledge that resides outside of the team [61] and require new knowledge [46]. Hence, we argue that internal cohesion has a differential impact on the success of patch-development and feature-request teams moderated by the artifact type. This leads us to the following hypotheses:

> ***Hypothesis 1a:*** *The success of patch-development and feature-request teams will be positively related to internal cohesion.*
>
> ***Hypothesis1b:*** *Internal cohesion has a differential impact on the success of patch-development and feature-request teams.*

## External Connectivity

Although the internal cohesion of a team provides various benefits in terms of trust and information exchange, developers have access to external resources from their relationships to other developers outside of a focal team. The structure of external connections affects the diversity of external knowledge available to a focal team. Hence, we focus on the external network structure (the cohesion of external connections) that affects the ability of developers to acquire novel information from external contacts [1, 14, 77].

### External Cohesion

External cohesion is cohesion among the external contacts of a team [66]. 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 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 [14]. A structural hole separates developers on either side of the hole and creates brokerage opportunities for those developers to obtain information from disconnected developers [14].

External cohesion basically measures the extent to which the external contacts of a team are connected to each other. If the external contacts of a team are highly connected with each other (high external cohesion or low structural holes), a team has limited access to novel information because too much cohesion results in homogenization of information, and the external contacts of a team may have relatively redundant information [13, 14, 29]. In contrast, if the external contacts of a team are not connected with each other (low external cohesion or high structural holes), a team has access to novel information from remote parts of the network such as other disconnected groups [14]. Therefore, the level of cohesion among the external contacts of a team determines the diversity of knowledge acquired from external contacts.

The OSS network is made up of distinct developer teams in which developers are highly connected with each other within each team, but weakly connected to other developers across other teams [65]. Teams tend to be heterogeneous across a network in terms of the knowledge they possess and produce because each team started with different initial conditions [22]. Therefore, external resources provide new knowledge, ideas, and insights [60].

Knowledge is developed through combinations of existing and new knowledge [42]. The process of sharing ideas with other teams that have novel information generates new knowledge, rather than merely exchanging existing information [54]. This idea is consistent with the idea put forth by March [50] that teams connected to other teams that have novel information may replicate innovative ideas and generate more new ideas that can be used to introduce new and innovative products. A team whose external contacts are not highly connected has access to new knowledge, ideas, and insights from disconnected external teams [13, 14] and is able to develop new knowledge through knowledge recombination [60]. Therefore, a team whose external contacts are not highly connected is able to develop new understandings, unlike a team whose external contacts are highly connected [77]. Combining diverse knowledge from other teams (different technology areas) also enhances the capacity for creative learning [24, 42, 58]. In order to acquire more novel information from external resources, the external contacts of a team should be diversified in terms of the knowledge they hold, and should therefore not be highly connected (low external cohesion). Thus, we argue that the success of artifact teams will be negatively related to external cohesion.

Feature-request teams build on diverse knowledge that resides outside of the team [61] and require new knowledge [46]. In contrast, patch-development teams build on existing knowledge and reinforce existing skills, processes, and structures [7, 46, 47]. Thus, we argue that external cohesion has a differential impact on the success of patch-development and feature-request teams moderated by the artifact type. These lead us to the following hypotheses:

> ***Hypothesis 2a:*** *The success of patch-development and feature-request teams will be negatively related to external cohesion.*
>
> ***Hypothesis 2b:*** *External cohesion has a differential impact on the success of patch-development and feature-request teams.*

## Network Location

Centrality is defined as the extent to which an actor occupies a central position in the network [75]. Developers who are more active in the network act as central actors in the network and are viewed as major channels of information in the network [66]. High centrality offers several benefits. First, central developers have access to unique knowledge, including an understanding of where such knowledge is located and how to obtain it [35]. Central developers also see a more complete picture of all

the alternatives available in the network than the peripheral developers [48]. With such information, centrality enables developers to make better decisions [4] by allowing them to discard irrelevant information [35]. Second, high centrality also allows developers to have quick access to greater amounts of knowledge in the network [71]. Third, high centrality allows the developer to control and regulate information flow among other developers [75], dispensing what is needed to other team members [4]. High centrality enhances a developer's ability to be central to the flow of information and resources in the network. Thus, we argue that the success of artifact teams will be positively related to centrality.

Central developers have a tendency toward overembeddedness in their network [48], which may result in access to redundant information [35] and incur the risk of learning myopia [46]. Extreme centrality through strengthening existing ties may be associated with too much domain-relevant knowledge and experience, which hinders the exploration of new ideas [48]. Centrality decreases the likelihood that team members seek innovative and new solutions [39]. Centrality also reduces the quality (novelty) and quantity of ideas [39]. Therefore, we argue that centrality is likely to hinder exploration. In contrast, centralized authority is beneficial to speeding up exploitation [39]. Exploitation depends mainly on the existing competence and processes, so it is limited in scope and newness [39]. Thus, we argue that centrality has a differential impact on the success of patch-development and feature-request teams moderated by the artifact type. These factors lead us to the following hypotheses:

***Hypothesis 3a:*** *The success of patch-development and feature-request teams will be positively related to centrality.*

***Hypothesis 3b:*** *Centrality has a differential impact on the success of patch-development and feature-request teams.*

## Network Decomposition

In the previous sections, we compared networks primarily from a micro perspective by examining network structures in which software developers are connected. In this section, we focus on the entire social network structure in which individual developers are embedded. Our aim is to investigate the structural properties of a whole network in order to discover naturally existing subgroups into which it can be divided and to explore their distinct pattern of network formation.

Collaboration improves information exchange by creating trust, shared norms, and identity within subgroups [18]. Cohesive subgroups have a high level of collaboration and information exchange among their members [63]. Therefore, cohesive subgroups can develop a deep understanding to further refine and improve existing products and processes [62]. However, cohesive subgroups share strong norms and establish shared expectations [71, 62]. Strong norms and shared expectations reduce deviant behavior, limit search scope, and increase a selective perception of

alternatives [71]. Therefore, cohesive subgroups are constrained in access to novel information insofar as high cohesion results in the homogenization of information [13, 29]. The patch-development network builds on existing knowledge and reinforces existing skills, processes, and structures [7, 46, 47], whereas the feature-request network builds on diverse knowledge [61] and requires new knowledge [46]. Thus, we argue that connectedness within subgroups has a differential impact on the success of patch-development and feature-request teams moderated by the artifact type. This leads us to the following hypotheses:

***Hypothesis 4a:*** *The success of patch-development and feature-request teams will be positively related to connectedness within subgroups.*

***Hypothesis 4b:*** *Connectedness within subgroups has a differential impact on the success of patch-development and feature-request teams.*

## Heterogeneity in Network Structures

Madey, Free, and Tynan [49] study the formation of OSS project teams from a social network perspective, and illustrate that this process has properties of self-organizing networks. It is important to realize that the overall structure of a network is the result of OSS participation decisions by individual developers. Their participation decisions, in turn, are driven by individual goals that include enjoyment, learning, obligation toward the OSS community, belief in the openness of code, a need for the software being developed, and a need for reputation [33]. The realization of these goals depends on perceptions of project success in terms of both outcome and process [33]. Since project success is uncertain, people who participate in self-organizing networks typically rely on social cues such as cohesion with other participants and the status of other participants [31]. These cues serve as surrogates for the likelihood of success, when the quality of collaboration cannot be directly measured. Prior research on OSS development indicates that developers use different cues depending on their perceived role within projects [33]. Since different artifacts in OSS projects are the result of different roles (exploitation for patch development and exploration for feature request) it is likely that developers in patch-development and feature-request networks attach varying degrees of importance to different social cues such as cohesion and status. The social networking literature also highlights the relationship between roles and ties. A large number of ties can deplete an individual's own resources because they are expensive to maintain [35] and more ties create larger role demands [4]. Thus, having many ties to others tends to constrain individual behavior within the role defined by those ties [4]. Different social network structures have been found to be appropriate for successful exploitation and exploration networks in organizational contexts [7, 22]. Hence, we propose that developers attach differing degrees of importance to cohesion cues and/or status cues depending on the type of activities. Over time, their decisions result in different network structures for patch-development and feature-request networks. Heterogeneity in

network structure can be measured using micro-level and macro-level social network variables. This leads us to the following hypotheses:

***Hypothesis 5a:*** *Heterogeneity exists in internal cohesion between patch-development and feature-request teams.*

***Hypothesis 5b:*** *Heterogeneity exists in external cohesion between patch-development and feature-request teams.*

***Hypothesis 5c:*** *Heterogeneity exists in centrality between patch-development and feature-request teams.*

***Hypothesis 5d:*** *Heterogeneity exists in connectedness within subgroups between patch-development and feature-request teams.*

## Data

### Data Sources and Collection

OSS network data required for this study have been collected from the SourceForge database (http://zerlot.cse.nd.edu). Although there are other OSS hosting websites such as GitHub and Google Code, the SourceForge database is a generally accepted source of OSS research data. Researchers analyzing issues related to OSS development have predominantly used SourceForge data [30, 33, 65, 66]. Consistent with these studies, this data has been supplemented with data from project repositories at the SourceForge website.

We used the whole network method in order to construct affiliation networks of developers engaged in patch-development and feature-request activities [34]. The data collection task is made more manageable by determining an appropriate boundary around the network, because the whole network method examines actors that are regarded as bounded social collectives [51, 66]. Prior studies on OSS development have used software development platforms called "project foundries" as a network boundary. Project foundries are built mainly on programming languages. Hence, project foundry and programming language are similar concepts. For example, Singh, Tan, and Mookerjee [66] used participation in Python foundry (which uses the Python programming language) and Grewal, Lilien, and Mallapragada [30] used participation in Perl foundry (which uses the Perl programming language) as a network boundary. However, foundry data associated with OSS projects were not available in the SourceForge database after 2005. 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 the UNIX/Linux operating system is dominant in the OSS community [68]. Second, developers who are familiar with the programming language are able to understand the source

code easily [68], thereby more efficient knowledge sharing may be possible within a project or across projects written in the same programming language. Third, 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 SourceForge.

Data collection started by identifying developer–activity pairs because OSS developers can work on multiple artifacts simultaneously if they are members of different artifact teams. 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 [75]. Affiliation data for activities and developers (associated with projects) were collected from the SourceForge database for projects registered from November 1999 (which is SourceForge's inception date) to December 2008 on the SourceForge website (http://SourceForge.net). We have set December 2008 as a cutoff date for our study. 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 five years, which provides sufficient variation in network characteristics.

In order to identify developer–activity pairs, we identified all projects that match the following criteria. First, we included projects that are written in C language (our network boundary). Second, we excluded projects that 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. Prior research has also indicated that a large proportion of projects hosted at the SourceForge database show no activity [65, 66]. These projects would be dead nodes in the network and the relationships involving them would not facilitate any knowledge transfers or spillovers [66]. Therefore, including such projects in the network could lead to misleading results. Following prior research [65, 66], 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 have shown no sign of activity from their inception until December 2008. For the projects that matched our criteria, we identified all patch-development and feature-request activities that have been successfully closed. We identified the type of activities by searching key words (i.e., patch/fix or feature/enhancement/change) on activity descriptions. Then, we identified the developers who joined either patch-development or feature-request activities. This allowed us to collect separate affiliation network data (developer–activity pairs) and construct separate affiliation networks for patch-development and feature-request activities. It is important to note that some developers belong to both patch-development and feature-request networks. We refer to these as ambidextrous developers. A patch-development 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).

We further analyzed our data sets regarding incomplete data associated with projects that may have moved out of SourceForge, or had come to SourceForge from other OSS hosting websites. SourceForge, GitHub, and Google Code are the top three OSS hosting websites in terms of number of projects and users [16, 19]. We identified all unique projects that are written in C language from GitHub and Google Code until a cutoff date for our study. Comparison shows no overlaps between projects in SourceForge and Google Code. However, we found three overlapping projects between SourceForge and GitHub. These projects possibly moved out of, but did not cancel their registrations in SourceForge. We have removed these projects from our data.

## Network Construction

The OSS network data analyzed in this study are 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. Network construction is based on Singh [66]. We constructed one patch-development network and one feature-request network consisting of unique developers in two steps.[2] In the first step, we constructed one affiliation network for patch-development activities and one affiliation network for feature-request activities based on developer–activity pairs. In the second step, we converted a two-mode network to a one-mode network by removing activities from an affiliation network and creating relationships between developers who work together on the same activity. Relationships among developers are undirected because a relationship between two developers is mutual. We constructed one patch network that includes 4,675 unique developers in 1,170 projects, and one feature-request network that includes 6,597 unique developers in 1,889 projects.

## Variable Definitions and Operationalization

The OSS network data analyzed in this study are 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 the following concerns. First, projects vary considerably in terms of the number of artifacts in the project as shown in Table 1. Second, artifacts under the same project have almost the same set of developers. Therefore, most of the observations are repeated at the artifact level. Therefore, the OSS network data analyzed in this study are aggregated data at the project level to eliminate repeated observations. In the following section, we describe the variables used in this study along with the construction of their measures. Constructs used to measure network structure are well-established in the social network literature [1, 14, 25, 75] and have been used in a variety of studies including OSS development [30, 33, 35, 36, 39, 45, 48, 58, 63, 65, 66, 77].

Table 1. Descriptive Statistics for Patch-Development and Feature-Request (FR) Networks

| | Patch network | FR network |
|---|---|---|
| Total number of projects | 1,170 | 1,889 |
| Project–Artifact Pair Statistics | | |
| Total Number of Artifacts | 23,051 | 31,122 |
| Avg. Number of Artifacts per Project | 19.70 | 16.48 |
| Std. dev. of Number Artifacts per Project | 123.33 | 52.20 |
| Min. Number of Artifacts per Project | 1 | 1 |
| Max. Number of Artifacts per Project | 2791 | 862 |
| Project–Developer Pair Statistics | | |
| Total Number of Developers | 4,675 | 6,597 |
| Number of Ambidextrous Developers | 3,078 | 3,078 |
| Number of Artifact Developers | 1,597 | 3,519 |
| Artifact–Developer Pair Statistics | | |
| Average Number of Developers per Artifact | 13.71 | 8.71 |
| Std. dev. of Number of Developers per Artifact | 12.49 | 10.69 |

## Dependent Variables

Software is a knowledge product [67] and the amount of knowledge created by a project measures the success of a project [66]. Extant research on software development has suggested the use of completion of modification requests (MRs) as a measure of the amount of knowledge creation by a project that follows an incremental software development approach [9, 30, 65, 66]. The MR measure, which is similar in concept to a work order, represents the addition of new functionality as well as the modification or repair of old functionality [9, 66]. In OSS development, the Concurrent Versions System (CVS) commit transactions measure a basic addition of functionality similar to that taken into account by the MR measure in a commercial development environment [52]. Therefore, the MR measure is equivalent to CVS commits [72]. Recent studies on OSS development have used the number of CVS commits as a measure of project success [30, 65, 66].

Artifact teams create one or more artifact files as outputs. These artifact files could be patch files that correct faults in a product or feature-request files that enhance a product by adding new features. These artifact files are committed to subsequent releases and represent changes in a product and are analogous to the concept of MR in an incremental software development process [9, 52]. Hence, an artifact file can be a CVS commit or a part of a CVS commit. However, the association of artifact files to CVS commits is not consistently available in CVS log files from CVS project repositories at SourceForge. As illustrated in Table 1, projects can vary considerably in terms of the number of artifacts in a project. Projects create different numbers of artifact files depending on the number of

artifacts in the project. The number of artifacts for each project can be used to control this effect, but a high correlation between the number of files and the number of artifacts results in inconsistent or unstable estimates. Hence, we normalized our dependent variable in terms of the number of artifacts and used the average number of files per artifact team (patch development or feature request) for a project as a measure of the success (the amount of knowledge creation) of an artifact team. We calculated the average number of files per artifact team for a project by averaging the total number of files created by patch-development or feature-request artifact teams of that project over the total number of patch-development or feature-request artifacts belonging to that project.

## Independent Variables

We categorized our social network variables[3] into two levels and four categories. Micro-level analysis includes internal cohesion, external connectivity, and network location variables. Macro-level analysis includes a network decomposition variable. These variables were measured using well-accepted constructs from the literature, as will be discussed later.

### Internal Cohesion

We measured internal cohesion for a project with the number of repeat ties. Repeated collaboration among project members developed through their joint participation in past teams captures the strength of interpersonal connections among team members [33, 66, 71]. We measured the number of repeat ties for a project by following Singh, Tan, and Mookerjee [66]. We counted the total number of projects on which each pair of project developers has 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.

### External Connectivity

We measured external connectivity for a project with external cohesion, which is operationalized with Burt's [14] 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 network constraint. We took an average of project developers' network constraints over all the project developers to calculate a measure of 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.

### Network Location

We measured network location for a project with Freeman's [25] degree centrality. Degree centrality is the measure of how an actor is connected to other actors in the network through direct connections [25, 75]. We calculated degree centrality for a project as follows. For each project developer, we calculated degree centrality. We took an average of project developers' degree centralities over all the project developers to calculate a measure of degree centrality for a project. Higher values (between 0 and 1) indicate that the project is made up of more connected developers.

### Network Decomposition

We analyze the macro-level structures of networks with a component analysis. Component analysis is used to identify the extent and nature of the pattern of connectedness in a network [55]. A component is a maximally connected subgroup in which all pairs of nodes are reachable, but between which any pairs of nodes in different components are disconnected [75]. Connectedness can be an indicator of opportunities and obstacles to communication or the transfer of resources in a network [64], and hence it can be an indicator for the behavior of the network as a whole [34].

We operationalized the macro-level structure of a network by calculating the average number of connections per component as follows. For each network, we counted the total number of connections within each component, divided by the number of developers within each component. We assigned this average number of connections per component to projects that belong to that component.

## Control Variables

Consistent with prior research, we included control variables in order to control effects of factors other than independent variables.[4] Control variables are categorized as software complexity constructed as the average artifact file size for a project [5, 6, 59, 69], market potential constructed as the number of page views for a project [30, 65, 66], a project life-cycle effect constructed as a project age [30, 66], and project characteristics constructed as project types with eighteen measures, intended audiences with seven measures, user interface with seven measures, and a project language as English [66]. We also controlled for the development status of software.

We included the artifact type to control the moderating effect of artifact type on social network variables. The artifact type has been constructed as a dummy variable. The measure of the artifact type takes a value 0 for patch-development activities, and 1 for feature-request activities. The interaction terms of the artifact type with social network variables are also included to capture the moderating effect of artifact type on social network variables as hypothesized in H1b–H4b.

## Research Methodology

Artifact success model and artifact network comparison analyses are performed by using micro-level network variables (repeat ties, external cohesion, and degree centrality) and a macro-level network variable (the average number of connections per component) to test our hypotheses.

## Artifact Success Model Analysis

We used ordinary least squares (OLS) regression to test our hypotheses about how social network variables affect the success of patch-development and feature-request networks. We define project success in terms of the amount of knowledge creation, and we use the average number of files as a dependent variable. We assume that artifact files produced by path-development and feature-request teams are identical measures for knowledge creation.[5] We tested our hypotheses by using a combined data set that included 3,059 observations (projects) consisting of 1,170 observations (projects) for patch-development activities and 1,889 observations (projects) for feature-request activities, as shown in Table 1. The artifact success model incorporates micro-level and macro-level network variables.

Preliminary investigation revealed that the dependent variable and some of the independent and control variables were not normally distributed. In such a case, the OLS regression might yield biased parameter estimates that cannot be easily interpreted [26]. As suggested by Gelman and Hill [26], we performed a logarithmic transformation on the dependent, and nonnormally distributed variables.

We examined the correlations among independent variables using Pearson correlation analysis [2] and the variance inflation factor (VIF) [53]. Pearson correlation analysis revealed that the interaction terms of independent variables with the artifact type were highly correlated and had a correlation coefficient exceeding the recommended cutoff point of 0.70 [2]. If not corrected, high multicollinearity can result in inflated standard errors and even inconsistent or unstable estimates [53]. As suggested by Myers [53], we mean-centered social network variables before calculating their interaction terms with the artifact type, which reduced the correlation between them to acceptable levels. We also tested the seriousness of correlations using the VIF [53]. The VIFs of all predictor variables are below 4.5, which does not indicate any multicollinearity problems.

We developed an artifact success model based on the general model specifications developed by prior research [30, 65, 66]. The general model specification is expressed as follows:

$$\begin{aligned} lnDV = f(&\beta_0 + \beta_1\ lnRT + \beta_2\ lnRT\ x\ AType + \beta_3\ EC + \beta_4\ EC\ x\ AType + \beta_5\ DC \\ &+ \beta_6\ DC\ x\ AType + \beta_7\ Con + \beta_8\ Con\ x\ AType + \beta_9\ AType + \beta_{10}\ DS \\ &+ \beta_{11}\ lnPage + \beta_{12}\ AAFS + \beta_{13}\ Age + \beta_{14}\ Eng + \sum\nolimits_{(m=1)}^{pt} \beta_{15m} PT_m \\ &+ \sum\nolimits_{(n=1)}^{ia} \beta_{16n} IA_n + \sum\nolimits_{(k=1)}^{ui} \beta_{17k} UI_k) \end{aligned}$$

where, *lnDV* is the logarithmic transformation of the dependent variable (the average number of files per artifact team) for a project; *lnRT* is the logarithmic transformation of the average number of repeat ties for a project; *EC* is the average external cohesion for a project; *DC* is the average degree centrality for a project; *Con* is the average number of connections per component to which a project belongs. *AType* is the artifact type; *DS* is the development status of a project; *lnPage* is the logarithmic transformation of the number of page viewsof a project; *AAFS* is the average artifact file size for a project; *Age* is the age of a project; and *Eng* is the project language (English). *PT* represents a set of dummy variables, one for each project type, which is denoted as *pt*. Note that *pt* is the total number of categories of the project type. *IA* represents a set of dummy variables, one for each intended audience of the project, which is denoted as *ia*. Note that *ia* is the total number of categories for the intended audience of a project. *UI* represents a set of dummy variables, one for each project user interface, which is denoted as *ui*. Note that *ui* is the total number of categories for the user interface of a project. We report the results of regression analyses in Table 2.

Table 2. Artifact Success Model Results (Dependent Variable: Average Number of Files, $N$ = 3,059)

| Variable name | Beta | Std. error |
|---|---|---|
| Artifact type | –1.025*** | 0.055 |
| Internal Cohesion | | |
| Repeat Ties | 1.474*** | 0.307 |
| Repeat Ties × Artifact Type | –0.623*** | 0.181 |
| External Cohesion | | |
| External Cohesion | –0.721*** | 0.229 |
| External Cohesion × Artifact Type | 0.270** | 0.135 |
| Network Location | | |
| Degree Centrality | 2.418*** | 0.501 |
| Degree Centrality × Artifact Type | –0.904*** | 0.334 |
| Network Decomposition | | |
| Avg. Number of Connections per Component | 0.834*** | 0.062 |
| Avg. Number of Connections per Component × Artifact Type | –0.361*** | 0.038 |
| Model statistics | | |
| *F*–statistic | 66.956*** | |
| Degree of freedom | 46 | |
| $R^2$ | 0.494 | |
| Adj. $R^2$ | 0.487 | |
| Std. error of the estimate | 0.822 | |

*Significant at the 10 percent level; **significant at the 5 percent level; ***significant at the 1 percent level.

## Micro-level Artifact Network Comparison

We examine network structures from a micro-level perspective based on how software developers are connected. We employed Bayesian analysis with a Markov Chain Monte Carlo (MCMC) to determine whether there is a statistically significant difference between patch-development and feature-request networks in terms of micro-level network variables [11, 44].

### Bayesian Analysis

Bayesian analysis is an inferential method used to test the difference of means [11, 44] of network variables for patch-development and feature-request networks. Bayesian statistics is preferred to traditional statistical methods such as the *t*-test for several reasons [44]. First, Bayesian analysis does not assume any specific distribution for two independent samples compared. Second, Bayesian analysis does not assume equal variance for two independent samples compared.

Bayesian analysis is a statistical method that specifies the relation between model parameters and observed data. It infers about a parameter by combining the prior distribution of a parameter and the information provided by observed data using a likelihood function. The prior distribution is the probability distribution of a parameter before the observed data are examined. The likelihood function is the likelihood of observing the data given a parameter value. The prior distribution and the likelihood function jointly define a posterior distribution of a parameter. The posterior distribution is the probability distribution of parameters obtained after examining and combining the information provided by observed data with the prior distribution of a parameter. Therefore, Bayesian analysis calculates the posterior distribution of parameters conditional on observed data. It uses the MCMC simulation method for sampling from posterior distributions since analytical calculations are often infeasible for realistic problems.

In the Bayesian analysis procedure, we used a list of 1,170 projects for patch-development activities and a list of 1,889 projects for feature-request activities as observed data. We report the descriptive statistics of micro-level network variables in Table 3. We defined the likelihood functions of network variables as being the normal distribution. We also used the noninformative prior distribution, which assigns the same probability to each possible value of the parameters. A noninformative prior distribution has a minimal impact on the posterior distribution [11, 44]. Five parallel Markov chains were generated independently for each artifact type. Each chain consisted of the total of 175,000 sample observations. In each chain, the first 50,000 observations were used in the burn-in process and the last 125,000 were used in the posterior process to calculate the conditional posterior distributions. The method suggested by Gelman and Rubin [27] was used to check whether a chain has converged to a stable equilibrium distribution. This statistic compares within-chain and between-chain variance for each parameter across multiple chains. Across five parallel chains, the diagnostic value for all parameters was around 1. This is lower

Table 3. Descriptive Statistics of Micro-Level and Macro-Level Variables ($N_{Patch}$ = 1,170, $N_{FR}$ = 1,889)

| Variable type | Variable name | Mean | Std. dev. |
|---|---|---|---|
| Internal Cohesion | Repeat Ties (Patch) | 0.7165 | 0.5043 |
| | Repeat Ties (FR) | 0.6270 | 0.5369 |
| External Connectivity | External Cohesion (Patch) | 0.5118 | 0.4137 |
| | External Cohesion (FR) | 0.4872 | 0.4401 |
| Network Location | Degree Centrality (Patch) | 0.1025 | 0.1612 |
| | Degree Centrality (FR) | 0.0537 | 0.0971 |
| Network Decomposition | Avg. Number of Connections per Component (Patch) | 2.7210 | 3.9406 |
| | Avg. Number of Connections per Component (FR) | 1.9471 | 3.3986 |

than 1.1, which is considered an acceptable limit by Gelman and Rubin [27]. This indicated that the convergence for all parameters was achieved. After convergence, we used a Gibbs sampler and the Metropolis–Hastings algorithm in the MCMC procedure to obtain the posterior sample observations using the posterior distribution of parameters. We thinned each chain by keeping every fifth of posterior sample observations in order to reduce possible autocorrelation, if not corrected, which may result in biased Monte Carlo standard errors [11, 44]. This procedure created 25,000 posterior sample observations in each chain. By combining all posterior sample observations created in five Markov chains, we created 125,000 sample observations for a patch-development network and 125,000 sample observations for a feature-request network. We calculated the posterior means of network variables for patch-development and feature-request networks. We used the mean differences to compare micro-level network variables of patch-development and feature-request activities and to test the relevant hypotheses. We report the mean differences between network variables in Table 4. We test the significance of our hypotheses (zero mean differences for network variables) using the percentage of the posterior distribution of differences in mean that is greater than zero for each network variable [11, 44].

## Macro-Level Artifact Network Comparison

We examined the entire network structures from a macro-level perspective based on how individual developers are embedded within a structure. We employed two statistical methods. First, we used the quadratic assignment procedure (QAP) to examine the degree of dissimilarity between patch-development and feature-request networks [38]. 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 additional validation and greater reliability of the findings of the Bayesian analysis performed in the previous section

Table 4. Bayesian Model Results of Micro-Level and Macro-Level Variables ($N_{Patch}$ = 125,000, $N_{FR}$ = 125,000)

| | | | | | Statistical analysis | |
|---|---|---|---|---|---|---|
| Variable type | Variable name | Mean difference | Std. dev. | MCSE | Result | Probability |
| Internal Connectivity | Repeat Ties (Patch–FR) | 0.0895 | 0.0193 | 0.000053 | Patch > FR | 100.00% |
| External Connectivity | External Cohesion (Patch–FR) | 0.0247 | 0.0158 | 0.000043 | Patch > FR | 95.13% |
| Network Location | Degree Centrality (Patch– FR) | 0.0488 | 0.0052 | 0.000014 | Patch > FR | 100.00% |
| Network Decomposition | Avg. Number of Connections per Component (Patch–FR) | 0.7740 | 0.1395 | 0.000381 | Patch > FR | 100.00% |

*Note:* Monte Carlo standard error (MCSE) is the standard error of the posterior sample mean estimate and measures the simulation accuracy in the MCMC procedure.

and improves its robustness. Second, we used Bayesian analysis with MCMC [11, 44] to determine whether there is a statistically significant difference between patch-development and feature-request networks in terms of a macro-level network variable.

### Quadratic Assignment Procedure (QAP) Analysis

The QAP is a social network analysis method to compare two networks [3] and examine the degree of dissimilarity between them [38]. The QAP is used to test the null hypothesis that two networks are dissimilar [38]. The QAP is a nonparametric permutation-based test that preserves the integrity of the network structures [38]. 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 a second matrix.

The QAP has several advantages. First, it does not impose any specific distributional assumptions because it is a permutation-based nonparametric test [3]. Second, it takes advantage of the dyadic information represented in each matrix by preserving the integrity of the network structures [3]. Third, it can be used for nonindependent relationships [3]. Fourth, it is immune to the highly complex autocorrelation structure of network data [43]. Fifth, the QAP is relatively unbiased [43].

The QAP requires social networks, which should be represented in the form of square matrices of equal size. However, the patch-development network includes 4,675 unique developers, whereas the feature-request network includes 6,597 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 nonambidextrous 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 nonambidextrous developers in each network. We created twenty-five stratified sample networks for patch-development activities, and twenty-five stratified sample networks for feature-request activities. We created twenty-five sample network pairs by pairing patch-development and feature-request sample networks.

We used the QAP test as implemented in UCINET 6 [10]. The QAP test reports five (similarity) measures (Jaccard coefficient, Pearson correlation, simple matching, Goodman–Kruskal gamma, and Hamming distance). Although the result of the QAP test is the same across five measures, we discuss the result of the QAP test with Jaccard similarity for twenty-five sample networks pairs. The Jaccard similarity of all network pairs was found to be insignificant.[6] Thus, we accepted the null hypothesis and found that sample networks are dissimilar for all network pairs. Therefore, we found that

patch-development and feature-request networks have different network structures. The result of the QAP test is consistent with the results of the Bayesian analysis.

### Network Decomposition Analysis

Component analysis is a social analysis method used to analyze the macro-level structures of a network based on connectedness, which enables us to better understand how the network as a whole is likely to behave [34, 55]. In order to compare the macro-level structures of patch-development and feature-request networks, we used Bayesian analysis with MCMC [11, 44] to determine whether there is a statistically significant difference between patch-development and feature-request networks in terms of the macro-level structures.

We used Bayesian analysis as explained in the Bayesian analysis section. In the Bayesian analysis procedure, we used a list of 1,170 projects for patch-development activities and a list of 1,889 projects for feature-request activities as observed data. We report the descriptive statistics of macro-level network variables in Table 3. Bayesian analysis created 125,000 sample observations for a patch-development network and 125,000 sample observations for a feature-request network. We calculated the posterior means of network variables for patch-development and feature-request networks. We used the mean differences to compare the macro-level network variables of patch-development and feature-request activities and to test the relevant hypotheses. The mean differences of network variables are reported in Table 4. We test the significance of our hypotheses (zero mean differences for network variables) using the percentage of the posterior distribution of differences in mean that is greater than zero for each network variable [11, 44].

## Results

The significance of an overall regression model is tested with the *F*-statistic [53]. As shown in Table 2, the *F*-statistic of our artifact success model is significant at the 0.01 alpha level. The adjusted $R^2$ statistic is 0.487, which indicates a reasonable fit. The regression coefficient: for repeat ties is positive and significant (1.474, $p < 0.01$); for external cohesion is negative and significant (–0.721, $p < 0.01$); for degree centrality is positive and significant (2.418, $p < 0.01$); and for the average number of connections per components is positive and significant (0.834, $p < 0.01$). Internal cohesion, network centrality, and the average number of connections per components have a positive impact, whereas external cohesion has a negative impact on the output of patch-development and feature-request teams. Thus, H1a, H2a, H3a, and H4a are supported.

In addition, as also shown in Table 2, the interaction terms: of repeat ties with the artifact type is negative and significant (–0.623, $p < 0.01$); of external cohesion with the artifact type is positive and significant (0.270, $p < 0.05$); of degree centrality with the artifact type is negative and significant (–0.904, $p < 0.01$); and of the average

Table 5. Summary of Hypotheses

| Variable type | Hypothesis | Hypothesis type | Tested with variable | Results |
|---|---|---|---|---|
| Internal Cohesion | Hypothesis 1a | Artifact success | Repeat Ties | Supported |
| | Hypothesis 1b | Differential impact | Repeat Ties | Supported |
| External Connectivity | Hypothesis 2a | Artifact success | External Cohesion | Supported |
| | Hypothesis 2b | Differential impact | External Cohesion | Supported |
| Network Location | Hypothesis 3a | Artifact success | Degree Centrality | Supported |
| | Hypothesis 3b | Differential impact | Degree Centrality | Supported |
| Network Decomposition | Hypothesis 4a | Artifact success | Avg. Number of Connections per Component | Supported |
| | Hypothesis 4b | Differential impact | Avg. Number of Connections per Component | Supported |
| All Network Variables | Hypothesis 5a | Network heterogeneity | Repeat Ties | Supported |
| | Hypothesis 5b | | External Cohesion | Supported |
| | Hypothesis 5c | | Degree Centrality | Supported |
| | Hypothesis 5d | | Avg. Number of Connections per Component | Supported |

number of connections per components with the artifact type is negative and significant (–0.361, $p < 0.01$). The impacts of internal cohesion, external cohesion, network centrality, and the average number of connections per components on the output of patch-development and feature-request teams are significantly different. Thus, H1b, H2b, H3b, and H4b are supported.

As shown in Table 4, we found that the mean difference: for repeat ties is positive and significant (0.0895); for external cohesion is positive and significant (0.0247); for degree centrality is positive and significant (0.0488); and for the average number of connections per components is positive and significant (0.7740). The internal cohesion, external cohesion, centrality, and connectedness of patch-development teams are greater than those of feature-request teams. This indicates that significant heterogeneity exists in network structures between patch-development and feature-request teams. Thus, H5a, H5b, H5c, and H5d are supported. We summarize the results of our hypotheses in Table 5.

## Discussion

Existing OSS research has recognized the importance of social networks of developers in influencing the likelihood that developers will join projects [33] as well as the output of projects [30, 66]. Projects consist of artifacts and developers who contribute to artifact development. This paper aimed to understand the underlying

network structures within projects by studying them at the artifact level. Based on organizational theory, OSS development (team formation), and social networking literature, we proposed that project teams have different subnetwork structures depending on the type of artifact. We focused on patch-development and feature-request artifacts and empirically examined the differences between the social network structures of developers. Network structures were examined from micro-level and macro-level perspectives since the macro-level network structure emerges from the micro-level interactions between individual developers [34]. This two-level analysis provided a holistic picture about how patch-development and feature-request networks are different. We illustrated that the social network structure of both patch-development and feature-request teams influences the output of these networks. However, network structures have a differential impact based on the type of activity and associated artifact.

This study makes several important contributions to multiple streams of the rich literature on OSS research. Our results contribute to research on OSS development and social networks [66], team formation in OSS projects [33], and OSS project success [30, 66, 68]. We introduce the use of organizational theory on exploitation and exploration together with social network analysis as a theoretical lens to study different types of artifact subnetworks in OSS development. Specifically, we propose that OSS project activities can be classified as implementation-oriented (exploitation) and innovation-oriented (exploration) based on organizational theory [50]. 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. Recent research on OSS development has studied the social network structure of software developers as a determinant of project success [30, 65, 66]. However, this stream of research has focused on the project level, and has not recognized the fact that projects can consist of different types of activities, each of which could require different types of expertise and network structures. We contribute to this stream of research by presenting one of the first artifact-level studies of OSS development.

Prior research on OSS team formation [33] has recognized the importance of social networks in the emergence of OSS project teams. However, differences between project activities that produce different types of artifacts have not been studied. Our results advance our understanding of OSS projects by illustrating that OSS developers end up involuntarily forming different social network structures depending on the type of activity. It is important to note that developers join projects based on social cues relating to existing social ties and the status of individuals in these projects at a micro level [33]. Over time, these micro-level interactions result in the emergence of subgroups and, ultimately, larger networks [34]. Developers in patch-development teams can develop stronger, trust-based relationships, which tie them into densely connected cohesive subgroups that have their own strong norms, values, orientations, and subcultures. In turn, higher trust facilitates collaboration and information exchange in patch-development teams. However, strong norms and shared expectations reduce deviant behavior, limit search scope, and increase

selective perception of alternatives [71], and thereby cohesive subgroups are constrained in access to novel information [13, 29]. Therefore, feature-request teams can develop relatively weak ties and be sparsely connected within subgroups in order to mitigate the restrictions of strong norms on information-seeking behaviors of feature-request teams. A high number of subgroups preserves the variety of knowledge within subgroups, and fosters the diversity of ideas across subgroups [22]. A feature-request network consists of a large number of subgroups in order to foster the diversity of ideas across subgroups. Therefore, the network structure of feature-request teams enhances the team's ability to access more novel information. As a result, a patch-development network consists of a smaller number of densely connected subgroups, whereas a feature-request network consists of a larger number of sparsely connected subgroups. In our data, the patch-development network is made up of 583 components, whereas the feature-request network is made up of 940 components. Thus, patch-development networks seem to be driven by the need for a high degree of trust and collaboration, whereas feature-request networks seem to be driven by the need to access novel ideas. The explanation for this difference could be that OSS developers emphasize different social cues depending on the type of artifact.

By using multiple statistical methods, we empirically illustrate that OSS developers end up involuntarily forming different social network structures depending on the type of activity. The social network structure of patch-development teams differs from that of feature-request teams at the micro (developers and interactions) as well as macro (networks and subgroups) levels. Interestingly, the resulting social networks resemble organizational structures for exploitation and exploration in actively managed organizations. Organizational units pursuing exploitation are larger and more centralized with tight cultures and processes, whereas organizational units pursuing exploration are small and decentralized with loose cultures and processes [7]. Thus, our results are consistent with the findings in the organizational literature and they represent an important contribution to research on OSS development and social networks [66].

Our results also contribute to the literature on OSS project success [30, 66, 68] by illustrating that artifact-level analysis has potential for further examining OSS project success. Thus, we provide a more nuanced understanding of different types of subnetworks within projects. Recognizing that projects consist of different types of artifact subnetworks can help us to better understand OSS development. Project success can depend on the types of artifacts and associated developer networks. In the organizational literature, ambidexterity is defined as the ability to manage exploitation and exploration simultaneously within an organization [70]. Organizations may become ambidextrous by creating two subdivisions with different functions at the next organizational level down [7]. However, different organizational units need to be integrated to achieve ambidexterity [70]. Based on organization theory [7, 56, 70], we propose that developers who are part of both exploitation and exploration networks be termed ambidextrous developers. Ambidextrous developers integrate and control information between exploitation and exploration teams. This view is consistent with the roles of controllers in OSS projects [20]. The

identification of ambidextrous developers facilitates the calculation of the concept of project ambidexterity. This is an important contribution to the OSS project success literature [30, 66, 68] because it represents a new theoretical construct that has the potential to facilitate further study and a deeper understanding of OSS projects.

This study also makes an important contribution to practice. We show that exploitation and exploration activities in OSS development require specific network structures, based on the characteristics of each activity. It is possible that OSS project leaders could influence project success. OSS project leaders can monitor projects at the activity level and try to influence (by inviting potential developers) the formation of different types of networks depending on the activity, instead of focusing only on a project-level analysis. Thus, they can possibly better coordinate OSS projects.

Given the multidisciplinary nature of this study, its validity merits discussion from theoretical as well as methodological perspectives. The theoretical supports for this study are based on organizational theory on exploitation and exploration [7, 22, 39, 46, 47, 48, 50, 56, 58, 63], social networking [1, 12, 13, 14, 18, 25, 28, 29, 34, 71, 75, 76, 77], and OSS development [30, 33, 65, 66, 68, 74] to study artifact networks in OSS development. Thus, our theories and associated hypotheses are developed combining theories from the organizational, social networking, and OSS development literatures. The constructs used in this study (for measuring internal cohesion, external cohesion, network centrality, and connectedness within subgroups) are well established in the social networking literature and consistently used by multiple studies that analyze social network structures, including OSS development [30, 33, 34, 35, 36, 39, 45, 48, 58, 63, 65, 66, 77]. Thus, the construction of our variables is based on well-accepted theories. We also employed micro-level and macro-level measures as well as multiple statistical methods to test our hypotheses regarding heterogeneity in network structures. This approach has the advantage not only of comparing multiple aspects of network structure but also of playing a validation role (triangulation) for our findings. However, our data are restricted to data available at the SourceForge database. As discussed in the next section, the SourceForge database has some potential limitations. The impacts of these limitations should be considered in evaluating our findings. Thus, our results may be subject to additional external validation.

## Conclusions

Prior research on OSS has focused on projects as the unit of analysis. In contrast, this research represents one of the first major artifact-level studies of OSS development. This research integrates the literature on team formation with that on OSS project performance. It empirically illustrates that OSS developers who join projects might emphasize different social cues depending on the type of artifact. Consequently, networks of patch-development and feature-request developers have different social network structures based on different needs of artifact teams (a high degree of collaboration for patch-development teams and an access to novel ideas for feature-request teams). In addition, the impact of network structure on development

success depends on the type of artifact (exploitation or exploration). Understanding exploitation and exploration in the context of OSS by using constructs such as ambidexterity can help to offer a better understanding of OSS project success. OSS project leaders can monitor projects at the activity level and try to influence (by inviting potential developers) the formation of different types of networks depending on the activity. Thus, they can possibly better coordinate OSS projects.

## Limitations and Future Research

In this study, we assume that network structure affects knowledge transfer. However, we did not observe knowledge transfer directly but rather inferred it from relationships in the network structure. Knowledge can flow through other mechanisms. For example, a developer can acquire knowledge from unconnected activities by using their software or by analyzing their software's source code. In this study, we did not consider other mechanisms for knowledge flow. We did not analyze the characteristics of individual team members such as their experiences and motivations, which might also influence the extent to which knowledge is transferred or absorbed [17]. 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 [65, 66].

The OSS network data required for this study have been collected from the SourceForge database. However, some projects hosted at the SourceForge website or data associated with them may not be present in the SourceForge database. Thus, our findings should be evaluated in the context of data available at the SourceForge database. Future research can collect data from different sources and provide external validity for our findings. OSS projects ideally perform patch-development and feature-request activities through patch-development and feature-request forums. However, some projects may not use related forums. For example, some projects use bug forums for posting feature requests. Therefore, the type of activity has been identified based on activity descriptions provided by artifact teams to minimize the incorrect identification of artifact types. Some projects may not use these forums to perform related activities, which makes it impossible to identify related activities. Thus, artifact data are restricted to data identified for projects using patch-development or feature-request forums. We selected one programming language as a network boundary. Thus, our data are restricted to projects using the same programming language. Additional research using other data sets, such as projects using other programming languages, could be conducted.

It is important to note that the concept of artifact success in this study is limited by data available from SourceForge. Other measures for artifact success could possibly be developed. More sophisticated project success models that incorporate different types of artifact-level variables are an interesting area for future research. It is also possible that the activity-level network structures and determinants of project success could be different depending on project type.

As discussed earlier, the concepts of ambidexterity and ambidextrous developers proposed in this paper are important. The role of ambidextrous developers and the relative importance of exploitation and exploration networks for the success of different types of OSS projects (e.g., games, security, and others) are interesting areas for future research.

## Notes

1. In social network terminology, the local actor under study is termed ego and the actors who have ties to the ego are termed alters.

2. Details of network construction are presented in the supplement to this paper that was made available to the reviewers and is available from the authors.

3. Calculations of independent variables are presented in the supplement to this paper.

4. Details of the control variables are presented in the supplement to this paper.

5. This assumption can be relaxed to run separate regressions for each type of team (one for patch development and one for feature request). The results of this analysis are presented in the supplement to this paper.

6. Results of the QAP test are presented in the supplement to this paper.

## References

1. Ahuja, G. Collaboration networks, structural holes, and innovation: A longitudinal study. *Administrative Science Quarterly*, *45*, 3 (2000), 425–455.

2. Allison, P.D. *Multiple Regression: A Primer*. Thousand Oaks, CA: Pine Forge Press, 1999.

3. Baker, F.B., and Hubert, L.J. The analysis of social interaction data: A nonparametric technique. *Sociological Methods and Research*, *9*, 3 (1981), 339–361.

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

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

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

7. Benner, M.J., and Tushman, M.L. Exploitation, exploration, and process management: The productivity dilemma revisited. *Academy of Management Review*, *28*, 2 (2003), 238–256.

8. Bittner, E.A.C., and Leimeister, J.M. Creating shared understanding in heterogeneous work groups: Why it matters and how to achieve it. *Journal of Management Information Systems*, *31*, 1 (Summer 2014), 111–144.

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

10. Borgatti, S.P.; Everett, M.G.; and Freeman, L.C. *UCINET for Windows: Software for Social Network Analysis*. Harvard, MA: Analytic Technologies, 2002.

11. Box, G.E.P., and Tiao, G.C. *Bayesian Inference in Statistical Analysis*. Wiley Classics Library Edition, New York: Wiley, 1992.

12. Breschia, S., and Catalini, C. Tracing the links between science and technology: An exploratory analysis of scientists' and inventors' networks. *Research Policy*, *39* (2010), 14–26.

13. Burt, R.S. Structural holes and good ideas. *American Journal of Sociology*, *110*, 2 (2004), 349–399.

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

15. Cao, L.; Mohan, K.; Ramesh, B.; and Sarkar, S. Evolution of governance: Achieving ambidexterity in IT outsourcing. *Journal of Management Information Systems*, *30*, 3 (Winter 2014), 115–140.

16. Code hosting comparison for open source projects. http://opensource.com/life/12/11/code-hosting-comparison/.

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

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

19. Comparison of open source software hosting facilities. http://en.wikipedia.org/wiki/Comparison_of_open-source_software_hosting_facilities/.

20. Di Tullio, D., and Staples, D.S. The governance and control of open source software projects. *Journal of Management Information Systems*, *30*, 3 (Winter 2014), 49–80.

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

22. Fang, C.; Lee, J.; and Schilling, M.A. Balancing exploration and exploitation through structural design: The isolation of subgroups and organizational learning. *Organization Science*, *21*, 3 (2010), 625–642.

23. Faraj, S., and Sproull, L.S. Coordinating expertise in software development teams. *Management Science*, *46*, 12 (2000), 1554–1568.

24. Fleming, L. Recombinant uncertainty in technological search. *Management Science*, *47*, 1 (2001), 117–132.

25. Freeman, L.C. Centrality in social networks: Conceptual clarification. *Social Networks*, *1*, 3 (1979), 215–239.

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

27. Gelman, A., and Rubin, D.B. Inference from iterative simulation using multiple sequences. *Statistical Science*, *7*, 4 (1992), 457–472.

28. Granovetter, M.S. Economic action and social structure: The problem of embeddedness. *American Journal of Sociology*, *91*, 3 (1985), 481–510.

29. Granovetter, M.S. The strength of weak ties. *American Journal of Sociology*, *78*, 6 (1973), 1360–1380.

30. Grewal, R.; Lilien, G.L.; and Mallapragada, G. Location, location, location: How network embeddedness affects project success in open source systems. *Management Science*, *52*, 7 (2006), 1043–1056.

31. Guimerà, R.; Uzzi, B.; Spiro, J.; and Amaral, L.A.N. Team assembly mechanisms determine collaboration network structure and team performance. *Science*, *308* (2005), 697–702.

32. Gulati, R. Network location and learning: The influence of network resources and firm capabilities on alliance formation. *Strategic Management Journal*, *20*, 5 (1999), 397–420.

33. Hahn, J.; Moon, J.Y.; and Zhang, C. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, *19*, 3 (2008), 369–391.

34. Hanneman, R.A., and Riddle, M. *Introduction to Social Network Methods*. Riverside: University of California, 2005. http://faculty.ucr.edu/~hanneman/.

35. Hansen, M.T. Knowledge networks: Explaining effective knowledge sharing in multiunit companies. *Organization Science*, *13*, 3 (2002), 232–248.

36. Hansen, M.T. The search-transfer problem: The role of weak ties in sharing knowledge across organizational subunits. *Administrative Science Quarterly*, *44*, 1 (1999), 82–111.

37. He, J.; Butler, B.S.; and King, W.R. Team cognition: Development and evolution in software project teams. *Journal of Management Information Systems*, *24*, 2 (Fall 2007), 261–292.

38. Hubert, L.J. *Assignment Methods in Combinatorial Data Analysis*. New York: Marcel Dekker, 1987.

39. Jansen, J.J.P.; Van Den Bosch, F.A.J.; and Volberda, H.W. Exploratory innovation, exploitative innovation, and performance: Effects of organizational antecedents and environmental moderators. *Management Science*, *52*, 11 (2006), 1661–1674.

40. Jiang, J.J.; Chang, J.Y.T.; Chen, H.G.; Wang, E.T.G.; and Klein, G. Achieving IT program goals with integrative conflict management. *Journal of Management Information Systems*, *31*, 1 (Summer 2014), 79–106.

41. Katz, R., and Tushman, M. Communication Patterns, project performance, and task characteristics: An empirical evaluation and integration in an R&D setting. *Organizational Behavior and Human Performance*, *23*, 2 (1979), 139–162.

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

43. Krackhardt, D. QAP partialling as a test of spuriousness. *Social Networks*, *9*, 2 (1987), 171–186.

44. Kruschke, J. K. Bayesian estimation supersedes the t-test. *Journal of Experimental Psychology: General*, *142*, 2 (2013), 573–603.

45. Levin, D.Z., and Cross, R. The strength of weak ties you can trust: The mediating role of trust in effective knowledge transfer. *Management Science*, *50*, 11 (2004), 1477–1490.

46. Levinthal, D.A., and March, J.G. The myopia of learning. *Strategic Management Journal*, *14*, S2 (1993), 95–112.

47. Lewin, A.Y.; Long, C.P.; and Caroll, T.N. The coevolution of new organizational forms. *Organization Science*, *10*, 5 (1999), 535–550.

48. Lin, Z.; Yang, H.; and Demirkan, I. The performance consequences of ambidexterity in strategic alliance formations: Empirical investigation and computational theorizing. *Management Science*, *53*, 10 (2007), 1645–1658.

49. Madey, G.; Freeh, V.; and Tynan, R. The open source software development phenomenon: An analysis based on social network theory. In *Proceedings of Eighth Americas Conference on Information Systems*. Dallas, TX, August 9-11, 2002, pp. 1806–1813.

50. March, J.G. Exploration and exploitation in organizational learning. *Organization Science*, *2*, 1 (1991), 71–87.

51. Marsden, P.V. Recent developments in network measurement. In P.J. Carrington, J. Scott, and S. Wasserman (eds.), *Models and Methods in Social Network Analysis*. New York: Cambridge University Press, 2005, 8–30.

52. Mockus, A.; Fielding, R.; and Herbsleb, J. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, *11*, 3 (2002), 309–346.

53. Myers, R.H. *Classical and Modern Regression Application*. 2nd ed. Belmont, CA: Duxbury Press, 1990.

54. Nahapiet, J., and Ghoshal, S. Social capital, intellectual capital, and the organizational advantage. *Academy of Management Review*, *23*, 2 (1998), 242–266.

55. Nooy, W.D.; Mrvar, A.; and Batagelj, V. *Exploratory Network Analysis with Pajek*. Cambridge: Cambridge University Press, 2005.

56. O'Reilly, C.A., and Tushman, M.L. The ambidextrous organization. *Harvard Business Review*, *82*, 4 (2004), 74–81.

57. Raymond, E.S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* 1st ed. Cambridge, MA: O'Reilly Media, 1999.

58. Reagans, R., and Zuckerman, E.W. Networks, diversity, and productivity: The social capital of corporate R&D teams. *Organization Science*, *12*, 4 (2001), 502–517.

59. Roberts, T.L.; Cheney, P.H.; Sweeney, P.D.; and Hightower, R.T. The effects of information technology project complexity on group interaction. *Journal of Management Information Systems*, *21*, 3 (Winter 2005), 223–247.

60. Rosenkopf, L., and Almeida, P. Overcoming local search through alliances and mobility. *Management Science*, *49*, 6 (2003), 751–766.

61. Rosenkopf, L., and Nerkar, A. Beyond local search: Boundary spanning, exploration, and impact in the optical disk industry. *Strategic Management Journal*, *22*, 4 (2001), 287–306.

62. Rowley, T.; Behrens, D.; and Krackhardt D. Redundant governance structures: An analysis of structural and relational embeddedness in the steel and semiconductor industries. *Strategic Management Journal*, *21*, 3 (2000), 369–386.

63. Schilling, M.A., and Phelps, C.C. Interfirm collaboration networks: The impact of large scale network structure on firm innovation. *Management Science*, *53*, 7 (2007), 1113–1126.

64. Scott, J. *Social Network Analysis: A Handbook*. 2nd ed. London: Sage, 2000.

65. Singh, P.V. The small world effect: The influence of macro level properties of developer collaboration networks on open source project success. *ACM Transactions on Software Engineering and Methodology*, *20*, 2 (2010), 1–37.

66. Singh, P.V.; Tan, Y.; and Mookerjee, V. Network effects: The influence of structural social capital on open source project success. *Management Information Systems Quarterly*, *35*, 4 (2011), 813–829.

67. Slaughter, S.A.; Levine, L.; Ramesh, B.; Pries-Heje, J., and Baskerville, R. Aligning software processes with strategy. *MIS Quarterly*, *30*, 4 (2006), 891–918.

68. Subramaniam, C.; Sen, R.; and Nelson, M.L. Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, *46*, 2 (2009), 576–585.

69. Temizkan, O.; Kumar, R.L.; Park, S. and Subramaniam, C. Patch release behaviors of software vendors in response to vulnerabilities: An empirical analysis. *Journal of Management Information Systems*, *28*, 4 (Spring 2012), 305–337.

70. Tushman, M.L., and O'Reilly, C.A., III. Ambidextrous organizations: Managing evolutionary and revolutionary change. *California Management Review*, *38*, 4 (1996), 8–30.

71. Uzzi, B. Social structure and competition in interfirm networks: The paradox of embeddedness. *Administrative Science Quarterly*, *42*, 1 (1997), 35–67.

72. Van Antwerp, M., and Madey, G. Advances in the SourceForge Research Data Archive (SRDA). *Proceedings of the Fourth International Conference on Open Source Systems*, Milan, Italy, 2008.

73. Van Vliet, H. *Software Engineering: Principles and Practices*. 2nd ed. West Sussex,UK: John Wiley, 2000.

74. Vlas, R.E., and Robinson, W.N. Two rule-based natural language strategies for requirements discovery and classification in open source software development projects. *Journal of Management Information Systems*, *28*, 4 (Spring 2012), 11–38.

75. Wasserman, S., and Frost, K. *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press, 1994.

76. Watts, D.J., and Strogatz, S.H. Collective dynamics of small world networks. *Nature*, *393* (1998), 440–442.

77. Zaheer, A., and Bell, G.G. Benefiting from network position: Firm capabilities, structural holes, and performance. *Strategic Management Journal*, *26*, 9 (2005), 809–825.