# Evolution of Coordination Structures in OSS Development: An Exponential Random Graph Model

**Abstract**

In complex Open-Source Software (OSS) development projects, coordination structures emerge from dyadic code interactions of dispersed developers. Prior research uses methods of network science to understand the evolution of OSS developer coordination. In this paper, we propose and estimate a model to study how endogenous network properties predict the formation of OSS coordination structures. We specify an Exponential Random Graph Model (p*) to study the significance of two relational mechanisms: Preferential attachment and knowledge similarity. In the model, the code-mediated nature of OSS coordination is reflected with a graph in which edges represent the coordination efforts of two developers working on the same software functions. We empirically estimate our model using granular development data of 619 developers in Nova, one of the oldest projects of OpenStack, working on 2597 Python files between 2012 and 2016. We find statistical evidence of positive non-linear (rather than linear) preferential attachment which causes skewed distributions distinct from a power-law. Further, we show that knowledge similarity has a significantly positive effect. Non-linear preferential attachment explains the formation of developer relationships across long evolutionary periods, but knowledge similarity impacts only early stages. Our model contributes to the literature on software evolution and OSS.

***Keywords***: OSS, software evolution, Exponential Random Graph Model, preferential attachment, homophily

# Evolution of Coordination Structures in OSS Development: An Exponential Random Graph Model

Open source software (OSS) development has become a commercially viable model for organizing large-scale software projects such as Linux, Apache, Perl, or Open Stack (Cataldo and Herbsleb 2013; Fitzgerald 2006; Raymond 1999). In such projects, a large group of virtually distributed software developers collaboratively build and constantly advance complex software (Howison and Crowston 2014). During this process, change in the software architecture is inevitable given the need to advance the software's functionality in response to a constant influx of the software's diverse customers and users while also ensuring its maintainability(Coleman et al. 1994; Joblin et al. 2017). As the OSS's technical architecture and its underlying components change over time (Myers 2003), the project's organizational structure may also need a change in order to prevent or resolve emerging technical interdependences (Cataldo and Herbsleb 2013; Joblin et al. 2017; Scholtes et al. 2016). For example, if two or more OSS developers work independently on different software components, they may cause unexpected functional interdependences in the software architecture, unless they engage in co-work and coordinate their work by modifying each other's software functionalities. Developers' unawareness of coordination needs has shown to have negative implications for software quality including its maintainability (Cataldo et al. 2008, 2009; Cataldo and Herbsleb 2013; Lindberg et al. 2016). As OSS developers self-organize outside formal employment relationships and in-house software engineering management (Joblin et al. 2017; Lindberg et al. 2016), there is a need to understand how they successful coordinate their actions as the software evolves (Setia et al. 2012).

Prior research on software engineering has turned to network modeling techniques to evaluate and understand the evolution of developer coordination on complex OSS projects (Hahn et al. 2008; Joblin et al. 2017). Taking a network view, this stream of work represents the software's organizational structure for coordination as a developer coordination network with developers representing nodes and coordination relationships among the developers representing edges (also referred to as ties or links). Following prior work, we define the evolution of developer coordination as the temporal change of structures that reflect code-mediated interactions and coordination among developers (Hahn et al. 2008; Joblin et al. 2017). Empirical studies on software evolution have shown that a developer coordination network evolves efficiently if it exhibits organizational principles akin to those observed in other technical but also social networks (Louridas et al. 2008; Myers 2003; Potanin et al. 2005).

We seek to advance this work further by focusing on the endogenous coordination patterns that underpin the formation of developer coordination structures as a whole. Instead of only describing the coordination structure as a whole, we investigate how the development behavior of OSS developers manifests in the formation of dyadic coordination relationships among developers within the OSS coordination structure (Hahn et al. 2008; Joblin et al. 2017). We conceptualize dyadic coordination relationships as edges where two developers add, remove, or modify code of the same software function to resolve functional coupling between different software components. It reflects artifact-mediated coordination related to the same content (Bolici et al. 2016; Joblin et al. 2017).

Building upon network exchange and coordination theory, as well as statistical advances in modeling network formation (Concas et al. 2007; Faraj and Johnson 2010; Faraj and Sproull 2000; Louridas et al. 2008; Maillart et al. 2008; Wang et al. 2017), this paper, examines two complementary micro-level mechanisms and their role in the non-random formation of complex coordination structures: On the one hand, we focus on an endogenous relational mechanism referred to as *preferential attachment* observed in technical as well as social networks (Chaikalis and Chatzigeorgiou 2015; Concas et al. 2007; Faraj and Johnson 2010; Scholtes et al. 2016). In the context of OSS coordination, it describes a developer's "bias" to form new coordination relationship with developers who are highly connected because they coordinate and interact with many developers. On the other hand, we also consider a relational mechanism related to two developers' *knowledge similarity* to account for variability among two developer's common experiences with working on certain software functionalities (Crowston and Kammerer 1998; Singh, Tan, and Youn 2011; Wang et al. 2017). Our goal is to investigate how these two relational mechanisms predict the formation of non-random coordination efforts among pairs of developers in the context of large OSS projects involving hundreds or thousands of developers. Specifically, we ask the following question: *What is the role of relational mechanisms of preferential attachment vis-a-vis knowledge similarity for the formation of evolving OSS coordination networks?*

To identify the likelihood of nonrandom appearances of coordination relationships formed by developers in a large OSS project, we introduce a stochastic network prediction model, an Exponential Random Graph Model (p*) that considers endogenous edge-related as well as node-related attributes to predict coordination edges in an evolving OSS development network considering two relational mechanisms - preferential attachment and coordination knowledge similarity. We empirically examine our model using a large development project with the OSS cloud-computing platform Open Stack (OpenStack 2016). We chose one of the oldest packages within OpenStack, the Nova package, in which 619 developers have been creating, adding, and modifying software functionalities of 2597 Python files between 2012 and 2016. They worked on 18141 unique functions for Open Stack's networking services and made 32546 commits. To conceptualize the coordination structure as a network, we establish a coordination relationship between two developers if they commit on the same function in a python file in Nova software. Given the success of coordination in Nova, reflected in the software's maintainability and quality, it is well suited to examine the formation of coordination relationships.

Our empirical analysis reveals important, somewhat unexpected findings of non-random coordination link formation in OSS. First, we find that there is a preferential attachment. However, unlike in scale-free networks where there is a linear pattern of how the "the richer get richer", we find a non-linear preferential attachment. This suggests that only those "star" developers who are highly involved with many developers across many different functions cause new coordination relationships to form. Second, we find that knowledge similarity among two developers has a positive effect, suggesting that overlapping development knowledge increases the likelihood of coordination. Third, we also find that while the relational mechanism of preferential attachment explains a five-year evolutionary period, the effect of knowledge similarity is only significant in the early stages of evolution. Our ERGM models and their results advance theory and method of software evolution and management.

## 2 BACKGROUND

In this section we provide the theoretical and methodological background for studying the formation OSS coordination networks. We summarize key properties of OSS coordination networks and discuss the two micro-level relational mechanisms that are essential for their formation: preferential attachment and knowledge similarity. After that we discuss statistical models for network formation.

## 2.1 OSS Developer Coordination as Socio-technical Networks

Scholars of software evolution have performed large archival studies that provide insights into how an OSS architecture evolves into more or less desired outcomes, such as maintainability and low risk of software failures (Coleman et al. 1994; Scholtes et al. 2016). Empirical evidence suggests that whether the OSS's software architecture evolves desirably depends upon the coordination structure among the developers. Prior studies show that coordination structures in which developers fail to effectively coordinate their efforts in developing software functions (e.g., by collaborating in resolving functional interdependencies and artifact coupling) - lower software maintainability as well as developer productivity and increase the risk of software failures (Cataldo and Herbsleb 2013; Sosa et al. 2004).

Unlike inhouse software development, does OSS coordination follow a self-organizing logic. Thus, coordination structures emerge from dyadic interactions among multiple developers who select software development tasks (e.g. a new feature request or a bug report) at their own time and guided by their individual interests (Lindberg et al. 2016; Madey et al. 2002). To empirically examine such self-organizing coordination structures, scholars in software engineering have turned to network theories and techniques (Cai and Yin 2009; Grewal et al. 2006; Hahn et al. 2008; Hong et al. 2011; Meneely et al. 2008; Meneely and Williams 2011; Scholtes et al. 2016; Singh, Tan, and Mookerjee 2011; Zanetti et al. 2013). In this stream of work, scholars represent, and model developer coordination networks based on different assumptions about what represents a coordinative link among two developers in the network. On the other end of the spectrum, there is a primarily social view about coordination, in which one assumes that if two developers work on the same project repository (e.g., projects on sourceforge.net) they coordinate their actions.  Thus, if two developers contribute to the same project there is an edge (or a link) among them in the developer coordination network (Grewal et al. 2006; Singh and Phelps 2012). However, such a network conceptualization ignores that even if two developers work within the same project, they not necessarily may no need to coordinate their actions because they do not work on the same artifacts (e.g., files). Thus, in contrast to a socialized view, a socio-technical view assumes that the two developers' joint contribution to the same artifact reflects coordinative interactions among developers (Cataldo and Herbsleb 2013; Joblin et al. 2015, 2017; Olivera et al. 2008). Only then they coordinate their actions because they engage with each other's code to resolve interdependencies between artifacts, such as executable files or functions in the source code file (Joblin et al. 2015, 2017). Thus, a link between two developers requires an artifact-based interaction centered around files and functions.

Scholars in software engineering have pointed out a function-based network representation is superior to a file-based one in representing coordinative actions: Adding code to a common file may not reflect that developers actually try to resolve functional interdependencies (Joblin et al. 2015, 2017). In this paper, we follow this fine-grained logic and assume that developers form coordination ties if they work on common software functions (rather than files). A coordination network represents edges and nodes, where nodes are the developers and edges represent code contributions to the same function in a particular source code file.

Using network representations of coordination in OSS, prior work has developed static or temporal network structures to offer a rich description of developer coordination structures as well as their evolution over time (Hong et al. 2011; Joblin et al. 2017; Madey et al. 2002). Such work has pointed us to distinct organizational properties of efficiently evolved coordination structures.

*1. Scale freeness:* Prior literature on developer coordination networks describe efficient networks as scale free. In a scale-free developer network, a few developers are highly connected, while a large number of developers only have a few ties (Joblin et al. 2017). Scale-freeness is associated with reliability and scalability. Prior literature suggests that networks with skewed distributions tolerate coordination breakdowns among a large number of loosely connected developers without negative implications for the software's maintainability (Cataldo and Herbsleb 2013). One of the possible explanations of emerging scale freeness in networks is the preferential attachment mechanism, which induces a power-law degree distribution (Barabási and Albert 1999; Joblin et al. 2017). The Barabasi-Albert model of scale-free networks starts with a small number ($m_0$) of nodes and at each time step $m$ edges are added which links to $m$ pre-existing nodes. The preferential attachment model asserts that a new node links with an existing node with a probability $\Pi(k_i) = k_i/(\sum_j k_j)$ where $k_i$ and $k_j$ is the degree of an existing node $i$ and $j$ respectively. The resulting degree distribution follows a power-law distribution of the form $p(k) \sim k^{-\gamma}$ with the power-law exponent $\gamma$ typically in the range $2 < \gamma < 3$. However, it is worth noting that OSS coordination networks can also be efficient if they follow a skewed distribution that does not follow the form of a power law (Kunegis et al. 2013).

*2. Modularity:* While scale freeness describes the number of coordination ties of an individual node in the network as a whole, does modularity describe the coordination ties among a node's immediate "neighbors", or in other words, direct collaborators? In an artifact-based coordination network, modularity is high when the neighbors of node *i,* those with a coordination edge with the node, also have connections to other direct neighbors of node *i*. The more locally connected, the more "clustered" the network. Prior literature shows that clustering is a common feature of coordination developer networks that are efficient. It facilitates specialization and allows developers coordinate their actions (Baldwin and Clark 2006; Cataldo and Herbsleb 2013; Joblin et al. 2017).

*3. Hierarchy:* Prior literature suggests that highly efficient coordination networks depict an organizational structure with a hierarchical logic (Borgatti and Everett 2000; Crowston and Howison 2005; Joblin et al. 2017): At the center of the network are several developers with a very high number of ties to other developers. They are part of a core group of developers who are also clustered because they coordinate their actions among each other. Developers in their periphery are less connected with the core group, but not disconnected from them. A core/periphery structure reflects on an organization mechanism that transcendent local coordination among densely clustered developers. Prior literature suggests that it facilitates coordination among different "subgroups" within the network (Hinds and McGrath 2006; Rullani and Haefliger 2013).

Prior literature on network formation in OSS suggests that developer coordination relationships that underpin the efficient evolution of OSS coordination do not form randomly (Barabási et al. 2002; Wang et al. 2017). We discuss the micro-level mechanism that may cause the formation of coordination relationships to emerge.

## 2.2 Micro-level Mechanisms of Coordination Network Formation

We posit that there are micro-level "forces" that underpin the formation of edges and lead to the emergence of efficient coordination network structures. Such arguments are rooted in theories and methods of network evolution devoted to the understanding of the underlying mechanism of link (or edge) formation (Barabási et al. 2002; Wang et al. 2017). Two mechanisms cause the emergence of scale-free coordination networks with modular and hierarchical order: *preferential attachment* and *developer (or node) similarity*. We discuss each mechanism and the role of

OSS coordination network formation first.

*1. Preferential attachment:* The linear preferential attachment mechanism signifies the so-called "Matthew effect" (the richer get richer): New links do not form randomly but attach to highly connected nodes with a high degree (number of edges)(Barabási and Albert 1999). Translated to the context of OSS coordination this implies that the higher the degree developer, those who coordinate their efforts with many others, attract the formation of new edges from other developers. This is due to the status of a highly connected developer, since it signals experience and influence, offering a developer a valuable resource when developing software functions. Prior literature in OSS coordination networks provides some descriptive evidence that hints at linear preferential attachment in OSS coordination, however there are studies that suggest this mechanism may unfold non-linearly (Faraj and Johnson 2010; Joblin et al. 2017).

*2. Similarity:* A similarity-based mechanism for link formation complements the edge-related mechanism of preferential attachment, by considering the developers' attributes (e.g. such as the skills and knowledge of a developer) to explain the formation of new links in a network (Wang et al. 2017). A similarity-based argument suggests that homophily among nodes is a force for tie formation: It asserts that nodes with similar attributes in common are more likely to connect than that of nodes with dissimilar attributes. Prior literature suggests that in the context of OSS developer coordination there is one important similarity attribute: The developer's knowledge (and experience) in designing and implementing software functionalities (Kovalenko et al. 2018). There are contradictory arguments about how the effect of similarity-based tie formation unfolds. On the one hand, one may assume that developers who work on semantically similar functions are more likely to seek to coordinate their actions in resolving functional interdependencies (Cataldo and Herbsleb 2013; Joblin et al. 2017; Qiqi Jiang et al. 2019). On the other hand, some counterarguments suggest a negative effect of similarity: Due to the specialization in OSS, developers with similar experiences are less likely to coordinate their efforts. Instead, developers with high dissimilar but complementary knowledge are needed to coordinate their effort to resolve functional interdependencies (Baldwin et al. 2014; Scholtes et al. 2016).

Even though prior literature points to the importance of both mechanisms for the evolution of OSS coordination networks, there is insufficient statistical evidence of their independent effects on the formation of coordination edges (Joblin et al. 2017; Wang et al. 2017). One of challenges in delineating the causal effects lies with the statistical models used. Observing network properties like scale-freeness over time (Joblin et al. 2017) provides only limited insights into the micro-level mechanism that causes network evolution. Further advanced regression techniques and econometric modeling builds upon the assumption of independence of observations and thus prevent the explanation of the probability of a link based on the presence (or absence) of a link between two nodes in a network.

We address this research gap by utilizing statistical network models that allow us to simultaneously examine different underlying micro-mechanisms for link formation and advance our theoretical understanding of the network formation. We discuss those models next.

### 2.3 Network Formation Modeling

Over the last years, several inferential statistical models for network formation have been developed to simultaneously analyze diverse network mechanisms underlying network formation while also accounting for the endogenous dependencies within it (Faraj and Johnson 2010; Lubbers and Snijders 2007; Robins et al. 2007). The most common and most advanced ones are Exponential Random Graph Models (ERGMs), Stochastic Actor Oriented Model (SAOM), and Relational Event Model (REM). These models allows us to infer whether certain local network attributes related to edges and/or nodes explain tie formation (Goodreau 2007; Liang et al. 2013; Morris et al. 2008). Even though these models have been widely adopted to understand the formation and evolution of social and biological systems (Chakraborty et al. 2019; Cranmer and Desmarais 2011; Goodreau 2007; Khalilzadeh 2018; Lubbers and Snijders 2007; Morris et al. 2008; Robins et al. 2007; Simpson et al. 2011), they have rarely been used in the context of software development projects (Wagstrom and Datta 2014). ERGMs are a family of statistical models that allow for the probability of a link to be dependent upon the presence of other links in the network as well as other parameters (Van Der Pol 2016). Simply speaking, ERGMs observe the probability of links in a network to produce a variety of samples of networks with the goal to infer the odds of a particular link (Van Der Pol 2016). ERGMs are best suited for cross sectional network data, a data structure that is most easily accessible in empirical OSS development coordination. For example, it is possible to construct a functional coordination network as discussed in 2.1 using command histories accessible via version control systems like Git (git 2017) (see more details also in section 3). As OSS coordination structures evolve over time, it is possible to observe and describe coordination structures among developers for shorter and longer evolutionary periods. ERGMs can then be applied to examine the formation at a network at different evolutionary stages of coordination.

A unique property of ERGMs is that they make it possible to establish a causal link between certain node and edge attributes and the formation of developer coordination ties over a particular time period compared to a condition of random tie formation in networks of similar size at a particular point of time. Thus, it offers a statistical way to explain the relative role of preferential attachment and knowledge similarity for coordination tie formation, while also controlling for other potential counterfactuals (e.g., developer productivity).

In the next section, we will build upon these conceptual and methodological insights to develop and estimate an ERGM model specified for evolving OSS coordination networks. The main goal is to examine the effect of preferential attachment and knowledge similarity for formation of efficient OSS coordination networks.

## 3 METHOD AND DATA

In this section, we discuss the data and methods used to specify and empirically estimate an ERGM model for OSS coordination network evolution. Our goal is to understand the role of preferential attachment vis-à-vis knowledge similarity for the formation of efficient OSS coordination structures.

### 3.1 Study Setting and Data

For this study, we selected the developer community of OpenStack to study the role of preferential attachment and knowledge similarity in the evolution of OSS coordination networks. OpenStack is an OSS software platform for cloud computing, typically deployed as infrastructure-as-a-service under an Apache 2.0 license (Rosado and Bernardino 2014). Released in 2010, OpenStack utilizes object-oriented programming in Python. For our modeling of OSS coordination networks, we chose the core developer group involved in the project Nova who is authorized to modify Nova's source code as our sample (OpenStack 2019; Teixeira and Karsten 2019).The Nova package of OpenStack automates and manages resources across virtual machines, containers and bare metal servers. Nova interoperates with virtual networking technologies such as KVM, VMWare, Xen, and Linux technologies such as LXC and LXD (Mishra 2017; Openstack 2019). Nova is one of the oldest of the 32 projects within OpenStack, in which the project's developers focus their efforts on a complex module for networking services.

Our sample of NOVA developers use web-based software development tools to coordinate their development activities. The most central one is Gerrit, a web-based distributed code review system, which supports the development workflow and version control using Git (git 2017). Our study focused on the period between Jan 2012 and Dec 2016, in which 619 project developers worked on *2597* Python files. We chose Jan 2012 as the starting month because at that time the size and the contribution activities of the developer group had grown to more than 144 developers, a size where coordination among virtually and temporally distributed developers becomes more challenging. Further, our sampling was also guided by the structure of the properties of the coordination networks in NOVA. After constructing the coordination networks, we choose an evolutionary period at which the network properties suggested an efficient coordination structure (see section 2.1). A visual inspection (see result section 4.1 and 4.2) suggested that the network is modular and shows a hierarchical structure.

This study uses large volumes of granular behavioral trace data as well as content data. To construct our data, we collected information about the developers' sequential commit activities using Gerrit. Further, we also extracted the source code files and their content associated with a commit. The source code allows us to extract functional calls. In Table 1 we provide a summary of the data structure we collected for every commit between 2012 and 2016. As mentioned in 2.1. such functional calls are essential for fine-grained coordination modeling.

Table 1: Summary of data structure

| Attribute | Description |
|---|---|
| Commit-id | *A unique identification number of 40 digits for a change committed by a developer* |
| Commit time | *Time at which a developer commits on a Python file. Each commit time has a unique commit id* |
| Python files | *Files with .py extensions where developers make additions, deletions, and changes in the codes* |
| Functions | *Functions in Python files where developers make changes during a commit* |
| Developer-names | *The core developers in Nova who commit on Python codes.* |

## 3.2 Construction of the Developer Coordination Network

As described in section 2.1, we follow a fine-grained logic to construct coordination networks(Cataldo and Herbsleb 2013; Joblin et al. 2017). We assume that developers form coordination ties if they work on common software functions (rather than files). This coordination network is represented by edges and nodes, where nodes are the developers and edges represent code contributions to the same function in a particular source code file. To construct our coordination ties, we matched the developers based on the functions associated with their commits (see Table 1).

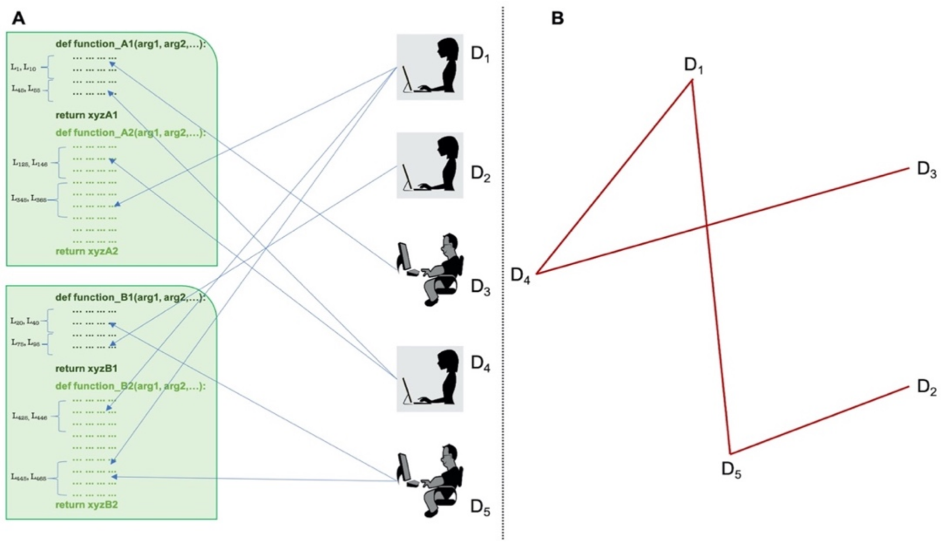

Figure 1 (A) Example of five committers ($D_1$, $D_2$, $D_3$, $D_4$, and $D_5$) working on two files, each file having two functions. (B) The five committers are connected if they work on the same function.

Developers who worked on the same function over different cumulative time periods are connected. We analyzed *619* developers who made *32546* commits on *2597* Python files in *Nova* for the following cumulative time periods: 2012, 2012-2013, 2012-2014, 2012-2015, and 2012-2016. Our choice of the length of these evolutionary periods accounts for the yearly releases of Open Stack. Second, it ensures a sufficiently large network size yielding greater statistical power. Moreover, the time periods have comparable network metrics (See Table 3). The network construction is illustrated in Figure 1.

Let us consider the tuple of commits {$D_1$, $f_a$, $L_{345}$}, {$D_1$, $f_a$, $L_{425}$}, {$D_1$, $f_a$, $L_{448}$},… , {$D_5$, $f_b$, $L_{448}$}, where *D* represents a developer with commit rights, *f* stands for a Python file and *L* provides the line number of commit change. In Figure 1A, $D_1$ commits on function *def function_A2* between lines $L_{345}$ and $L_{365}$, and another committer $D_4$ commits on the same function at line numbers between $L_{125}$ and $L_{146}$. We infer coordination between the two developers $D_1$ and $D_4$ on the function *def function_A2* whenever either of them commits a change in the function in the code. This method of link formation ($D_1$, $D_4$) is extended to all the developers who commit changes in different functions across all the Python files in *Nova.* As shown in Figure 1B, the weighted links of the developers' coordination network are: ($D_1$, $D_4$), ($D_1$, $D_5$), ($D_2$, $D_5$), and ($D_3$, $D_4$): developers $D_1$ and $D_5$ are connected since both committed changes on the function *def function_B2, $D_2$* and $D_5$ committed to the function *def function_B1*, and $D_3$ and $D_4$ commit changes on *def function_A1*. We further develop a weighted version of the network by aggregating multiple links between the developers $D_1$ and $D_4$ to a weighted link ($D_1$, $D_4$, *w*), *w* is the size of the overlap of the common functions $D_1$ and $D_4$ work on. We construct networks of developers following the cumulative time periods defined above.

## 3.3 Model Specification

Following our discussion in section 2.3, we use Exponential Random Graph Models (ERGMs) to understand how the role of preferential attachment and knowledge similarity for OSS coordination network formation over a period of time. To the best of our knowledge, prior to our current work, ERGMs have rearely been applied in information systems literature (Faraj and Johnson 2010; Shi et al. 2016). Given a network and a set of explanatory variables, an ERGM models the probability distribution function of a network. A basic ERGM defines the probability distribution function of the network given a set of actor *n* as:

$$Pr\ (Y = y|X = x) = (1/\kappa) exp\ (\Sigma_k \theta_k Z_k(y, x))$$

Where $Y$ is an array of size $n \times n$ containing ties of network variables with $y$ realizations, $X$ is an array of size $n \times p$ containing individual attributes with $x$ realizations, $Z_k\ (y,\ x)$ is a network statistic corresponding to any realization of $y$, $\theta_k$ is the coefficient of network statistics $Z_k\ (y, x)$ , and $1/\kappa$ is a normalizing constant to ensure the probability is between 0 and 1. The summation is taken over all the network statistics, which are included in the model. The network effects could also be dependent on exogenous measures pertaining to certain attributes of the nodes. In this paper, we focused on two theoretical mechanisms – preferential attachment (node attribute) and knowledge similarity (edge covariate) to specify our ERGM.

We model the observed networks of developer coordination as a function of endogenous effects. Specifically, in our study, we include the following three endogenous factors to statistically infer their explanatory role for OSS coordination network formation: (1) non-randomness (number of coordination edges), (2) preferential attachment (degree of the node), and (3) knowledge similarity (number of common functions). While it can be assumed that OSS structures form non-randomly, it is not clear how preferential attachment and knowledge similarity affect the formation of coordination times in large coordination network structures. In addition to those three key relational mechanisms (operationalized with node and edge attributes), we also included a set of control variables that may confound the effect of the relational mechanisms. In Table 2 we provide a summary of mechanisms and parameters that have been included in our empirical model.

Table 2: ERGM variables constructs, mechanism and measures

| **Construct** | **Description of mechanism/variables** | **Parameters** |
|---|---|---|
| **Relational mechanisms/endogenous variables** | | |
| Non-randomness | *The non-randomness factor infers whether the number of coordination edges among developers do not form randomly. The measure "number of coordination edges" allows us to statistically infer non-randomness. If the number of coordination edges in the network is statistically larger than the number of edges observed in a random structure, we can assume non-randomness.* | $E_i$ represent the number of coordination edges |
| Preferential attachment | *Preferential attachment refers to the concept of rich getting richer. In network science preferential attachment reflects the tendency of higher degree nodes to received new links. In our research preferential attachment refers to the situation in which new developers choose to coordinate with already developers with a larger number of coordination ties (and thus high coordination attention). We statistically examine preferential attachment with degree of a node ($k_i$) as an ERGM node covariate.* | $k_i = \sum_{j=1}^{N} Y_{ij}$ , where$Y_{ij}$ is the adjacency matrix, which takes value 1 if there exists at least one |

| | | |
|---|---|---|
| | | common function between two developers. |
| Knowledge similarity | *Knowledge similarity represents the commonality among two developers. The similarity mechanism examines whether knowledge similarity among two nodes causes the formation of a coordination edge between them. To examine this, we include the variable number of common functions between two developers as an edge level parameter, which measures the total number of common functions two developers have committed changes during a time interval. In network parlance, this represents the weight of an edge between two developers.* | $\Lambda_{ij}$ represent knowledge similarity between two developers, measured based on the number of common functions between two developers *i* and *j* |
| **Controls** | | |
| Developer commit effort | *A developer's commit effort may also confound the formation of a coordination edge. The commit effort is a measure of a developer's productivity used in prior literature on software engineering* (Adams et al. 2009; Sornette et al. 2014). | $C_i$ measures the number of commits by a developer *i* |
| Developer's code added | *Following prior literature, we add a second measure of a developer's productivity (Koch and Schneider 2002) which may affect link formation. We measure the lines of python code added (LCA) over the total number of commits during a time interval. We specifically consider files with ".py" extensions, since those are the main focus of when studying developer coordination*(Cataldo and Herbsleb 2013). | $LCA_i$ estimates the number of Python lines of code added by the developer *i*. |
| Developer's code removed | *Following prior literature, we add* a third measure of a developer's productivity (Koch and Schneider 2002): *Lines of Python code removed (LCR). It measures the number of Python code deleted by a developer while committing changes to a Python file. Again, we specifically consider files with ".py" extensions.* | $LCR_i$ represents the number Python lines of code removed by the developer *i* |
| Developer tenure | *Following prior literature, we also account for the tenure of a developer as tenure may increase the likelihood of edge formation (Faraj and Sproull 2000). We measure tenure with the total time of activity of the developer, starting from the first commit time through the last commit time when she commits changes on software codes.* | $\Gamma_i$ describes the tenure of a developer *i* |
| Developer's code quality | *Following prior literature, a developer's code quality may also affect the formation of an edge (Mishra and Otaiwi 2020). We measure a developer's code quality with the mean of maintainability indices of Python files committed by the developer. The maintainability index is computed using the formula proposed by Microsoft Visual Studio* (*Microsoft VS Docs 2020; Virtual Machinery 2019)* | $DCQ_i$ describes how much the Python codes edited by developer *i* are maintainable. |

The final ERGM defined as the probability distribution function of tie formation between OSS developers is given as:

$$\Pr(Y = y \mid X = x)$$

$$= (1/\kappa)\exp\left(\theta_1\ E(y,x) + \theta_1\ k(y,x) + \theta_3\ \Lambda(y,x) + \theta_4\ C(y,x) + \theta_5\ LCA(y,x) + \theta_6\ LCR(y,x) + \theta_7\ \Gamma(y,x) + \theta_8\ DCQ(y,x)\right)$$

Where $\theta_1, \theta_2, \ldots\theta_8$ are coefficients of network statistics.

## 4. RESULTS

In this section, we report the results of our study. We first present a visualization of the final coordination network and a descriptive analysis of the coordination network as whole. This descriptive analysis provides an insight into the properties of the coordination network as it evolves over time. It offers insights into the efficiency of the network. Afterwards, we present and discuss the results of our ERGM modeling to statistically explain how the OSS coordination network forms. Specifically, we focus on the role of preferential attachment vis-à-vis knowledge similarity and a descriptive analysis of the coordination network as whole. This descriptive analysis provides an insight into the properties of the coordination network as it evolves over time. It offers insights into the efficiency of the network. Afterwards, we present and discuss the results of our ERGM modeling to statistically explain how the OSS coordination network forms. Specifically, we focus on the role of preferential attachment vis-à-vis knowledge similarity.

### 4.1 Descriptive Analysis of Coordination Network

In Figure 2, we visually explore the coordination network structure of developers in its final evolutionary stage. It considers the coordination actions of all developers between 2012 and 2016. The network graph was produced in Gephi, using the Fruchterman-Reingold algorithm, a force-directed layout algorithm in which edges between two nodes are represented as springs

(Fruchterman and Reingold 1991).

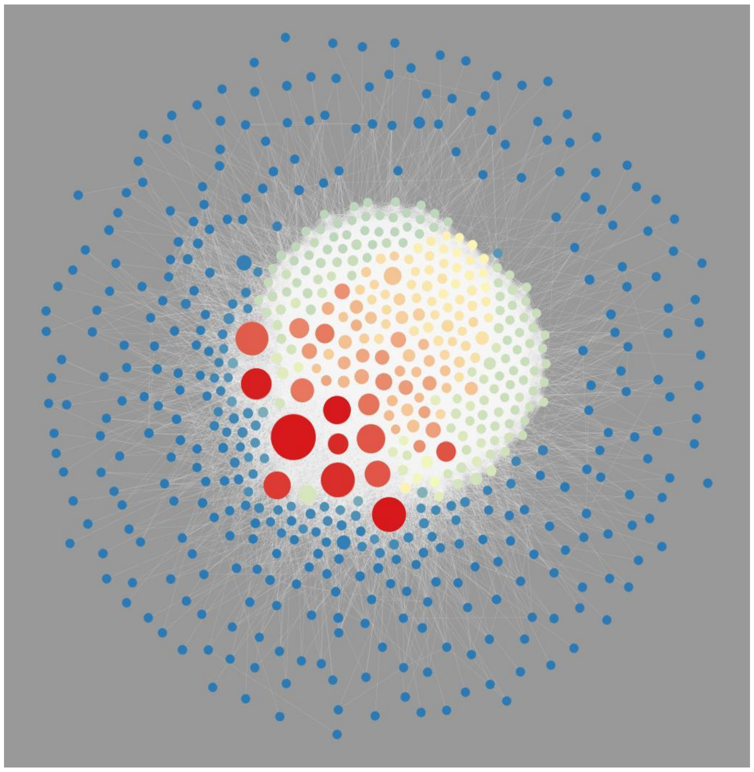

Figure 2: Developer's coordination network. Node size is proportional to the total number of commits of a developer (node). Node color reflects the degree of the node and reflects the coordination effort of the developers: Blue nodes have a very low degree (<5), while red nodes have a very high degree (>250).

A visual examination of the network hints at a hierarchical structure with at least three distinct clusters of developers in the center – nodes colored in red, orange, and green – surrounded by a larger number of developers in the periphery (blue nodes). As such the coordination network exhibits principles that facilitate efficient coordination (Joblin et al. 2017). Within the three clusters, developers are densely connected because they coordinate with each other. One of the three clusters (orange nodes) serves as a connector between the other two clusters (red and green nodes). However, developers in those coordination clusters (red and green nodes) have few direct connections with each other, suggesting that they are not directly coordinating with each other. The structure of the network illustrates the efficient nature of developer coordination in Nova.

Developers cluster into distinct coordinative groups in which the group members' coordination efforts focus on common programming knowledge and functionality. A “connector” hub acts as a bridge to facilitate the coordination between two semantically distinct groups.

Table3: Networks metrics over time

| **Network metrics** | **P1** | **P2** | **P3** | **P4** | **P5** |
|---|---|---|---|---|---|
| Average Degree | 27.83 | 64.518 | 85.640 | 96.695 | 103.932 |
| Network Diameter | 4 | 4 | 5 | 5 | 5 |
| Network Density | 0.194 | 0.195 | 0.193 | 0.173 | 0.168 |
| Average Clustering Coefficient | 0.717 | 0.780 | 0.798 | 0.794 | 0.779 |
| Assortativity | -0.222 | -0.225 | -0.229 | -0.227 | -0.215 |
| Average Path Length | 1.983 | 1.970 | 1.985 | 2.019 | 2.027 |

Table 3 presents the descriptive of the six-network metrics of the developer coordination networks cumulatively formed during five cumulative time periods (P 1: 2012, P2: 2012-2013, P3: 2012-2014, P4: 2012-2015, P5: 2012-2016). We chose those metrics as they allow us to judge the efficiency of the network (Joblin et al. 2017). The average degree increases over time since the edges and nodes increase (see Table 3), suggesting that coordination efforts increase as the social structure grows. The network diameter, defined as the longest of all shortest paths stabilizes at the value of 5. In other words, the coordination efforts are dense and do not span all developers. The developer coordination network has a relatively low network density (< 0.2) and a higher clustering coefficient (about 0.7). A high clustering is also indicative of modularity and specialization among developers (Joblin et al. 2017; von Krogh et al. 2003).

A high clustering is also indicative of modularity and specialization among developers (Joblin et al. 2017; von Krogh et al. 2003). Further, low density with high clustering suggests the frequent occurrence of coordination triads, in which three developers are all in coordination with each other. Our network is relatively disassortative (about -0.22) indicating that low degree developers, those with a lower coordination effort, typically establish a coordinative edge with high-degree developers (and vice-versa). In other words, they get help from more developers who are highly aware of coordination needs(von Krogh et al. 2003). Developers with low coordination profiles are not organized into separate homogenous cliques but remain connected to the developers with a high coordination profile (suggesting hierarchy in the coordination structure). In all the networks, assortativity remains about -0.22 suggesting moderate levels of heterogeneity in coordination among the developers with respect to their degree. It suggests that the coordination does not necessarily happen among developers with a similar coordination profile (high degree-high degree, and low-degree-low-degree). Finally, we note that the average path length in the networks remains almost constant at around 2. Low values of average path length suggest that on average it takes two steps to connect from one developer in the network to another developer. The network metrics indicate that while developers interact with each other within focused coordination groups (as shown in Figure 2), they are only indirectly coordinating their actions with another group focusing on another set of functions.

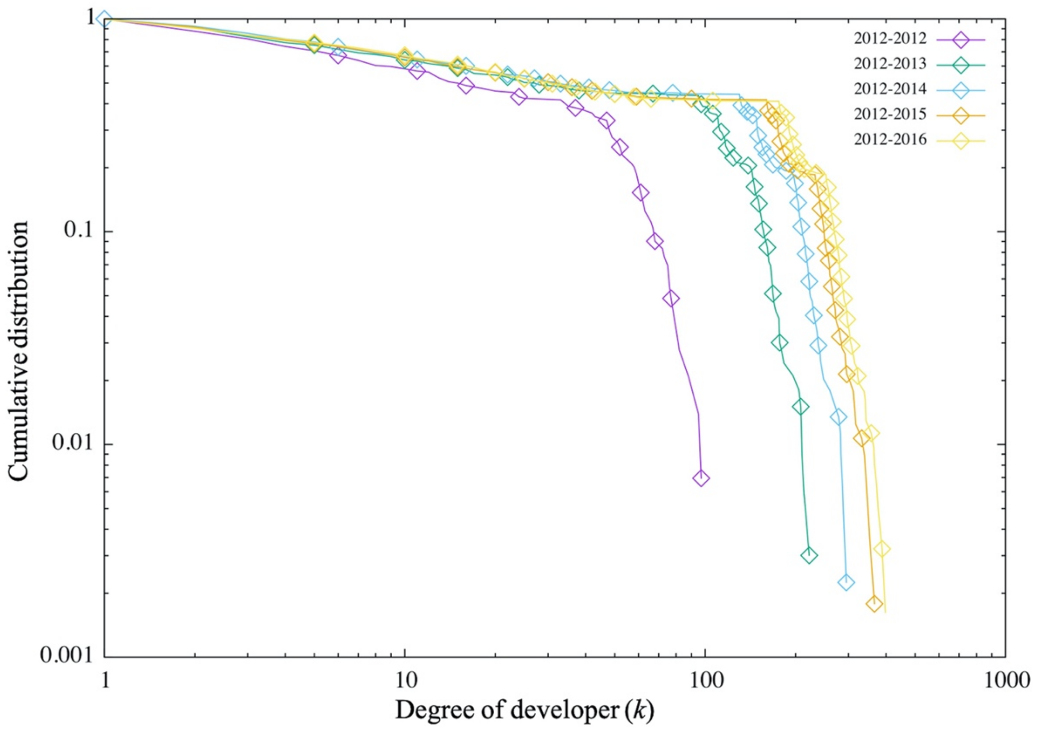

Figure 3: Cumulative distribution of degree of developers in 2012 (purple), 2012-2013 (green), 2012-2014 (cyan), 2012-2015 (orange), and 2012-2016 (yellow).

We next explore the connectivity of coordination patterns visually. In Figure 3 we plot the cumulative distribution of the coordination edge degree of OSS developers. As is evident from Figure 3, the distribution plot shows a common pattern – a slower decay followed by a fast decay, supporting the above notion of a mixed population of developers. For the network formed in period P1 (2012), the crossover from faster decay to slow decay is rather smooth compared to the networks formed over a longer period (P2 to P5). In Figure 3, the distribution of the developer's degree is skewed. However, it doesn't appear to follow a power law distribution. This suggests a non-linear rather than a linear preferential attachment process (Bauer and Kaiser 2017). We will investigate this further using ERGM to understand the relational mechanism that led to coordination edge formation in our OSS coordination network.

## 4.2 ERGM Estimation

Table 4 reports the mean, standard deviations of the variables used in the ERGM models for networks constructed and the number of nodes and edges in our cumulative time periods P1 to P5.

Table 4: Descriptive statistics showing the mean and standard deviation of the endogenous variables.

| | **Mean (Standard deviation)** | | | | |
|---|---|---|---|---|---|
| **Measure** | P1<br>2012 | P2<br>2012-2013 | P3<br>2012-2014 | P4<br>2012-2015 | P5<br>2012-2016 |
| ***Relational mechanism measures*** | | | | | |
| Degree ($k$) | 27.83<br>(27.13) | 64.52<br>(64.94) | 85.64<br>(87.18) | 96.69<br>(103.78) | 103.93<br>(112.91) |
| Number Of Common Functions Among Two Developers ($\Lambda$) | 1.696<br>(2.228) | 1.749<br>(2.818) | 1.692<br>(2.825) | 1.695<br>(3.019) | 1.676<br>(2.918) |
| *Control variables* | | | | | |
| Total Number Of Commits ($C$) | 56.82<br>(139.24) | 64.87<br>(181.89) | 68.93<br>(203.32) | 71.31<br>(226.06) | 72.21<br>(233.19) |
| Lines Of Code Added ($LCA$) | 1084.15<br>(6601.25) | 718.87<br>(4803.51) | 867.60<br>(5723.14) | 816.62<br>(5230.06) | 796.48<br>(5019.59) |
| Lines Of Code Removed ($LCR$) | 981.28<br>(6782.39) | 718.87<br>(4803.51) | 867.60<br>(5723.14) | 816.62<br>(5230.06) | 796.48<br>(5019.59) |
| Developer Tenure ($\Gamma$) | 51.92<br>(67.27) | 78.51<br>(111.38) | 116.59<br>(172.24) | 141.5<br>(203.06) | 152.06<br>(217.9) |
| Developer Code Quality ($DCQ$) | 41.07<br>(14.89) | 38.58<br>(17.03) | 38.82<br>(17.22) | 38.29<br>(17.89) | 39.38<br>(18.08) |
| **Network size** | | | | | |
| Number of Nodes | 144 | 331 | 445 | 561 | 619 |
| Number of Edges | 2004 | 10710 | 19055 | 27123 | 32167 |

As expected, the number of nodes and number of edges increased from 144 to 619 as we increased the cumulative time period. Further, the mean degree $k$ also increases. Interestingly, the average number of common functions ($\Lambda$) among pairs of developers remains almost constant in time. Due to the cumulative nature of the coordination network, the average number of commits ($C$), and the average developer tenure ($\Gamma$) increase with time. However, the average number of lines of code added ($LCA$) and the lines of code removed (($LCR$) decreases with time. This reflects the greater stability of the technical structure at the early stages of software evolution. The average developer code quality (measured with the developer's code Maintainability Index (MI)) is relatively stable over time, and even decreases despite the significant growth in the social and the technical structure. These observations suggest that as the coordination network increases, the number of coordination edges increases while the code quality decreases (potentially because of greater coordination).

We next turn to the statistical estimation of the ERGM models. We specified and estimated an ERGM for the five different evolutionary periods from 2012 to 2016. In Table 5 we present the final models that include both the controls as well as the endogenous variables to examine the role of relational mechanisms. We report the effect size and the statistical significance of the co-efficients $\theta_1, \theta_2, \dots\theta_8$ for the endogenous and the exogenous predictors to explain to the formation of a coordination edge in the OSS coordination network in Nova at different evolutionary periods. We used five models (M1 to M5) for the five different evolutionary periods from 2012 to 2016 (M 1: 2012, M2: 2012-2013, M3: 2012-2014, M4: 2012-2015, M5: 2012-2016).

The ERGM estimates were obtained from MCMC maximum likelihood estimation (MCMCMLE) procedure, which follows the Metropolis-Hastings algorithm. The MCMC algorithm simulates a Markov chain of networks such that the probability of observing a chain in a particular state after a large number of iterations is approximated by a parameter value. In this, we estimate the network statistics that explain the link formation of developer-coordination networks. These network statistics control the overall probability of a link in the developer coordination networks. For all the networks, our final model of interest explains the endogenous effects (non-randomness ($E$), preferential attachment ($k$), and knowledge similarity (*Edgecov ($\Lambda$))* of interest for this study, while also controlling for exogenous effects (developer commits (C), developer's lines of code added (LCA), and developer's lines of code removed (LCR), and developer's code quality (DCQ)).

Table 5: ERGM estimates of the networks with node attributes and graph theoretical properties

| | **M1: 2012** | **M2: 2012-2013** | **M3: 2012-2014** | **M4: 2012-2015** | **M5: 2012-2016** |
|---|---|---|---|---|---|
| ***Relational mechanisms (endogenous effects)*** | | | | | |
| $E$ | -1.513*** | -0.984*** | -0.916*** | -0.812*** | -1.030*** |
| | (0.0027) | (0.0021) | (0.0019) | (0.0018) | (0.002) |
| $k$ | 0.007*** | 0.004*** | 0.0002*** | 0.0014*** | 0.0019*** |
| | (0.0008) | (0.0001) | (0.00007) | (0.00004) | (0.00005) |
| *Edgecov(Λ)* | 0.031*** | 0.019* | 0.012 | 0.015** | 0.0022 |
| | (0.0012) | (0.0078) | (0.0063) | (0.0046) | (0.0047) |
| ***Control variables (exogenous effects)*** | | | | | |
| $C$ | -0.0025*** | 7.84x10$^{-05}$ | -0.00074*** | -9.5x10$^{-05}$* | 1.25x10$^{-05}$ |
| | (0.00022) | (7.08x10$^{-05}$) | (0.000045) | (3.94x10$^{-05}$) | (3.93x10$^{-05}$) |
| $LCA$ | 1.8x10$^{-05}$ | -4.2x10$^{-05}$*** | 1.9x10$^{-05}$*** | -1.3x10$^{-05}$*** | 3.66x10$^{-05}$*** |
| | (1.62x10$^{-05}$) | (8.33x10$^{-06}$) | (2.18x10$^{-06}$) | (3.10x10$^{-06}$) | (4.4x10$^{-06}$) |
| $LCR$ | 1.3x10$^{-05}$ | 1.98x10$^{-06}$ | 1.1x10$^{-05}$*** | -9.3x10$^{-06}$*** | -5.3x10$^{-05}$*** |
| | (1.58x10$^{-05}$) | (7.55x10$^{-06}$) | (1.47x10$^{-06}$) | (2.09x10$^{-06}$) | (4.65x10$^{-06}$) |
| $\Gamma$ | 0.0033*** | 0.00046*** | 0.00054*** | 4.86x10$^{-5}$ | 1.28x10$^{-04}$*** |
| | (0.00026) | (0.00006) | (0.000035) | (2.65x10$^{-05}$) | (2.11x10$^{-05}$) |
| $DCQ$ | 0.012*** | -0.00098*** | -0.0011*** | -0.0033*** | -2.09x10$^{-05}$ |
| | (0.00056) | (0.0003) | (0.0002) | (1.76x10$^{-04}$) | (1.54x10$^{-04}$) |
| *AIC* | 9803 | 52624 | 95071 | 142734 | 169714 |
| *BIC* | 9745 | 52696 | 95147 | 142813 | 169796 |

(*p<0.05, **p<0.01, ***p<0.001)

To examine the model fit, we compared the Bayesian Information Criterion (BIC) values of ERGMs with control variables only and ERGMs with control variables and relational mechanisms. This allows us to examine the explanatory power of the three measures – randomness (E), preferential attachment, and knowledge similarity $\Lambda$ – over and above the control variables. The BIC values for the full models (M1 to M5) are significantly lower than the models containing the control variables only: Across all time periods, the drop in BIC is greater than 10, suggesting better fit to the data. For M1, the BIC drops from 10519 to 974, for M2, from 55694 to 52696, for M3, from 99716 to 95147, from M4, from 148781 to 142813, for M5, from 180267 to 169795. An inspection of the Akike Information Criterion (AIC) values shows similar model improvements.

We briefly discuss the results on the control variables, before discussing each of the three relational mechanisms individually. For each metric, we discuss their effects across the different evolutionary stages of the OSS coordination network.

*Control variables*: We included five control variables in our model: developer commits (C), developer lines of code added (LCA), developer lines of code removed (LCR), developer tenure ($\Gamma$ ), and developer code quality (DCQ). The log-odds of developer's commit (*C*) in predicting developer's ties are negative and significant in M1, M3, and M4, while in M2 and M5 the values are positive but not significant.  The commit effort seems to be associated with a greater likelihood of time formation, but the effects do not sustain. The effects of *LCA* and *LCR* are not statistically significant in M1, but become significant in subsequent models, suggesting that greater contribution effort leads to also to greater coordination awareness among two developers. For the developer's tenure ($\Gamma$) the log-odds are positive and significant effect, with an exception for M4. The coefficient of developer's code quality (*DCQ*) is positive and significant in M1, but turns negative and significant in M2, M3, M4. This suggests that if developers who work on files with a high code quality, they are less likely to coordinate their action. In such cases, there is simply no reason to coordinate as functional dependencies have already been resolved.

*Non-randomness*: For the developer coordination network in 2012 we observe that the coefficient for non-randomness (*E*) is negative and significant. It provides statistical evidence for our hypothesis that the developers do not coordinate randomly while committing on function codes. The edges term (*E*) refers to the number of coordination edges in the network. The log-odds of two developers having an edge is *-1.513* (*p<0.0001*), the corresponding probability being $\exp(-1.513)/(1+\exp(-1.513)) = 0.180$. There is a probability of 18% that two developers are forming a coordination link among each other (compared to a random chance of 50%). The power and significance of the non-randomness term is consistent across models M2 through M5, suggesting that the formation of the OSS coordination network over longer evolutionary stages is also non-random.

*Preferential attachment*: The degree term (*k*) represents the frequency distribution for nodal degrees and empirically suggests if preferential attachment mechanism influences link formation among developers. In M1, the effect of degree is positive and statistically significant (*p<0.0001*). In Model M1, the log-odds of forming an edge is 0.006, the corresponding probability of two high degree developers forming an edge is $\exp(0.006)/(1+\exp(0.006)) = 0.501$. The strength and significance of the degree term is consistent across models spanning a longer time period (M2 – M5), suggesting that the evolution of the OSS coordination network emerges from mechanism of preferential attachment. However, the consistent low values of log-odds of the degree term in predicting edge formation among developers hints at a non-linear preferential attachment mechanism rather than a linear preferential attachment. This supports our explorative results discussed in section 4.1. A non-linear preferential attachment implies that only at very high levels of degree, developers "attract" other developers to collaborate with. In other words, extremely highly connected "star" developers influence the coordination and help others by committing to the same functions. Past research analyzing open source softwares as complex networks observed that Gentoo networks display non-linear preferential attachment (Zheng et al.

2008).

*Knowledge similarity*: The effect of *edgecov* term knowledge similarity provides insight into the role of knowledge similarity for coordination edge formation. The log-odd of knowledge similarity ($\Lambda$) in M1 is 0.031 ($p<0.0001$), suggesting that in the early evolutionary stage developers with similar experiences in building software functions, are more likely to coordinate their actions. Apparently, holding common knowledge is one of the reasons why developers coordinate their actions in the early stages of the OSS project. However, we find there are some dynamics as the coordination structure evolves. In M2 the log-odd value drops to 0.019 ($p<0.05$). In M3, the effect of $\Lambda$ loses its significance ($p>0.05$), only to emerge significant in M4 ($p<0.01$) and subsequently losing its significance in the final model M5 ($p>0.05$). The changing effect of knowledge similarity suggests that in the early stages of the OSS software evolution, common knowledge is essential for coordination. However, as the coordination network grows, knowledge similarity loses its importance in explaining coordination while preferential attachment effects sustain.

## 5 DISCUSSIONS

In this study, we empirically examined the evolution of the OSS coordination network and disentangle the role of exogenous relational mechanisms that may potentially explain the formation of coordination relationships in evolving OSS coordination networks: Preferential attachment and knowledge similarity. We specified and empirically tested a series of ERGM models for OSS coordination network formation considering different evolutionary stages of the OSS coordination structure. Our findings make three contributions to the literature on software evolution.

First, our findings advance the understanding of the role of *preferential attachment* for evolving software coordination structures. This provides more rigorous support for claims made in the existing literature in software evolution has about the potential role of preferential attachment for the evolution of software coordination structures (Concas et al. 2007; Joblin et al. 2017). For example, (Joblin et al. 2017) finds skewed distributions of coordination edges and speculates that there is linear preferential attachment in the realms of "the richer get richer" that leads to scale-free networks following a power-law distribution. Using an ERGM model, we not only describe skewed distribution at the network level but also provide statistical evidence about whether and how the micro-level mechanism of preferential attachment leads to the non-random formation of coordination relationships among two developers. We add to prior literature by providing statistical evidence that while controlling for other exogenous and endogenous preferential attachment is an essential mechanism that explains the early as well as the later stages of software coordination evolution. Further, we provide further nuances to the nature of preferential attachment. We find that the preferential attachment mechanism may unfold in a non-linear rather than linear way. In our empirical setting the probability of link formation is approximately 0.5 rather than 1. Thus, we show that non-linear preferential attachment causes the formation of coordination structures that are efficient but do not follow a power-law distribution (Joblin et al. 2017).

Second, we bring new lights to the role of homophily and *knowledge similarity* among OSS developers for  coordinating their action and resolving interdependencies that may emerge "in situ" when advancing the software's functionality (Cataldo and Herbsleb 2013; Faraj and Sproull 2000; Joblin et al. 2017; Scholtes et al. 2016; Wang et al. 2017).  We find that the homophily among two developers with respect to their semantic knowledge is an important relational mechanism for the formation of edges among developers in the early stages of software evolution. In other words, we support prior claims that the similarity among the two developers' semantic knowledge increase the likelihood for them to work on the same part of the software (Wang et al. 2017). However, this effect does not sustain if we consider a larger evolutionary time period as the statistical effect vanishes as the OSS coordination network evolves. The lack of statistical significance over a long evolutionary period could be attributed to the emergence of "specialized" clusters of developers who work on common parts and functions to resolve critical functional interdependencies (Joblin et al. 2017). Once those specialized clusters have emerged, similarity-based coordination activities stabilize, and only preferential attachement causes the coordination across specialized clusters.

Third, our study also makes methodological contributions. On the one hand, we support prior claims about the significance of conceptualizing coordination networks in a fine-grained way to understand the evolution of the coordination structure that lead to greater software quality and maintainability (Baldwin et al. 2014; Cataldo et al. 2009; Joblin et al. 2017). Our analysis of the OSS coordination network of Open Stack Nova suggests that an increase in coordination efforts among pairs of developers who work on common functions is associated with greater software quality, despite the growth of the technical and social structure. To our knowledge, we present the first ERGMs for OSS coordination to statistically distentangle and explain which endogenous mechanisms cause OSS coordination networks to form and evolve (Wang et al. 2017). It sets the stage for other researchers to make contributions about non-random coordination in software development.

Finally, it is also worth mentioning that this study makes a broader contribution to the literature on network formation in coordination structures in digital environments outside of software engineering and OSS. We support prior work that emphasizes the unique nature of coordination and collaboration in an online setting, in which coordination and collaboration may not necessarily form based on linear preferential attachment in the realms of "the richer get richer" (Faraj and Johnson 2010).

## 6 FUTURE RESEARCH

There are multiple routes for future work that can build upon our ERGM modeling and resolve some of its limitations. First, we encourage others to consider additional node-level attributes to understand the non-random formation of coordination relationships. In our model, we only focused on common semantic programming knowledge to constitute "homophily". However, there is a need for examining other properties that constitute the similarity of the two developers. For example, we suggest future research to consider overlapping experiences outside of an OSS project (e.g., common experiences outside of the project), or geographic and gender similarity (May et al. 2019).

Second, future research should focus on the direction of interaction during the process of coordination and construct directed networks. Our network was undirected and so our analysis was unable to capture who initiated the coordination effort: Was it the 'star' coordinator with a larger number of coordination relationships, or the developer in the periphery with little coordination awareness? Using a directed network, we suggest that future versions of our ERGM models should incorporate the network statistics reciprocity to capture the directional nature of coordination (Morris et al. 2008).

Third, we invite other software engineering scholars to study OSS coordination in other OSS projects, that unlike OpenStack – a community with high quality software engineering standards and processes - are dominated by voluntary contributions and lack of use of formal software engineering practices and standards. For example, work in OSS communities like Freenet or R are less formalized in terms of software engineering practices (von Krogh et al. 2003). An ERGM model would bring light to the role of preferential vis-à-vis knowledge similarity for the formation and evolution of loosely organized OSS structures.

To conclude, this research intended to advance our understanding of the formation and evolution of OSS coordination. OSS is growing, and our research also highlights the significant role of a few very powerful developers who are essential for the successful evolution of OSS software. Further, this work offers researchers a new method to disentangle the effect of different endogenous mechanisms for coordination network formation. We intend to inspire others to build and empirically examine revised versions of our ERGM modeling. ERGM models offer explainable insights into OSS coordination that are important for research and practice.

## 7 References

Adams, P. J., Capiluppi, A., and Boldyreff, C. 2009. "Coordination and Productivity Issues in Free Software: The Role of Brooks' Law," in *2009 IEEE International Conference on Software Maintenance*, , September, pp. 319–328. (https://doi.org/10.1109/ICSM.2009.5306308).

Baldwin, C., MacCormack, A., and Rusnak, J. 2014. "Hidden Structure: Using Network Methods to Map System Architecture," *Research Policy* (43:8), pp. 1381–1397. (https://doi.org/10.1016/j.respol.2014.05.004).

Baldwin, C. Y., and Clark, K. B. 2006. "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?," *Management Science* (52:7), pp. 1116–1127. (https://doi.org/10.1287/mnsc.1060.0546).

Barabási, A. L., Jeong, H., Néda, Z., Ravasz, E., Schubert, A., and Vicsek, T. 2002. "Evolution of the Social Network of Scientific Collaborations," *Physica A: Statistical Mechanics and Its Applications* (311:3), pp. 590–614. (https://doi.org/10.1016/S0378-4371(02)00736-7).

Barabási, A.-L., and Albert, R. 1999. "Emergence of Scaling in Random Networks," *Science* (286:5439), American Association for the Advancement of Science, pp. 509–512. (https://doi.org/10.1126/science.286.5439.509).

Bauer, R., and Kaiser, M. 2017. "Nonlinear Growth: An Origin of Hub Organization in Complex Networks," *Royal Society Open Science* (4:3). (https://doi.org/10.1098/rsos.160691).

Bolici, F., Howison, J., and Crowston, K. 2016. "Stigmergic Coordination in FLOSS Development Teams: Integrating Explicit and Implicit Mechanisms," *Cognitive Systems Research* (38), Special Issue of Cognitive Systems Research – Human-Human Stigmergy, pp. 14–22. (https://doi.org/10.1016/j.cogsys.2015.12.003).

Borgatti, S. P., and Everett, M. G. 2000. "Models of Core/Periphery Structures," *Social Networks* (21:4), pp. 375–395. (https://doi.org/10.1016/S0378-8733(99)00019-2).

Cai, K.-Y., and Yin, B.-B. 2009. "Software Execution Processes as an Evolving Complex Network," *Information Sciences* (179:12), Special Section: Web Search, pp. 1903–1928.

(https://doi.org/10.1016/j.ins.2009.01.011).

Cataldo, M., and Herbsleb, J. D. 2013. "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures," *IEEE Transactions on Software Engineering* (39:3), pp. 343–360. (https://doi.org/10.1109/TSE.2012.32).

Cataldo, M., Herbsleb, J. D., and Carley, K. M. 2008. "Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, New York, NY, USA: ACM, pp. 2–11. (https://doi.org/10.1145/1414004.1414008).

Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D. 2009. "Software Dependencies, Work Dependencies, and Their Impact on Failures," *IEEE Transactions on Software Engineering; New York* (35:6), pp. 864–878. (http://dx.doi.org/10.1109/TSE.2009.42).

Chaikalis, T., and Chatzigeorgiou, A. 2015. "Forecasting Java Software Evolution Trends Employing Network Models," *IEEE Transactions on Software Engineering* (41:6), pp. 582–602. (https://doi.org/10.1109/TSE.2014.2381249).

Chakraborty, A., Krichene, H., Inoue, H., and Fujiwara, Y. 2019. "Exponential Random Graph Models for the Japanese Bipartite Network of Banks and Firms," *Journal of Computational Social Science* (2:1), pp. 3–13. (https://doi.org/10.1007/s42001-019-00034-y).

Coleman, D., Ash, D., Lowther, B., and Oman, P. 1994. "Using Metrics to Evaluate Software System Maintainability," *Computer* (27:8), pp. 44–49. (https://doi.org/10.1109/2.303623).

Concas, G., Marchesi, M., Pinna, S., and Serra, N. 2007. "Power-Laws in a Large Object-Oriented Software System," *IEEE Transactions on Software Engineering* (33:10), pp. 687–708. (https://doi.org/10.1109/TSE.2007.1019).

Cranmer, S. J., and Desmarais, B. A. 2011. "Inferential Network Analysis with Exponential Random Graph Models," *Political Analysis* (19:1), pp. 66–86. (https://doi.org/10.1093/pan/mpq037).

Crowston, K., and Howison, J. 2005. "The Social Structure of Free and Open Source Software Development," *First Monday* (10:2). (http://firstmonday.org/ojs/index.php/fm/article/view/1207).

Crowston, K., and Kammerer, E. E. 1998. "Coordination and Collective Mind in Software Requirements Development," *IBM Systems Journal; Armonk* (37:2), pp. 227–245.

Faraj, S., and Johnson, S. L. 2010. "Network Exchange Patterns in Online Communities," *Organization Science* (22:6), pp. 1464–1480. (https://doi.org/10.1287/orsc.1100.0600).

Faraj, S., and Sproull, L. 2000. "Coordinating Expertise in Software Development Teams," *Management Science* (46:12), pp. 1554–1568. (https://doi.org/10.1287/mnsc.46.12.1554.12072).

Fitzgerald, B. 2006. "The Transformation of Open Source Software," *MIS Quarterly* (30:3), pp. 587–598.

Fruchterman, T. M. J., and Reingold, E. M. 1991. "Graph Drawing by Force-Directed Placement," *Software: Practice and Experience* (21:11), pp. 1129–1164.

(https://doi.org/10.1002/spe.4380211102).

git. 2017. “Git- Fast-Version-Control.” (https://git-scm.com/).

Goodreau, S. M. 2007. “Advances in Exponential Random Graph (P*) Models Applied to a Large Social Network,” *Social Networks* (29:2), pp. 231–248. (https://doi.org/10.1016/j.socnet.2006.08.001).

Grewal, R., Lilien, G. L., and Mallapragada, G. 2006. “Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems,” *Management Science* (52:7), pp. 1043–1056. (https://doi.org/10.1287/mnsc.1060.0550).

Hahn, J., Moon, J. Y., and Zhang, C. 2008. “Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties,” *Information Systems Research* (19:3), pp. 369–391. (https://doi.org/10.1287/isre.1080.0192).

Hinds, P., and McGrath, C. 2006. “Structures That Work: Social Structure, Work Structure and Coordination Ease in Geographically Distributed Teams,” in *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW ’06, New York, NY, USA: Association for Computing Machinery, November 4, pp. 343–352. (https://doi.org/10.1145/1180875.1180928).

Hong, Q., Kim, S., Cheung, S. C., and Bird, C. 2011. “Understanding a Developer Social Network and Its Evolution,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, , September, pp. 323–332. (https://doi.org/10.1109/ICSM.2011.6080799).

Howison, J., and Crowston, K. 2014. “Collaboration Through Open Superposition: A Theory of the Open Source Way,” *MIS Quarterly* (38:1), pp. 29-A9.

Joblin, M., Apel, S., and Mauerer, W. 2017. “Evolutionary Trends of Developer Coordination: A Network Approach,” *Empirical Software Engineering* (22:4), pp. 2050–2094. (https://doi.org/10.1007/s10664-016-9478-9).

Joblin, M., Mauerer, W., Apel, S., Siegmund, J., and Riehle, D. 2015. “From Developer Networks to Verified Communities: A Fine-Grained Approach,” in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE ’15, Piscataway, NJ, USA: IEEE Press, pp. 563–573. (http://dl.acm.org/citation.cfm?id=2818754.2818824).

Khalilzadeh, J. 2018. “Demonstration of Exponential Random Graph Models in Tourism Studies: Is Tourism a Means of Global Peace or the Bottom Line?,” *Annals of Tourism Research* (69), pp. 31–41. (https://doi.org/10.1016/j.annals.2017.12.007).

Koch, S., and Schneider, G. 2002. “Effort, Co-Operation and Co-Ordination in an Open Source Software Project: GNOME,” *Information Systems Journal* (12:1), pp. 27–42. (https://doi.org/10.1046/j.1365-2575.2002.00110.x).

Kovalenko, V., Tintarev, N., Pasynkov, E., Bird, C., and Bacchelli, A. 2018. “Does Reviewer Recommendation Help Developers?,” *IEEE Transactions on Software Engineering*, pp. 1–1. (https://doi.org/10.1109/TSE.2018.2868367).

von Krogh, G., Spaeth, S., and Lakhani, K. R. 2003. “Community, Joining, and Specialization in Open Source Software Innovation: A Case Study,” *Research Policy* (32:7), Open Source Software Development, pp. 1217–1241. (https://doi.org/10.1016/S0048-7333(03)00050-7).

Kunegis, J., Blattner, M., and Moser, C. 2013. "Preferential Attachment in Online Networks: Measurement and Explanations," in *Proceedings of the 5th Annual ACM Web Science Conference*, WebSci '13, New York, NY, USA: Association for Computing Machinery, May 2, pp. 205–214. (https://doi.org/10.1145/2464464.2464514).

Liang, L., Yuanzheng, G., and XiaoGang, Q. 2013. "Using Exponential Random Graph (P∗) Models to Generate Social Networks in Artificial Society," in *Proceedings of 2013 IEEE International Conference on Service Operations and Logistics, and Informatics*, , July, pp. 596–601. (https://doi.org/10.1109/SOLI.2013.6611484).

Lindberg, A., Berente, N., Gaskin, J., and Lyytinen, K. 2016. "Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project," *Information Systems Research* (27:4), pp. 751–772. (https://doi.org/10.1287/isre.2016.0673).

Louridas, P., Spinellis, D., and Vlachos, V. 2008. "Power Laws in Software," *ACM Trans. Softw. Eng. Methodol.* (18:1), 2:1-2:26. (https://doi.org/10.1145/1391984.1391986).

Lubbers, M. J., and Snijders, T. A. B. 2007. "A Comparison of Various Approaches to the Exponential Random Graph Model: A Reanalysis of 102 Student Networks in School Classes," *Social Networks* (29:4), pp. 489–507. (https://doi.org/10.1016/j.socnet.2007.03.002).

Madey, G., Freeh, V., and Tynan, R. 2002. "THE OPEN SOURCE SOFTWARE DEVELOPMENT PHENOMENON: AN ANALYSIS BASED ON SOCIAL NETWORK THEORY," in *AMCIS*, p. 247.

Maillart, T., Sornette, D., Spaeth, S., and von Krogh, G. 2008. "Empirical Tests of Zipf's Law Mechanism in Open Source Linux Distribution," *Physical Review Letters* (101:21), p. 218701. (https://doi.org/10.1103/PhysRevLett.101.218701).

May, A., Wachs, J., and Hannák, A. 2019. "Gender Differences in Participation and Reward on Stack Overflow," *Empirical Software Engineering*. (https://doi.org/10.1007/s10664-019-09685-x).

Meneely, A., and Williams, L. 2011. "Socio-Technical Developer Networks: Should We Trust Our Measurements?," in *2011 33rd International Conference on Software Engineering (ICSE)*, , May, pp. 281–290. (https://doi.org/10.1145/1985793.1985832).

Meneely, A., Williams, L., Snipes, W., and Osborne, J. 2008. "Predicting Failures with Developer Networks and Social Network Analysis," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, New York, NY, USA: ACM, pp. 13–23. (https://doi.org/10.1145/1453101.1453106).

Microsoft VS Docs. 2020. "Calculate Code Metrics - Visual Studio (Windows)." (https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values, accessed September 21, 2021).

Mishra, A., and Otaiwi, Z. 2020. "DevOps and Software Quality: A Systematic Mapping," *Computer Science Review* (38), p. 100308. (https://doi.org/10.1016/j.cosrev.2020.100308).

Mishra, B. 2017. *Guidelines on How to Contribute to the Development of OpenStack - IEEE Conference Publication*, IEEE. (https://doi.org/10.1109/ICCNI.2017.8123813).

Morris, M., Handcock, M. S., and Hunter, D. R. 2008. "Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects," *Journal of Statistical Software*

(24:4), pp. 1548–7660.

Myers, C. R. 2003. *Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs*. (https://doi.org/10.1103/PhysRevE.68.046116).

Olivera, F., Goodman, P. S., and Tan, S. S.-L. 2008. “Contribution Behaviors in Distributed Environments,” *Mis Quarterly*, pp. 23–42.

OpenStack. 2016. “Foundation » OpenStack Open Source Cloud Computing Software.” (http://www.openstack.org/foundation/, accessed January 4, 2016).

OpenStack. 2019. “OpenStack Docs: Developer’s Guide.” (https://docs.openstack.org/infra/manual/developers.html#committing-a-change, accessed January 26, 2019).

Openstack. 2019. *OpenStack Services*. (https://www.openstack.org/software/project-navigator/openstack-components#openstack-services).

Potanin, A., Noble, J., Frean, M., and Biddle, R. 2005. “Scale-Free Geometry in OO Programs,” *Commun. ACM* (48:5), pp. 99–103. (https://doi.org/10.1145/1060710.1060716).

Qiqi Jiang, Chuan-Hoo Tan, Choon Ling Sia, and Kwok-Kee Wei. 2019. “Followership in an Open-Source Software Project and Its Significance in Code Reuse,” *MIS Quarterly* (43:4), MIS Quarterly, pp. 1303–1319. (https://doi.org/10.25300/MISQ/2019/14043).

Raymond, E. 1999. “The Cathedral and the Bazaar,” *Knowledge, Technology & Policy* (12:3), pp. 23–49. (https://doi.org/10.1007/s12130-999-1026-0).

Robins, G., Snijders, T., Wang, P., Handcock, M., and Pattison, P. 2007. “Recent Developments in Exponential Random Graph (P*) Models for Social Networks,” *Social Networks* (29:2), Special Section: Advances in Exponential Random Graph (P*) Models, pp. 192–215. (https://doi.org/10.1016/j.socnet.2006.08.003).

Rosado, T., and Bernardino, J. 2014. “An Overview of Openstack Architecture,” in *Proceedings of the 18th International Database Engineering & Applications Symposium on - IDEAS ’14*, Porto, Portugal: ACM Press, pp. 366–367. (https://doi.org/10.1145/2628194.2628195).

Rullani, F., and Haefliger, S. 2013. “The Periphery on Stage: The Intra-Organizational Dynamics in Online Communities of Creation,” *Research Policy* (42), pp. 941–953.

Scholtes, I., Mavrodiev, P., and Schweitzer, F. 2016. “From Aristotle to Ringelmann: A Large-Scale Analysis of Team Productivity and Coordination in Open Source Software Projects,” *Empirical Software Engineering* (21:2), pp. 642–683. (https://doi.org/10.1007/s10664-015-9406-4).

Setia, P., Rajagopalan, B., Sambamurthy, V., and Calantone, R. 2012. “How Peripheral Developers Contribute to Open-Source Software Development,” *Information Systems Research* (23:1), pp. 144–163. (https://doi.org/10.1287/isre.1100.0311).

Shi, Z., Lee, G., and Whinston, A. 2016. “Toward a Better Measure of Business Proximity: Topic Modeling for Industry Intelligence,” *MIS Quarterly* (40:4), pp. 1035–1056.

Simpson, S. L., Hayasaka, S., and Laurienti, P. J. 2011. “Exponential Random Graph Modeling for

Complex Brain Networks," *PLOS ONE* (6:5), p. e20039. (https://doi.org/10.1371/journal.pone.0020039).

Singh, P. V., and Phelps, C. 2012. "Networks, Social Influence, and the Choice among Competing Innovations: Insights from Open Source Software Licenses," *Information Systems Research* (24:3), pp. 539–560.

Singh, P. V., Tan, Y., and Mookerjee, V. 2011. "Network Effects: The Influence of Structural Capital on Open Source Project Success," *MIS Quarterly* (35:4), pp. 813-A7.

Singh, P. V., Tan, Y., and Youn, N. 2011. "A Hidden Markov Model of Developer Learning Dynamics in Open Source Software Projects," *Information Systems Research* (22:4), pp. 790–807.

Sornette, D., Maillart, T., and Ghezzi, G. 2014. "How Much Is the Whole Really More than the Sum of Its Parts? 1 ⊞ 1 = 2.5: Superlinear Productivity in Collective Group Actions," *PLOS ONE* (9:8), p. e103023. (https://doi.org/10.1371/journal.pone.0103023).

Sosa, M. E., Eppinger, S. D., and Rowles, C. M. 2004. "The Misalignment of Product Architecture and Organizational Structure in Complex Product Development," *Management Science* (50:12), pp. 1674–1689. (https://doi.org/10.1287/mnsc.1040.0289).

Teixeira, J. A., and Karsten, H. 2019. "Managing to Release Early, Often and on Time in the OpenStack Software Ecosystem," *Journal of Internet Services and Applications* (10:1), p. 7. (https://doi.org/10.1186/s13174-019-0105-z).

Van Der Pol, J. 2016. *Introduction to Network Modeling Using Exponential Random Graph Models (ERGM)*, , March. (https://hal.archives-ouvertes.fr/hal-01284994, accessed October 10, 2017).

Virtual Machinery. 2019. "Virtual Machinery - Sidebar 4 - MI and MINC - Maintainability Index." (http://www.virtualmachinery.com/sidebar4.htm, accessed September 21, 2021).

Wagstrom, P., and Datta, S. 2014. "Does Latitude Hurt While Longitude Kills? Geographical and Temporal Separation in a Large Scale Software Development Project," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, New York, NY, USA: Association for Computing Machinery, May 31, pp. 199–210. (https://doi.org/10.1145/2568225.2568279).

Wang, H., Hu, W., Qiu, Z., and Du, B. 2017. "Nodes' Evolution Diversity and Link Prediction in Social Networks," *IEEE Transactions on Knowledge and Data Engineering* (29:10), pp. 2263–2274. (https://doi.org/10.1109/TKDE.2017.2728527).

Zanetti, M. S., Scholtes, I., Tessone, C. J., and Schweitzer, F. 2013. "The Rise and Fall of a Central Contributor: Dynamics of Social Organization and Performance in the GENTOO Community," in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, , May, pp. 49–56. (https://doi.org/10.1109/CHASE.2013.6614731).

Zheng, X., Zeng, D., Li, H., and Wang, F. 2008. "Analyzing Open-Source Software Systems as Complex Networks," *Physica A: Statistical Mechanics and Its Applications* (387:24), pp. 6190–6200. (https://doi.org/10.1016/j.physa.2008.06.050).