Dec 12th, 12:00 AM

The Influence of Dependency Networks on Developer Attraction in Open Source Software Ecosystems

Mario Müller
University of Cologne, mario.mueller@wiso.uni-koeln.de

Christoph Rosenkranz
University of Cologne, rosenkranz@wiso.uni-koeln.de

The Influence of Dependency Networks on Developer Attraction in Open Source Software Ecosystems

Completed Research Paper

Mario Müller
University of Cologne
Cologne, Germany
mario.mueller@wiso.uni-koeln.de

Christoph Rosenkranz
University of Cologne
Cologne, Germany
rosenkranz@wiso.uni-koeln.de

Abstract

Open source software projects rely on the continuous attraction of developers and therefore access to the pool of available developer resources. In modern software ecosystems, these projects are related through technical dependencies. In this study, we investigate the influence of these dependencies on a project's ability to attract developers. We develop and test our hypothesis by observing the dependency networks and repository activities of 1832 projects in the JavaScript ecosystem. We find that dependencies to other projects have a positive effect on developer attraction while we did not find an effect of dependencies from other projects. Our study contributes theoretically and practically to the understanding of developer attraction and highlights the role of technical interdependencies in software ecosystems.

Keywords: Open Source Sustainability, Sustained Participation, Developer Attraction, Dependency Networks, Software Ecosystems, Bayesian Multilevel Regression

Introduction

Open source software (OSS) is becoming increasingly important for and within firms and their information technology (IT) infrastructures (Aksulu & Wade, 2010; Crowston et al., 2007; Nagle, 2019). Recent estimates suggest that up to 80-90% of any current software is now based on OSS (Nagle et al., 2020). While OSS has been shown to offer several benefits and advantages to firms compared to proprietary software (Aksulua and Wade 2010), this increasing reliance on OSS comes also with downsides. One key issue is that, in comparison to software developed and maintained by organizations, OSS projects heavily rely on communities of distributed developers (Roberts et al., 2006), and therefore, have to sustain these communities to ensure continuous development (Gamalielsson & Lundell, 2014). Without continuous development, OSS projects become vulnerable to security issues or even risk breaking altogether (Bogart et al., 2016). This issue becomes even more problematic when considering complete *software ecosystems of OSS*, that is, collections of interdependent software components (Decan et al., 2019), where continued maintenance and development are critical not just for a single OSS project but for the ecosystem as a whole (Cox, 2019; Valiev et al., 2018).

However, many OSS projects fail to maintain sustained development activity over time or are abandoned soon after their initiation (Chengalur-Smith et al., 2010; Fang & Neufeld, 2009; Stewart et al., 2006). To avoid these problems and remain viable, the *attraction* and retention of developers in the community has become a major issue in OSS projects (Butler, 2001; Crowston et al., 2003). Several studies have identified various project characteristics and signals that indicate the attractiveness and legitimacy of a project, and thus increase its capabilities to attract developers. For example, the size of a project's

community is an important indicator of a project's legitimacy (Butler, 2001; Chengalur-Smith et al., 2010) as well as the general community activity (Butler, 2001; Setia et al., 2020). Since OSS projects exist in a virtual environment of distributed developers that collaborate via online platforms without traditional hierarchies (Lindberg et al., 2016), one stream of research has investigated the relationships between and affiliations of developers to understand the network-related characteristics that influence developer participation choice. For example, a developer's decision to participate in an OSS project has been shown to be influenced by previous collaborations with the project owner (Hahn et al., 2008). More recently, Maruping et al. (2019) showed that centrality in a communication network reduces the uncertainty of developers about how to contribute to a project and therefore helps in attracting developers.

While these studies focus on the social relationships and affiliations of developers in projects and communities as factors influencing developer attraction, the technical interdependencies between OSS projects largely have been neglected so far. Today, OSS projects are almost always situated in larger software ecosystems of interdependent projects. In software ecosystems, *dependencies* arise through the reuse of established projects, which are available as packaged software components, that are integrated into new OSS projects (Decan et al., 2019), which is a common practice in OSS to reduce cost and time (Haefliger et al., 2008). In modern programming languages, the effort to reuse or publish projects has decreased through the availability of dependency management tools and registries, which resulted in an increase in this practice (Cox, 2019). Hence, first studies have identified these dependencies between projects as a major issue for OSS projects because they need to be carefully managed and monitored to avoid breaking changes or security issues (Bogart et al., 2016; Valiev et al., 2018). Furthermore, when reusing projects available in a software ecosystem, dependent projects rely on the continued development and maintenance of the reused project by its creator. Specifically, this is important for studies of sustainability because now concerns and issues related to the sustained participation not only apply to the focal OSS project (Chengalur-Smith et al., 2010) but also to its dependent projects within the software ecosystem (Valiev et al., 2018). Thus, projects with a central position in the network of dependencies should be able to leverage these dependency relations with other projects to sustain their development by continuously attracting developers. Hence, we ask the following research question: "*What is the influence of a project's technical dependencies in the software ecosystem on its ability to attract developers?*"

To answer our research question, we draw from the resource-based as well as the ecological view on sustainable online communities (Butler, 2001; Chengalur-Smith et al., 2010; X. Wang et al., 2013) and theorize that more connected OSS projects within a software ecosystem (i.e., the dependency network) are able to attract more developers. To test this proposition, we adopt a network perspective and analyze projects in the dependency network of a large OSS ecosystem. Adopting a network perspective allows us to investigate a project's position and embeddedness within the dependency network that emerges from its dependency relations. We observe the dependency network and its projects over a period of one year. With the gathered data, we run a Bayesian multilevel regression model to estimate the effect of a project's embeddedness in the dependency network on its ability to attract developers. We find that upstream dependencies, which are created by the focal project through the reuse of other projects in the ecosystem, increase a project's ability to attract developers. However, we also find evidence that this is not the case for downstream dependencies, which result from other projects being dependent on the focal project. These findings contribute to our understanding of sustained participation in OSS projects, which helps organizations to make informed decisions on the adoption of OSS and thereby avoid challenges resulting from abandoned projects (Chengalur-Smith et al., 2010). From a theoretical point of view, we introduce the importance of technical interdependencies of modern OSS projects to fully understand the dynamics driving sustained participation.

The remainder of this paper is structured as follows. In Section 2, we provide an overview of related work on sustained participation in OSS and dependency networks in software ecosystems. In Section 3, we develop our theoretical model. Section 4 describes our research design. Subsequently, in Section 5, we present the results of our analysis. Finally, we discuss our results, implications, and limitations in Section 6.

Related Work and Theoretical Background

Open Source Software Sustainability and Developer Attraction

OSS projects are usually undertaken by a decentralized community of developers who collaborate via development platforms to produce the software (Fang & Neufeld, 2009). These projects rely on the continuous participation of their community (Roberts et al., 2006; Shah, 2006), which makes the sustainability of their communities essential for the long-term sustainability and success of the whole project (Gamalielsson & Lundell, 2014). Consequently, it is no surprise that *sustained participation* is a major topic in studies on OSS sustainability (Curto-Millet & Corsín Jiménez, 2022).

To be sustainable, OSS projects need to maintain a certain quantity of developers by continuously attracting and retaining developers (Butler, 2001; Crowston et al., 2003). To do so, they require access to the pool of potentially available developer resources in the environment (Butler, 2001; X. Wang et al., 2013), thereby competing with other projects in the ecosystem for a limited number of unpaid, motivated, and skilled developers (Setia et al., 2020; X. Wang et al., 2013). Hence, OSS projects need to demonstrate their attractiveness and legitimacy through project-related characteristics and signals to gain developers' attention (Curto-Millet & Corsín Jiménez, 2022).

Several such characteristics and signals have been investigated in the literature. For instance, sustained development has been shown to keep users interested in a project (Subramaniam et al., 2009); in return, user interest has a positive effect on the development activity (Stewart et al., 2006), which creates a virtuous circle between sustained development and user interest. Moreover, having users interested in the project enhances developers' motivation to participate in a project, which leads to increased development activity (Stewart et al., 2006). It is also well-known that developer pool size influences a project's ability to attract resources in the form of developers in the future (Chengalur-Smith et al., 2010). With an increased developer pool size, the available resources to the project increase and thereby its potential to attract more developers in the future (Bock et al., 2015; Butler, 2001; Chengalur-Smith et al., 2010).

In addition to these characteristics, previous studies have also investigated the influence of relationships between developers and projects that emerge through collaboration among developers. The resulting networks grow and change over time in online communities (Faraj & Johnson, 2011; X. Wang et al., 2013), which changes the underlying mechanisms that drive continuous participation of their members (Bock et al., 2015). To study these emerging networks in the context of OSS development, IS scholars intensively used (social) network theory. For instance, studies have investigated the affiliation networks between projects and developers and their influence on success (Grewal et al., 2006), incentives of network formation (Singh & Tan, 2010), or cooperation between developers (Hahn et al., 2008). Important findings highlight that more central projects have an increase in developer attention (Hahn et al., 2008) and also better access to network knowledge (Mallapragada et al., 2012; Singh et al., 2011) and resources (Sojer & Henkel, 2010). Moreover, projects benefit from their network position in terms of access to resources (Grewal et al., 2006). However, these studies have focused on the relationships and connections that emerge through social interactions between developers and, therefore, ignore the technical interdependencies between OSS projects that emerge in software ecosystems.

Software Ecosystems and Dependency Networks

Software ecosystems are "large collections of interdependent software components that are maintained by large and geographically distributed communities of collaborating contributors" (Decan et al., 2019). A software component, or package, is thereby defined as a "reusable code or set of components that can be included in other applications by using dependency management tools" (Kikas et al., 2017). We refer to those packaged software components as projects through this paper.

The reuse of projects is common practice in OSS development, saves developers time, and helps to implement proven solutions (Haefliger et al., 2008; von Hippel & von Krogh, 2003). The emergence of dependency management tools further facilitated reuse by easing the process of publishing own software

as a reusable dependency and adding dependencies from the software ecosystem to a project (Cox, 2019). As the name implies, adding a project as a dependency to an OSS project makes it dependent on that component (Bogart et al., 2016).

In the context of modular design and component-oriented development, a dependency means that a project cannot function without the other projects it then depends on (Valiev et al., 2018). Considering the software ecosystem and its dependency relations as a whole, we can therefore distinguish between *upstream* and *downstream dependencies*. From the perspective of the focal project, an upstream dependency emerges when the focal project itself adds dependencies to other projects to its codebase (Valiev et al., 2018). Downstream dependencies are therefore the result of other projects depending on the focal projects (Valiev et al., 2018). Taken together, the projects and their dependency relations then form a *dependency network* (Kikas et al., 2017).

Some initial studies in social computing have investigated dependency networks in the context of OSS. For example, Valiev et al. (2018) investigated the influence of a project's position in the dependency network on its general probability of survival. They show that a project's position in the dependency network significantly impacts project survival in the form of sustained project activity and argue that this also influences the project's access to developers in the ecosystem (Valiev et al., 2018). However, they do not investigate the latter hypothesis. Furthermore, having more dependents has been found to increase access to development resources (Sojer & Henkel, 2010; Valiev et al., 2018). However, these studies do not answer the crucial question if the project's position actually helps in attracting developers.

The Influence of Dependencies on Developer Attraction

In this study, we propose that a project's embeddedness in the dependency network, as reflected by upstream and downstream dependencies, influences its ability to attract developers to participate. In summary, we propose that both up- and downstream dependencies increase a project's ability to attract developers. Figure 1 summarizes our research model.

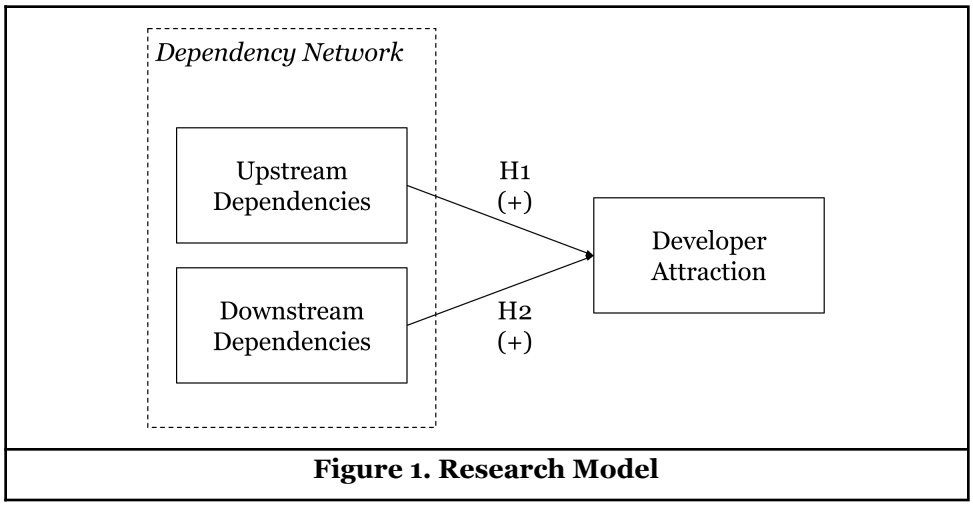


Figure 1. Research Model

From a network perspective, scholars have studied the network embeddedness of an actor, that is its connections with other network actors, which is associated with more access to information and resources, and signaling more prestige (Borgatti & Foster, 2003; Newman, 2018). In the context of OSS projects, it has been shown that a project with higher embeddedness is perceived as more important and signals a higher quality (Grewal et al., 2006; Setia et al., 2020). Hence, these projects should benefit from higher developer attention (Hahn et al., 2008) and better access to network knowledge (Mallapragada et al., 2012; Singh et al., 2011) and resources (Grewal et al., 2006; Sojer & Henkel, 2010). We argue that a project's embeddedness in the dependency network also signals a project's importance in the software ecosystem and grants access to developer resources through its upstream and downstream dependencies.

Upstream dependencies arise when the focal project reuses other projects that have been made available as packaged software components in the software ecosystem (Valiev et al., 2018). The reuse of projects allows developers to save time and build upon proven solutions (Haefliger et al., 2008; von Hippel & von Krogh, 2003). Thus, it enables developers to focus on tasks that they actually like to work on (Haefliger et al., 2008), because basic functionality is mostly provided by the upstream dependency. Furthermore, it decreases a developers perceived participation cost, which influences a developers participation choice (Butler et al., 2014). This is because the number of upstream dependencies indicates that functionality is provided by other projects and therefore parts of the project are maintained by others outside the focal project (Haefliger et al., 2008). Therefore, we argue that projects with extensive reuse of projects, reflected in a larger number of upstream dependencies, become more attractive for developers. Hence, we propose:

H1: An increase in the number of upstream dependencies increases the attraction of developers.

Downstream dependencies reflect the number of other projects in the ecosystem that use the focal project as a building block (Kikas et al., 2017). Therefore, an increase in downstream dependencies potentially grants the focal project access to the developer resources of the dependent project. These developers, often referred to as 'peripheral developers', are motivated to enhance the focal project for their own use (Setia et al., 2012). They do so not only through bug detection but are also willing to solve issues or create required features on their own (Shah, 2006). For example, they might encounter problems during implementation for which they seek help by opening an issue in the dependencies' repository, find a bug, which they then report, or even propose a fix by creating a pull request (Z. Wang et al., 2020). Moreover, they might also miss certain functionality that is then proposed or implemented. In general, due to the downstream dependency, they are potentially invested in the functioning and survival of the originating OSS projects of the downstream dependencies.

Furthermore, having many downstream dependencies potentially signals that a project is important in the ecosystem because many other projects rely on it. Therefore, the number of downstream dependencies can be seen as an indicator of a project's popularity. Thus, we argue that a higher number of downstream dependencies increases a project's legitimacy and therefore, positively influences the participation decision of developers. Taking all these arguments into consideration, we propose:

H2: An increase in the number of downstream dependencies increases the attraction of developers.

Research Methodology

Data and Network Construction

In our empirical study, we focused on the JavaScript ecosystem because it is one of the largest OSS ecosystems and it is intensively using dependencies (Decan et al., 2019). We leveraged two different data sources to cover both the evolution of the ecosystem's dependency network as well as the development activities related to each project. The meta and dependency data for each project was collected from the npm registry[1]. The development activity data was collected from the related GitHub repositories from October 2020 until January 2022, which resulted in 10,968 hours of event logs. Due to the GitHub API restrictions, we utilized the collected event archives from GHArchive[2]. Unfortunately, some event logs were missing from GHArchive during the observation. In total, missing data accounted for 225 hours of event logs, which is about 2% of the total hours of included event logs. We linked both datasets by matching the repository URLs available in the project metadata with the related repository.

In network studies, setting boundary conditions for the network is important (Marsden, 2005). To reduce the number of projects and thereby make the size of the network feasible for the analysis, we started the network construction by identifying 3000 projects from Libraries.io's list of top ranked projects[3] in March

[1] https://registry.npmjs.org/
[2] http://www.gharchive.org/
[3] https://libraries.io/search?order=desc&platforms=npm&sort=rank

2022. As our boundary specification and sampling strategy, we followed the expanding selection approach by Doreian and Woodard (1992). This approach starts with a fixed set of nodes in the network and adds further nodes linked to the set (Marsden, 2005). In doing so, we started with the 3000 identified projects as our initial set of nodes and added further projects to the set that had been listed as a dependency in any of the initial set's projects between 2019 and 2022. This time restriction allowed us to avoid adding abandoned projects to the network while still having a broad timespan for observations and being able to identify the most relevant projects in the ecosystem. Furthermore, we only included those projects listed as a runtime dependency (which is required to run the software) and excluded projects needed only during the development process. We repeated this process for every newly identified project which allowed us to identify all relevant projects further down the dependency tree. In total, following this approach, we identified and collected data of 12678 projects.

Based on the dependencies of the total amount of identified projects, we constructed the dependency network. We observed the network between January 2021 and January 2022 at the beginning of each quarter and created snapshots of its state at each observation point. This resulted in a total of 5 observation points. Each snapshot was created by first creating an ego-networks for each project (*ego*) by identifying its recent version and collecting its related dependencies (*alters*). Afterward, we combined all created ego-networks into a whole-network, which can be constructed from egocentric network data when egos are densely sampled (Marsden, 2005), which is applicable in our case due to the used sampling technique. Finally, we reduced the networks to their largest components, that is the largest subset of nodes in the network that are connected by one or more paths (Newman, 2018), to focus on the most important and used projects in the ecosystem.

For our analysis, we selected all projects from the reduced network that met our criteria. First, projects needed to be already created at the start of our observation and present in the network in each period. Second, we selected only projects with at least one developer participating in each period to remove abandoned or feature-complete projects without further development activity. Fourth, some projects consist of multiple smaller projects that can be used independently of each by other projects and hence have their own dependencies, while sharing the same repository. To make sure that the observed development activity was related to a specific project, we excluded these projects from our observations. After this process, our final data consisted of 1832 projects. For these projects, we then collected the related development activities from the retrieved event logs.

Variables and Measures

As our dependent variable, we measure *developer attraction* similar to previous studies (e.g., Chengalur-Smith et al., 2010) by comparing all developers that participated in the project in period *t* with those in the previous period *t-1*. Thus, developer attraction is equal to the number of developers that participated in period *t* but not in period *t-1*. In line with earlier work on participation in traditional information system development, we define participation as "the behaviors, assignments, and activities" during the development process (Hartwick & Barki, 1994). Accordingly, we counted as participation writing comments related to issues or pull requests, opening issues or pull requests or changing their status (e.g., from open to closed), and pushing commits to the repository. To do so, we identified and counted the unique user accounts that were involved in any of these activities from the event logs. Thereby, we explicitly excluded user accounts labeled as bots in our data.

Upstream and *downstream dependencies* were measured by analyzing a project's ego network derived from the complete dependency network for each observation snapshot. Each ego network was represented as a directed graph with a node's (project's) outgoing ties representing its upstream dependencies and its incoming ties representing its downstream dependencies. Therefore, we measured the number of upstream and downstream dependencies by calculating the in- and outdegree centrality (Freeman, 1979). This measure can be applied to egocentric networks because the egocentric network for a node by definition includes all alters of ego, which leads to identical measures for both ego- and whole-networks (Marsden, 2002).

We controlled for known factors with an impact on developer attraction (Chengalur-Smith et al., 2010; Crowston et al., 2003; Gamalielsson & Lundell, 2014) such as developer pool size, community interest, release activity, project age, and organizational ownership. *Developer pool size* was counted by collecting all developers that participated in a period applying the same procedure as for developer attraction. *Community interest* was measured as the number of stars given to the project's related repository in a period, which was derived from the event logs. *Release activity* was measured by counting the number of version releases of a project during the observation period. The version history was gathered from the npm data, which includes timestamps for each version release allowing us to count the releases during a specific period. Pre-release versions such as release candidates, beta, and alpha releases were not included. *Project age* was measured by calculating the number of months between the project's creation date and the date of the respective observation. For *organizational ownership*, we constructed a dummy variable indicating if the project's repository is owned by an organizational account.

Moreover, we also considered including the project's license. Previous studies showed that license choice plays a huge role in influencing, for example, development activity (Subramaniam et al., 2009) and also attracting developers (Santos et al., 2013; Stewart et al., 2006). Therefore, we categorized the used licenses in our dataset by their level of restrictiveness (Lerner & Tirole, 2005) and followed Santos et al. (2013) by including also a category for dual licenses. However, we found that 98.1% of the projects in our dataset used a permissive license. Therefore, we did not include the license type as a variable in our analysis. We also considered controlling for user interest operationalized by the number of downloads for each project. However, downloads are highly correlated with the number of downstream dependencies because the more downstream dependencies a project has, the more projects potentially download it regularly. Therefore, we refrained from adding downloads as a measure of user interest to our model. Table 1 provides a summary of our used variables including a brief description of their operationalization and Table 2 reports the related descriptive statistics.

Variable	**Description**
Developer Attraction	The number of developers that participated in the project in period *t* but not in the previous period *t-1*.
Upstream Dependencies	The number of projects the focal project depends on in the observed network in period *t*.
Downstream Dependencies	The number of other projects depending on the focal project in the observed network in period *t*.
Developer Pool Size	The number of developers that participated in the project in period *t*-1.
Community Interest	The number of developers staring the project's repository in period *t*.
Release Activity	The number of version releases of a project in period *t*, excluding release candidates, alpha and beta versions.
Project Age	The number of months from the project's creation date until the end of period *t*.
Organizational Ownership	Dummy variable indicating if the project's repository is owned by an organizational account.

Table 1. Variable Definitions

Variable	Mean	Median	St. Dev.	Min	Max
Developer Attraction	16.40	4	54.38	0	1,073
Upstream Dependencies	4.94	2	7.78	0	89
Downstream Dependencies	5.21	1	18.15	0	428
Developer Pool Size	19.80	6	63.66	1	1,290
Community Interest	98.26	16	824.38	0	66,913
Release Activity	1.53	0	4.20	0	103
Project Age	71.59	72.23	29.48	6.28	132.43
$N = 1832 \times 4$					

Table 2. Descriptive Statistics for Selected Variables

Results

Dependency Network Evolution

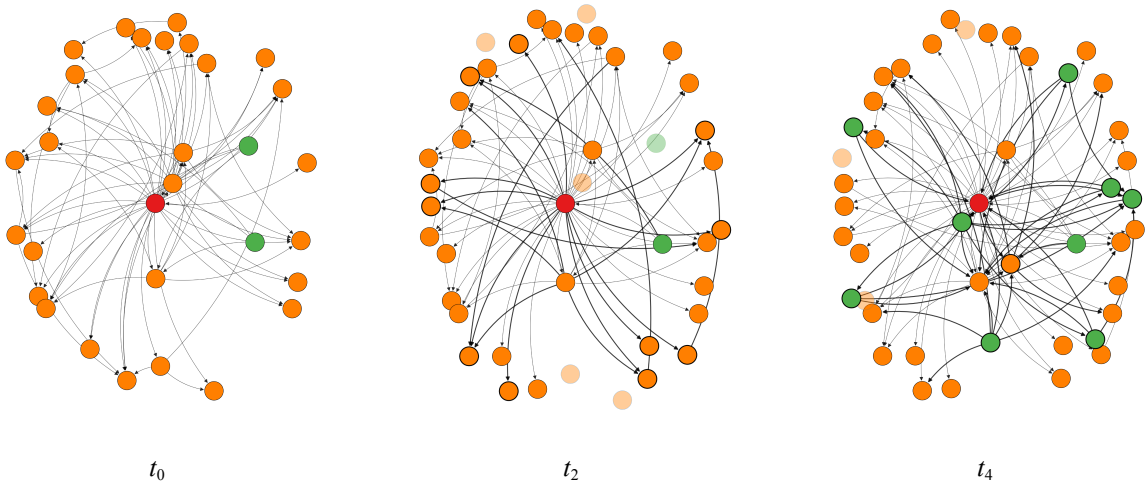
Table 3 provides descriptive statistics for the total observed dependency network. The size of the largest component in the dependency network increases from 10,779 projects in January 2021 to 11,186 at the end of 2021. This shows that many new projects were created that year that depend on another project in the network. Also, even though new projects are introduced into the ecosystem and the number of dependencies increases over time, the network's density, that is, the proportion of possible dependency links actually created and an indicator for the connectedness of the network, constantly decreases and is close to zero. That indicates a sparse network where most of the possible edges are not present (Newman, 2018). Furthermore, the largest dependency tree in the network, which is represented by the network's diameter, is stable overall but in the first observation. Also, the average dependency tree depth for each project, which is the average path length, is relatively stable. The average degree of a project, which is a measure of the average number of dependency links a project is involved in, decreases over time.

	Observ. Date	Nodes	Edges	Density	Network Diameter	Avg. Path Length	Avg. Degree
t_0	2021-01	10,779	35,082	0.000302	14	3.332	6.509
t_1	2021-04	10,900	34,997	0.000294	13	3.283	6.418
t_2	2021-07	10,983	35,119	0.000291	13	3.204	6.395
t_3	2021-10	11,048	35,133	0.000288	13	3.202	6.360
t_4	2022-01	11,186	35,505	0.000284	13	3.192	6.348

Table 3. Descriptive Statistics for Largest Component in Dependency Network

Figure 2 exemplarily shows the ego-network's evolution of the project "ember-auto-import"[4] in the observed dependency network over time. Due to its large size, we refrain from displaying the whole network. The figure includes the project itself, its direct neighbors, and the dependencies between them. To illustrate, in t_4, the network consists of 32 upstream, 9 downstream dependencies, and 96 edges, which is higher than the number of nodes. This is due to ego's dependencies also having dependency relations between each other. Compared to the network state in t_2, 1 upstream dependency was added and 3 removed, and 8 downstream dependencies were added, which resulted in 39 added dependency relations and 11 removed. These changes in upstream and downstream dependencies of the focal project as well as the dependency relationships between its dependencies show how changes related to a single project affect a large number of other projects in the ecosystem.

[4] https://github.com/ef4/ember-auto-import



Note: The networks represent the ego-network of the project "ember-auto-import", colored in red in center position, at three observation points. Orange nodes are upstream- and green nodes are downstream dependencies of ego. Edges between the nodes indicate dependency relations with arrows indicating the direction. Added nodes and edges are highlighted through increased line width. Removed nodes have decreased opacity.

Figure 2. Exemplary Evolution of an Ego-Network

Model Specification

We specified a Bayesian multilevel regression model to estimate the effect of a project's dependencies on developer attraction. A multilevel model allows for clustered data structures, which are present in the case of repeated measurements belonging to the same project (Gelman & Hill, 2007). Thereby, it allows for varying intercepts for each project in our dataset. We choose a negative binomial (gamma-Poisson) posterior distribution because our outcome variable is count data and previous analysis showed overdispersion. This distribution takes this additional variation due to unobserved influences into account (McElreath, 2020). Furthermore, initial analyses showed that all predictors, but project age were not normally distributed and were highly right-skewed. Therefore, we performed log-transformation and added 1 to account for zeros to avoid biased parameter estimates (Gelman & Hill, 2007). The variable for project age was mean-centered.

We used a log-link function, which is the typical link function used for count data (Gelman et al., 2014). For our priors, we used weakly informative priors as suggested by McElreath (2020). Hence, we defined the prior for the intercept as a normal distribution with a mean of 0 and a standard deviation of 10. For all coefficients and the standard deviations, we set the priors to $N(0, 1)$, and for the shape parameter of the gamma-poisson distribution to $\gamma(0.01, 0.01)$. We conducted the estimations using *R* and the *brms* package (Bürkner, 2018). Markov chain Monte Carlo (MCMC) simulations were applied to draw samples from the posterior distribution. We used 4 chains with 4,000 iterations (2,000 warmup, 2,000 sampling) to achieve convergence. All chains converged, were well-mixed ($\hat{R} = 1.00$), and of sufficient size (effective sample size $ESS > 700$). In each model, we found some observations with Pareto k-values > 0.7, which indicates a high influence on the model. Hence, we refitted the models for each influential observation by removing it from the data and checked the models' prediction accuracy using leave-one-out cross-validation (LOO), which is recommended in case of weak priors and influential observations (Vehtari et al., 2017).

During our analysis process, we specified a series of regression models. We started by estimating a baseline model including all control variables and the random effect with a varying intercept to account for our repeated measure data structure. Model 2 adds the independent variables for up- and downstream dependencies to Model 1. In Model 3, we built on Model 2 and additionally accounted for the fact that

some projects are owned by the same individual developer or organization. Hence, we included an additional random effect with varying intercepts for the hierarchical structure of projects being nested within owners to control for unobserved owner heterogeneity. In all models, our dependent variable leads the other independent variables except project age by 1 period to account for reverse causality.

Regression Results

Table 4 shows the correlation matrix for our selected transformed variables as well as the variance-inflation (VIF) scores. All VIF scores are < 2, which indicates that multicollinearity is not a problem, and no corrective measures are required (James et al., 2021). The Pearson correlation indicates a strong positive association between developer attraction and developer pool size in the next period (r = .60), and between developer pool size and community interest (r = 0.73).

	Variable	VIF	1	2	3	4	5	6	7	8
1	Developer Attraction$_{t}$	–	1							
2	Developer Pool Size$_{t-1}$	1.26	.60	1						
3	Community Interest$_{t-1}$	1.11	.42	.73	1					
4	Release Activity$_{t-1}$	1.16	.25	.43	.20	1				
5	Project Age$_{t}$	1.02	.07	.13	.28	-.15	1			
6	Organizational Ownership$_{t}$	1.04	.13	.19	-.00	.15	-.05	1		
7	Upstream Dependencies$_{t-1}$	1.06	.18	.30	.13	.19	.02	.25	1	
8	Downstream Dependencies$_{t-1}$	1.04	-.03	.02	.17	.00	.24	.01	-.22	1

Table 4. Correlation Matrix

In Table 5 we report the results of the estimated models. As indicated by the LOOIC scores, model 3 including the random effect for the hierarchical structure performed best. Therefore, we discuss the results for Model 3 in the following. In general, the estimates for all models are robust.

Our first hypothesis predicted a positive effect of upstream dependencies on developer attraction. The results support this hypothesis with a positive coefficient for upstream dependencies ($\beta = .12^{***}$; 95%-CI: [.10, .15]). When interpreting the coefficient and understanding its effect, it is important to consider that the variable has been log-transformed and we used a log-link function. Thus, the coefficient needs to be interpreted as the percent increase in the dependent variable for every 1% percent in the independent variable. For the coefficient of upstream dependencies, that translates into a 1% increase in upstream dependencies leading to a 0.12% increase in developer attraction.

Based on our second hypothesis, we also expected a positive effect of downstream dependencies. However, we found no effect of the number of downstream dependencies on developer attraction with (β = .00; 95%-CI: [-.01, .02]).

In terms of the control variables, our results confirm the importance of previous variables such as developer pool size ($\beta = .63^{***}$; 95%-CI: [.58, .67]). We also found positive effects for community interest ($\beta = .26^{***}$; 95-CI%: [.24, .29]). Interestingly, compared to previous findings, we found a negative effect of release activity in previous periods ($\beta = -.09^{***}$; 95%-CI: [-.11, -.06]). Also, in line with previous findings, we did not find an effect of a project's age on developer attraction (β = -.00; 95%-CI: [-.00, .00]). Lastly, we found a positive effect of organizational ownership on developer attraction ($\beta = .17^{***}$; 95%-CI: [.11, .22]).

In summary, our results show a positive effect of upstream dependencies on developer attraction supporting H1. Further, we did not find an effect of downstream dependencies on developer attraction. Therefore, H2 is not supported. Figure 3 plots the posterior uncertainty intervals for each variable.

	DV: Developer Attraction$_t$		
	Model 1	***Model 2***	***Model 3***
Fixed Effects			
Intercept	-.58*** (0.03)	-.67*** (0.03)	-.64*** (0.03)
Developer Pool Size$_{t-1}$	.71*** (0.02)	.65*** (0.02)	.63*** (0.02)
Community Interest$_{t-1}$	.24*** (0.01)	.26*** (0.01)	.26*** (0.01)
Release Activity$_{t-1}$	-.09*** (0.01)	-.10*** (0.01)	-.09*** (0.01)
Project Age$_t$	-.00 (0.00)	-.00 (0.00)	-.00 (0.00)
Organizational Ownership	.17*** (0.02)	.14*** (0.02)	.17*** (0.03)
Upstream Dependencies$_{t-1}$	–	.12*** (0.01)	.12*** (0.01)
Downstream Dependencies$_{t-1}$	–	-.00 (0.01)	.00 (0.01)
Random Effects			
σ_{Project}	.28*** (0.02)	.31*** (0.02)	–
$\sigma_{\text{Repository Owner}}$	–	–	.26*** (0.02)
$\sigma_{\text{Repository Owner/Project}}$	–	–	.22*** (0.02)
$N_{\text{Observations}}$	7328	7328	7328
N_{Projects}	1832	1832	1832
$N_{\text{Repository Owners}}$	–	–	1016
LOOIC	39707.33	39518.62	39338.04

Estimation errors in parentheses; * 90%, ** 95%, *** 99% of the confidence interval not including 0.

Table 5. Bayesian Multilevel Model Results

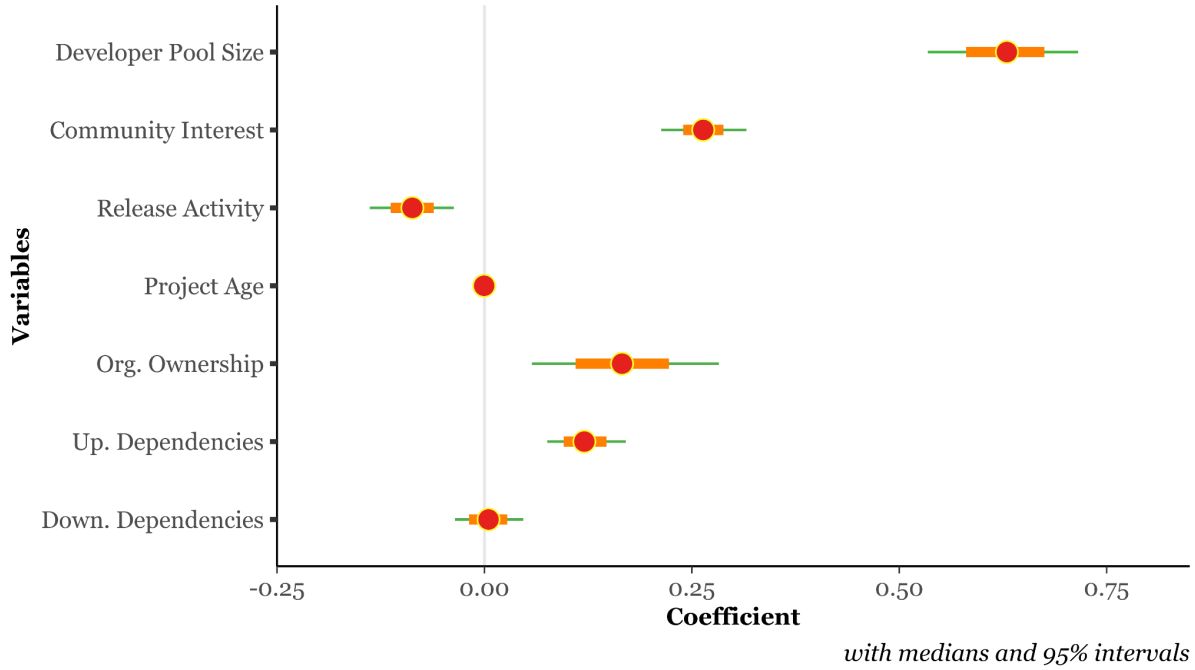


Figure 3. Posterior Uncertainty Intervals

Discussion

Contributions to Research

With this study, we contribute to the literature on OSS project characteristics and signals that lead to attracting developers. The continuous attraction of developers is one of the major issues for OSS projects and is important for their sustainability (Curto-Millet & Corsín Jiménez, 2022). In this study, we investigated how a project's dependencies in a software ecosystem affect its ability to attract developers. We argued that more up- and downstream dependencies increase a project's attractiveness and legitimacy and, therefore, increase its capability to attract developers. We empirically tested these hypotheses using data collected by observing the dependency network and repository activities of 1832 projects in the JavaScript ecosystem.

The results support our hypothesis of a positive effect of upstream dependencies (H1). Prior studies have shown that, contrary to our findings, upstream dependencies have a mixed impact on an OSS project's probability of survival, without an overall significant negative effect (Valiev et al., 2018). Our findings provide reasoning for why this may be the case, as attracting developers may well balance out negative effects such as the creation of potentially more points of failure due to breaking changes. However, despite this positive effect on developer attraction, we should not ignore the negative aspects of intensive use of dependencies. Further studies need to investigate both potentially negative and positive outcomes. In addition, other characteristics of a project's upstream dependencies might play an important role on their effect on developer attraction, such as their development stage, activity level, and popularity. By taking these additional characteristics into account, future research might be able to resolve the reasons for the mixed impact.

Interestingly, we did not find a significant effect for downstream dependencies and therefore no support for hypothesis (H2). This result is worrisome because it highlights the problem of important projects in the ecosystem not benefiting from their downstream dependencies: developers "build on the shoulder of giants", but the foundation may be ignored or forgotten. We thus cannot support the argument that more downstream dependencies lead to higher developer attraction. One possible explanation for our findings could be cognitive biases (Chattopadhyay et al., 2022), which lead developers to prioritize their own OSS projects. Another explanation is that because one of the main reasons developers reuse projects is that they do not have to maintain them (Haefliger et al., 2008), they do not get involved. Therefore, future research could further investigate if developers affiliated with a downstream dependency of a reused project actually do not participate or if the missing effect of the number of downstream dependencies can be explained by other project- or individual-level factors.

Furthermore, our results support prior findings and further highlight the importance of developer pool size in attracting new developers (e.g., Butler, 2001; Chengalur-Smith et al., 2010). However, this indicates that without considerable momentum and a sufficient number of developers in the beginning, it becomes more difficult for an OSS project to further grow its size. Here, as within so many other online contexts, network effects are important. Surprisingly, and contrary to previous studies that suggest a positive effect of activity on attraction (e.g., Butler, 2001; Chengalur-Smith et al., 2010), our results suggest that release activities in previous periods decrease an OSS project's ability to attract new developers in the following period. We suspect that healthy release activities might signal to potential developers a functioning, sustainable, and viable project. Therefore, they might not see the need for their participation to keep the project alive and maintained - in effect creating a detrimental effect. In line with the findings of Chengalur-Smith et al. (2010), we also did not find an effect of a project's age on developer attraction. In addition, our results indicate that organizationally owned projects are able to attract more developers. Organizational ownership has been shown to influence developers' intrinsic as well as extrinsic motivation to participate in a project (Lerner & Tirole, 2002; Spaeth et al., 2015), which is reflected in its ability to attract developers.

We also contribute to studies on OSS projects and the role of the networks based on relationships between and affiliations of developers on sustained participation (e.g., Hahn et al., 2008; Maruping et al., 2019;

Oh & Jeon, 2007; Peng, 2019). We add to this research by focusing on the role of the technical network in the form of technical dependencies between OSS projects) and highlighting the importance of dependency networks in understanding developer participation. Indeed, OSS are socio-technical systems, and both the social network as well as the technical network should be investigated together in future studies.

Contributions to Open Source Software Development Practice

From a practical perspective, we provide several insights for OSS developers and managers. Our study indicates that using dependencies can make a project more attractive to developers. On the downside, OSS projects should not expect increased participation by their downstream dependencies. However, the reuse of projects available as packaged software components also has other benefits and drawbacks besides attracting developers, which we did not investigate. For example, using available projects in the ecosystems saves time, implements a proven solution, and reduces code complexity and maintenance effort. However, all of this comes at the cost of increased dependency management effort, the risk of breaking changes, and potential security risks in terms of poor or stopped maintenance. This may well be dangerous for a project's survivability (Valiev et al., 2018). However, it is not clear if these effects may not balance each other out. Future studies should investigate these potential benefits and drawbacks together.

Limitations

As regards limitations, our study focused on a single software ecosystem, and only a part of the overall ecosystem, due to our sampling strategy. Therefore, results might differ in other ecosystems or even other parts of the JavaScript ecosystem. Furthermore, this study used digital trace data, which implies several potential problems related to validity (Howison et al., 2011), especially with data mined from GitHub (Kalliamvakou et al., 2014). Specifically, due to the amount of required data and API restrictions imposed by GitHub, we relied on the collected event archives by GHArchive. Even though we conducted various reliability checks and followed recommendations to ensure construct validity and avoid temporal mismatch (Howison et al., 2011), we cannot guarantee that the data is completely accurate. However, since our derived constructs are mostly descriptive and the analyzed projects have been carefully sampled, the available data can provide viable information about the projects' environments (Kalliamvakou et al., 2014). The same problem emerges for our second data source, the project metadata from the npm registry. Here, dependencies of a project are usually maintained by the developers themselves. Therefore, dependencies could have been added to a project without actually using it in the code or have been abandoned during the development process without properly removing them from the metadata.

Furthermore, our study focused solely on the impact of dependencies on attraction. However, there are also several negative aspects associated with dependencies, such as an increase in inter-project complexity and coordination, that have damaging effects on sustained participation. Another potentially interesting avenue for future research is the dynamics between the dependency network and the social relations and co-membership of developers that lead to the formation and evolution of the dependency networks and how they shape the overall community.

In addition, our analysis only used the number of dependencies without differentiating between the dependencies' characteristics. These differences might have an effect on their influence on developer attraction. For example, dependencies with a larger developer pool, greater popularity in the community, or more development activity might have greater potential to attract developers. Hence, future research could take these differences into account.

Finally, our measure for developer attraction does not provide information about the actual relation of the developer with the project. Therefore, further studies could investigate the question if participating developers actually have their own projects with a dependency relation with the project. In addition, it might be of interest to find out which actual dependencies or combinations of dependencies increase the attractiveness of the project for potential developers.

Conclusion

To conclude, our study theoretically and practically contributes to our understanding of sustained participation in OSS by introducing and highlighting the role of the underlying technical dependency network in a software ecosystem. We hope that our study motivates others to build on our findings and investigate the various avenues of research that we pointed out. In essence, this requires a true socio-technical lens on OSS ecosystems. Future research should generalize and test our hypotheses in different software ecosystems.

Acknowledgements

We would like to thank Markus Weinmann for his advice and support in the data analysis.

References

Aksulu, A., & Wade, M. R. (2010). A Comprehensive Review and Synthesis of Open Source Research. *Journal of the Association for Information Systems*, *11*(11), 576–656. https://doi.org/10.17705/1jais.00245

Bock, G.-W., Ahuja, M. K., Suh, A., & Yap, L. X. (2015). Sustainability of a Virtual Community: Integrating Individual and Structural Dynamics. *Journal of the Association for Information Systems*, *16*(6), 418–447. https://doi.org/10.17705/1jais.00400

Bogart, C., Kästner, C., Herbsleb, J., & Thung, F. (2016). How to break an API: cost negotiation and community values in three software ecosystems. *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 109–120. https://doi.org/10.1145/2950290.2950325

Borgatti, S. P., & Foster, P. C. (2003). The Network Paradigm in Organizational Research: A Review and Typology. *Journal of Management*, *29*(6), 991–1013. https://doi.org/10.1016/S0149-2063(03)00087-4

Bürkner, P.-C. (2018). Advanced Bayesian Multilevel Modeling with the R Package brms. *The R Journal*, *10*(1), 395–411. https://doi.org/10.32614/RJ-2018-017

Butler, B. S. (2001). Membership Size, Communication Activity, and Sustainability: A Resource-Based Model of Online Social Structures. *Information Systems Research*, *12*(4), 346–362. https://doi.org/10.1287/isre.12.4.346.9703

Butler, B. S., Bateman, P. J., Gray, P. H., & Diamant, E. I. (2014). An Attraction-Selection-Attrition Theory of Online Community Size and Resilience. *MIS Quarterly*, *38*(3), 699–728. https://doi.org/10.25300/MISQ/2014/38.3.04

Chattopadhyay, S., Nelson, N., Au, A., Morales, N., Sanchez, C., Pandita, R., & Sarma, A. (2022). Cognitive biases in software development. *Communications of the ACM*, *65*(4), 115–122. https://doi.org/10.1145/3517217

Chengalur-Smith, I., Sidorova, A., & Daniel, S. (2010). Sustainability of Free/Libre Open Source Projects: A Longitudinal Study. *Journal of the Association for Information Systems*, *11*(11), 657–683. https://doi.org/10.17705/1jais.00244

Cox, R. (2019). Surviving Software Dependencies. *Communications of the ACM*, *62*(9), 36–43. https://doi.org/10.1145/3347446

Crowston, K., Annabi, H., & Howison, J. (2003). Defining Open Source Software Project Success. *Proceedings of the Twenty-Fourth International Conference on Information Systems (ICIS)*.

Crowston, K., Li, Q., Wei, K., Eseryel, U. Y., & Howison, J. (2007). Self-organization of teams for free/libre open source software development. *Information and Software Technology*, *49*(6), 564–575. https://doi.org/10.1016/j.infsof.2007.02.004

Curto-Millet, D., & Corsín Jiménez, A. (2022). The sustainability of open source commons. *European Journal of Information Systems*, 1–19. https://doi.org/10.1080/0960085X.2022.2046516

Decan, A., Mens, T., & Grosjean, P. (2019). An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*, *24*, 381–416.

https://doi.org/10.1007/s10664-017-9589-y
Fang, Y., & Neufeld, D. (2009). Understanding Sustained Participation in Open Source Software Projects. *Journal of Management Information Systems*, *25*(4), 9–50. https://doi.org/10.2753/MIS0742-1222250401
Faraj, S., & Johnson, S. L. (2011). Network Exchange Patterns in Online Communities. *Organization Science*, *22*(6), 1464–1480. https://doi.org/10.1287/orsc.1100.0600
Freeman, L. C. (1979). Centrality in social networks conceptual clarification. *Social Networks*, *1*(3), 215–239. https://doi.org/10.1016/0378-8733(78)90021-7
Gamalielsson, J., & Lundell, B. (2014). Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? *Journal of Systems and Software*, *89*, 128–145. https://doi.org/10.1016/j.jss.2013.11.1077
Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2014). *Bayesian Data Analysis* (3rd ed.). CRC Press.
Gelman, A., & Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.
Grewal, R., Lilien, G. L., & Mallapragada, G. (2006). Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems. *Management Science*, *52*(7), 1043–1056. https://doi.org/10.1287/mnsc.1060.0550
Haefliger, S., von Krogh, G., & Spaeth, S. (2008). Code Reuse in Open Source Software. *Management Science*, *54*(1), 180–193. https://doi.org/10.1287/mnsc.1070.0748
Hahn, J., Moon, J. Y., & Zhang, C. (2008). Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties. *Information Systems Research*, *19*(3), 369–391. https://doi.org/10.1287/isre.1080.0192
Hartwick, J., & Barki, H. (1994). Explaining the Role of User Participation in Information System Use. *Management Science*, *40*(4), 440–465. https://doi.org/10.1287/mnsc.40.4.440
Howison, J., Wiggins, A., & Crowston, K. (2011). Validity Issues in the Use of Social Network Analysis with Digital Trace Data. *Journal of the Association for Information Systems*, *12*(12), 767–797. https://doi.org/10.17705/1jais.00282
James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: With Applications in R*. Springer. https://doi.org/10.1007/978-1-4614-7138-7
Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The Promises and Perils of Mining Github. *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*, 92–101. https://doi.org/10.1145/2597073.2597074
Kikas, R., Gousios, G., Dumas, M., & Pfahl, D. (2017). Structure and Evolution of Package Dependency Networks. *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*. https://doi.org/10.1109/MSR.2017.55
Lerner, J., & Tirole, J. (2002). Some Simple Economics of Open Source. *The Journal of Industrial Economics*, *50*(2), 197–234. https://doi.org/10.1111/1467-6451.00174
Lerner, J., & Tirole, J. (2005). The Scope of Open Source Licensing. *The Journal of Law, Economics, and Organization*, *21*(1), 20–56. https://doi.org/10.1093/jleo/ewi002
Lindberg, A., Berente, N., Gaskin, J., & Lyytinen, K. (2016). Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project. *Information Systems Research*, *27*(4), 751–772. https://doi.org/10.1287/isre.2016.0673
Mallapragada, G., Grewal, R., & Lilien, G. (2012). User-Generated Open Source Products: Founder's Social Capital and Time to Product Release. *Marketing Science*, *31*(3), 474–492. https://doi.org/10.1287/mksc.1110.0690
Marsden, P. V. (2002). Egocentric and sociocentric measures of network centrality. *Social Networks*, *24*(4), 407–422. https://doi.org/10.1016/S0378-8733(02)00016-3
Marsden, P. V. (2005). Recent Developments in Network Measurement. In P. J. Carrington, J. Scott, & S. Wasserman (Eds.), *Models and Methods in Social Network Analysis* (pp. 8–30). Cambridge University Press.
Maruping, L. M., Daniel, S. L., & Cataldo, M. (2019). Developer Centrality and the Impact of Value Congruence and Incongruence on Commitment and Code Contribution Activity in Open Source Software Communities. *MIS Quarterly*, *43*(3), 951–976.

https://doi.org/10.25300/MISQ/2019/13928
McElreath, R. (2020). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan* (2nd ed.). CRC Press.
Nagle, F. (2019). Open Source Software and Firm Productivity. *Management Science*, *65*(3), 1191–1215. https://doi.org/10.1287/mnsc.2017.2977
Nagle, F., Wilkerson, J., Dana, J., & Hoffman, J. L. (2020). *Vulnerabilities in the Core: Preliminary Report and Census II of Open Source Software*. The Linux Foundation & The Laboratory for Innovation Science at Harvard. https://www.coreinfrastructure.org/programs/census-program-ii/
Newman, M. (2018). *Networks* (2nd ed.). Oxford University Press.
Oh, W., & Jeon, S. (2007). Membership Herding and Network Stability in the Open Source Community: The Ising Perspective. *Management Science*, *53*(7), 1086–1101. https://doi.org/10.1287/mnsc.1060.0623
Peng, G. (2019). Co-membership, networks ties, and knowledge flow: An empirical investigation controlling for alternative mechanisms. *Decision Support Systems*, *118*, 83–90. https://doi.org/10.1016/j.dss.2019.01.005
Roberts, J. A., Hann, I.-H., & Slaughter, S. A. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science*, *52*(7), 984–999. https://doi.org/10.1287/mnsc.1060.0554
Santos, C., Kuk, G., Kon, F., & Pearson, J. (2013). The attraction of contributors in free and open source software projects. *The Journal of Strategic Information Systems*, *22*(1), 26–45. https://doi.org/10.1016/j.jsis.2012.07.004
Setia, P., Bayus, B. L., & Rajagopalan, B. (2020). The Takeoff of Open Source Software: A Signaling Perspective Based on Community Activities. *MIS Quarterly*, *44*(3), 1439–1458. https://doi.org/10.25300/MISQ/2020/12576
Setia, P., Rajagopalan, B., Sambamurthy, V., & Calantone, R. (2012). How Peripheral Developers Contribute to Open-Source Software Development. *Information Systems Research*, *23*(1), 144–163. https://doi.org/10.1287/isre.1100.0311
Shah, S. K. (2006). Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science*, *52*(7), 1000–1014. https://doi.org/10.1287/mnsc.1060.0553
Singh, P. V., & Tan, Y. (2010). Developer Heterogeneity and Formation of Communication Networks in Open Source Software Projects. *Journal of Management Information Systems*, *27*(3), 179–210. https://doi.org/10.2753/MIS0742-1222270307
Singh, P. V., Tan, Y., & Mookerjee, V. (2011). Network Effects: The Influence of Structural Capital on Open Source Project Success. *MIS Quarterly*, *35*(4), 813–829. https://doi.org/10.2307/41409962
Sojer, M., & Henkel, J. (2010). Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. *Journal of the Association for Information Systems*, *11*(12), 868–901. https://doi.org/10.17705/1jais.00248
Spaeth, S., von Krogh, G., & He, F. (2015). Research Note—Perceived Firm Attributes and Intrinsic Motivation in Sponsored Open Source Software Projects. *Information Systems Research*, *26*(1), 224–237. https://doi.org/10.1287/isre.2014.0539
Stewart, K. J., Ammeter, A. P., & Maruping, L. M. (2006). Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects. *Information Systems Research*, *17*(2), 126–144. https://doi.org/10.1287/isre.1060.0082
Subramaniam, C., Sen, R., & Nelson, M. L. (2009). Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, *46*(2), 576–585. https://doi.org/10.1016/j.dss.2008.10.005
Valiev, M., Vasilescu, B., & Herbsleb, J. (2018). Ecosystem-Level Determinants of Sustained Activity in Open-Source Projects: A Case Study of the PyPI Ecosystem. *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 644–655. https://doi.org/10.1145/3236024.3236062
Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*(5), 1413–1432. https://doi.org/10.1007/s11222-016-9696-4
von Hippel, E., & von Krogh, G. (2003). Open Source Software and the "Private-Collective" Innovation

Model: Issues for Organization Science. *Organization Science*, *14*(2), 209–223. https://doi.org/10.1287/orsc.14.2.209.14992

Wang, X., Butler, B. S., & Ren, Y. (2013). The Impact of Membership Overlap on Growth: An Ecological Competition View of Online Groups. *Organization Science*, *24*(2), 414–431. https://doi.org/10.1287/orsc.1120.0756

Wang, Z., Feng, Y., Wang, Y., Jones, J. A., & Redmiles, D. (2020). Unveiling Elite Developers' Activities in Open Source Projects. *ACM Transactions on Software Engineering and Methodology*, *29*(3), 1–35. https://doi.org/10.1145/3387111